Universidade do Minho
Escola de Engenharia

Luís Manuel Alves

Empirical Software Engineering in Educational Context

March 2023

Empirical Software Engineering in Educational Context

Luís Manuel Alves

UMinho | 2023

Universidade do Minho
Escola de Engenharia

Luís Manuel Alves

Empirical Software Engineering in Educational Context

Doctoral Thesis
Doctoral Program on Information Systems and Technology

Work done under the guidance of
Professor Doutor Ricardo Jorge Silvério de Magalhães Machado
Professor Doutor Pedro Miguel Gonzalez Abreu Ribeiro

March 2023

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

ACKNOWLEDGMENTS

My life will change after this arduous task. This PhD was a solitary journey into scientific discovery. Yet, at the same time, it was a magnificent window of opportunity to expand my knowledge and to learn with incredible people.

Reach the PhD degree requires a hard work that will not be possible without the contribution of many different people. In a different way all the aid is important in a work of this magnitude. During this work, I had the opportunity to meet people from different countries with research interests similar to mine. Also, in a way, all these people built for this demanding work.

Firstly, I wish to express my sincerest gratitude to my supervisors Professor Ricardo Machado and Professor Pedro Ribeiro for providing me the opportunity to study and conduct research as part of his research team. I would like to highlight their constructive critics, patience and valuable comments and suggestions to my work.

I would also like to thank my colleagues of the SEMAG (Software-based Information Systems Engineering and Management Group), they give me good ideas to continue this research. They also gave me motivation, courage, and strength to complete this complicated task.

I must also thank all the doctoral program's professors by lessons on how to carry out scientific research. I hope I have understood the teachings. The classes they gave me boosted the autonomy needed to carry out this work.

Finally, I would like to thank my family for their support and encouragement that helped me get through university. This work would not have been possible without them, and I hope not to have disappointed.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

RESUMO

Engenharia de Software Empírica em Contexto Educacional

A engenharia de software empírica é uma subárea da engenharia de software que visa a aplicação de teorias e métodos empíricos para a medição, compreensão e melhoria do processo de desenvolvimento de software em empresas de software reais. A experimentação é realizada para nos ajudar a melhor avaliar, prever, entender, controlar e melhorar o processo de desenvolvimento de software e o produto. Os estudos empíricos são importantes na engenharia de software para avaliar novas ferramentas, técnicas, métodos e tecnologias de forma estruturada antes de serem introduzidas no processo de software industrial (real).

A realização de experimentos na área da engenharia de software num contexto industrial é extremamente difícil por diversos motivos. As empresas têm de entregar resultados e são pressionadas por custos, prazos e qualidade. As empresas não podem libertar os colaboradores para fazer experimentos. Consciente desta limitação, esta tese apresenta um ambiente experimental em contexto educacional.

Criar um ambiente industrial em contexto educacional não é uma tarefa fácil. Envolve sinergias do corpo docente, compromisso dos estudantes e disponibilidade de recursos. No entanto, as *soft skills* que os estudantes podem adquirir nesse contexto valem todo o esforço necessário. Esta tese apresenta um ambiente experimental em contexto educacional que permite realizar estudos empíricos usando estudantes como sujeitos. Para além das competências técnicas e tecnológicas, este novo ambiente, potencia as competências de comunicação, trabalho em equipa, gestão e engenharia aos estudantes envolvidos.

Esta tese apresenta também, um *framework* que guia o processo experimental em contexto educacional. Este *framework* também permite classificar os estudos empíricos realizados pelos estudantes. O novo *framework* surge de uma adaptação do *framework* de experimentação de Basili dirigido a contextos industriais.

À luz deste *framework* foram classificados dezenas de projetos realizados por estudantes de diferentes unidades curriculares. Com eventuais ajustamentos que possa sofrer, é entendido que o *framework* é útil, ajustado e pertinente.

Palavras-chave: Engenharia de Software Empírica; Processo de Desenvolvimento de Software; Métricas de Software; Gestão de Projetos de Software.

ABSTRACT

Empirical Software Engineering in Educational Context

Empirical Software Engineering is a sub-field of software engineering which aims at applying empirical theories and methods for measuring, understanding, and improvement of the software development process in real software companies. The experimentation is performed in order to help us better evaluate, predict, understand, control, and improve the software development process and product. Empirical studies are important in software engineering to evaluate new tools, techniques, methods, and technologies in a structured way before they are introduced in the industrial (real) software process.

Conducting experiments in software engineering area in an industrial context is extremely difficult for several reasons. Companies must deliver results and they are pressured by costs, timing, and quality. Companies cannot release their employees to do experiments. Being aware of this situation, this thesis presents an experimental environment in an educational context.

Creating an industrial environment in an educational context is not an easy task. It involves synergies from the teaching staff, student commitment, and resource availability. However, the soft skills that students can acquire in such a context are worth all the effort required. This thesis presents an experimental educational environment that allows realizing empirical studies using students as subjects. In addition to technical and technological skills, this new environment enhances the communication, team working, management and engineering skills of the students involved.

This thesis presents a framework that guides the experimental process in an educational context. This framework also allows classifying the empirical studies carried out by the students. The new framework arises from an adaptation of Basili's experimentation framework aimed at industrial contexts.
In light of this framework, dozens of projects carried out by students from different course units were classified. With any adjustments that may be made, it is understood that the framework is useful, adjusted, and relevant.

Keywords: Empirical Software Engineering; Software Process Development; Software Metrics; Software Project Management.

Contents

Acronyms

ADIS Analysis and Design of Information Systems

ASD Agile Software Development

CMMI Capability Maturity Model Integration

CMMI-DEV Capability Maturity Model Integration for Development

CS Computer Science

DCA Development of Computer Applications

ECTS European Credit Transfer and Accumulation System

EF Experience Factory

ES Empirical Study

ESE Empirical Software Engineering

ESWS Empirical Study With Students

FPA Function Points Analysis

FPs Function Points

GQM Goal/Question/Metric

IDE Integrated Development Environment

IFPUG International Function Point Users Group

IID Iterative and Incremental Development

IS Information Systems

IT Information and Technology

KA Knowledge Area

ML Maturity Level

PRINCE2 Projects in a Controlled Environment

PMBOK Project Management Body of Knowledge

PMI Project Management Institute

PMIS Project Management of Information Systems

QIP Quality Improvement Paradigm

RUP Rational Unified Process

SE Software Engineering

SPM Software Process and Methodologies

SWEBOK Software Engineering Body of Knowledge

UML Unified Modeling Language

List of Figures

List of Tables

Chapter 1
Introduction

1.1. Scientific Background

In this chapter, the scientific background of the research, its focus, research questions and the structure of the thesis are presented. The background of the research includes an introduction to the research area and the motivation behind the research.

In the early nineties, Basili introduced, for the first time, the concept of *Experience Factory*. As the authors refer in (Basili *et al.*, 1992) the concept was introduced to "institutionalize the collective learning of the organization that is at the root of continual improvement and competitive advantage". Thus, the *Experience Factory* provides an organizational schema for collecting experiences on reuse of empirical results, for analyzing them and generalizing the knowledge contained (Visaggio, 2008). This scheme was designed based on many years of the Software Engineering Laboratory (SEL) work. Over several years, this well-known laboratory has conducted several studies and experiments for the purpose of understanding, assessing, and improving software and software processes within a production software development environment at the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) (Basili *et al.*, 1992).

With our approach, we do not intend to create a new software engineering laboratory. Instead, we created an educational experimental environment that allows us to conduct empirical studies in the software engineering area by involving students that are enrolled in our current software engineering courses (undergraduate (BA/BSc), graduate (MA/MSc) and doctorate (PhD) degrees). Our goal was to create a platform that allows us to perform empirical evaluations of the tools, techniques, methods, models, and technologies used in software engineering.

Unlike other mature disciplines, the field of software engineering continues to lack a research and development infrastructure that supports systematic testing of novel software engineering methodologies.

With our new experience factory approach based on one explicit educational environment we try to mitigate this gap.

In the paper *Experimentation in Software Engineering*, Basili *et al.* presented the first framework for analyzing most of the experimental work performed in software engineering over the past several years (Basili, Selby, & Hutchens, 1986). With the work published in that paper, Basili *et al.* had three overall goals: (1) describe a framework for experimentation in software engineering. This framework is intended to help structure the experimental process and provide a classification scheme for understanding and evaluating experimental studies; (2) classify and discuss a variety of experiments from literature according to the framework; (3) identify problems areas and lessons learned in experimentation in software engineering.

References to research works that use or adapt the Basili experimentation framework that can be found in the literature are scarce. The few that exist refer research in a real context involving industrial settings or real-word projects (Bourque & Abran, 1996; Bourque & Côté, 1991; Hayes, 2002; Hayes & Dekhtyar, 2005).

With the motivation of knowing that there is no experimentation framework dedicated exclusively to the educational context, we decided to adapt Basili's experimentation framework for this context. After this, we use it to classify the project works carried out by students of courses in the Information Technology area to support empirical research in software engineering. As in all areas of science and engineering, empirical research can only be considered rigorous when it is conducted using a valid experimental approach or protocol. In this sense, our new framework of experimentation was used to discipline the experimentation process, both for the students involved in the projects and for the researcher during the development of the empirical studies.

The two main motivations for to create an educational experimental environment were knowing that: (1) the lack of preparation of Software Engineering graduates for a professional career is a common complaint raised by industry practitioners; (2) empirical studies are important in software engineering to evaluate new tools, techniques, methods, and technologies in a structured way before they are introduced in the industrial (real) software process. Based on these two triggers, we built an experimental environment that allows us to train students for industry, by involving them with real clients within the development of software projects.

In this educational experimental environment were concretized three real concrete cases (real cases) of research in software engineering topics. The first one, it was applied the Use Case Point (UCP) method to estimate the size of the software projects developed by our graduate and undergraduate students enrolled in Information Technology courses. The second, it was applied the Function Point Analysist method to estimate

the size and complexity of the software projects developed by our students. The third, it was assessed the longevity of risks in software development projects. All three real cases involved research in the software engineering area, namely, software metrics and project management. The experimentation framework was used to classify these three real cases and to discipline and to support the experimentation process. For validation of our framework, we collected all projects carried out in the PMIS (Project Management of Information Systems) unit course in the period of 2010/2011 up to 2017/2018. In this period, 105 students enrolled in the PMIS course developed 79 projects.

1.2. Methodological Context

To create an industrial environment in an educational context is not an easy task. It involves synergies from the teaching staff, student commitment, and resource availability. However, the soft skills that students can acquire in such context are worth all the effort required. This experimental environment allows us to realize empirical studies using students as subjects.

Empirical studies are important in software engineering to evaluate new tools, techniques, methods, and technologies in a structured way before they are introduced in the industrial (real) software process. Empirical Software Engineering (ESE) is a sub-field of software engineering which aims at applying empirical theories and methods for the measuring, understanding, and improvement of the software development process in real software companies (Jaccheri & Osterlie, 2005). This definition extends the concept for ESE proposed by Basili *et al.*, when they said that "experimentation is performed in order to help us better evaluate, predict, understand, control, and improve the software development process and product" (Basili *et al.*, 1986).

Currently, some universities offer courses in the Software Engineering area, as is the case of Norwegian University of Science and Technology (NTNU) (Jaccheri & Osterlie, 2005; NTNU, 2021). These courses have one assessment element that is a software development project. Participation in project meetings with supervisors, develops workshop and a presentation related to the project are mandatory. These projects work as an empirical study where students develop, in addition to technical and technological skills, also soft skills.

Another university that has a research group in the Software Engineering area is the Lund University in Sweden (SERG, 2021) . This university has a Software Engineering Research Group (SERG) known for conducting relevant and rigorous research on the large-scale engineering of software systems, which have significant impact on industry and society. They use, particularly, empirical research methodology for long-

term industry-academia collaboration. These two institutions have worked with students as subjects of experiments. These institutions run the experiments out of the courses' context, whereas in our approach the students perform the experiments as part of their regular academic courses.

The Department of Computer Science of the University of Helsinki created an experimental software laboratory for basic and applied software development research and education. The name of this laboratory is *Software Factory* and they involve researchers, students, and industry partners in their projects (Software Factory, 2021). This infrastructure allows students to use state-of-the-art tools, modern processes, and best practices to prototype and develop great software for businesses in an environment made to support top-level research.

Fraunhofer Institute for Experimental Software Engineering (Fraunhofer IESE) is one of the founding organizations of Empirical Software Engineering (IESE, 2021). Fraunhofer IESE (Fraunhofer, 2021) is one of the founders and drivers of the International Software Engineering Research Network (ISERN), a unique network of more than 45 international software engineering research groups, Fraunhofer IESE have access to all relevant research groups and experts world-wide in ESE area. Fraunhofer IESE is one of allied institutes of the Department of Computer Science of the Kaiserslautern University. Fraunhofer IESE's mission is to promote experimental software engineering, the best approach for introducing engineering style rigor into business practice (Rombach, 2000).

Simula Research Laboratory is another research institution that works closely with the Norway universities. Simula was founded in 2001 and has since become a nationally and internationally acclaimed research institution. Simula's main objective is to create knowledge about fundamental scientific challenges that are of genuine value for society. The strong focus on basic research is combined with both teaching of postgraduate students and the development of commercial applications (Simula, 2021).

Our experimental environment involves students from four curricular units, namely, Software Process and Methodologies (SPM), Development of Computer Applications (DCA), Analysis and Design of Information Systems (ADIS) and Project Management of Information Systems (PMIS). The first two belong to the second year of the master's degree in Engineering and Management of Information Systems (MIEGSI). The last two, ADIS and PMIS belongs to fourth and fifth year of the same course. Our experimental environment also involves PhD students that find a favorable environment for the development of their research. This is a win-to-win approach where all stakeholders win. Our IT students developing a software project requested by real clients. The educational approach is mainly based on Project-Based Learning (PBL) principles. Working in a

team for a limited period of time and delivering a high-quality product are some of the skills that students should gain during their studies at the university.

The lack of professionals in software engineering experiments is due to the conception of high costs and large organizational effort. Usually, companies do not provide their professionals to carry out empirical studies. Port and Klappholz refer that very little empirical research is done in industry and that, for competitive reasons, even when such research is done, its results are often withheld as proprietary or are too difficult to contextualize without divulging sensitive information (Port & Klappholz, 2004).

In the survey conducted in (Sjøberg et al., 2005), a total of 5,488 subjects took part in the 113 experiments investigated, eighty-seven percent were students and nine percent were professionals. This survey demonstrates the importance of using students in this context.

Tichy refers that software students are much closer to the world of software professionals than psychology students are to the general population (Tichy, 2000). In particular, software graduate students are so close to professional status that the differences are marginal. Software graduate students are technically more up to date than the "average" software developer who may not even have a degree in computing. Software professionals, on the other hand, may be better prepared in the application domain and may have learnt to deal with systems and organizations of larger scale than a student.

In our experimental environment, students follow roles and models in IT area very close to real ones. Students have contact with a real enterprise and who challenge them to develop an application that results from a real need. The students collect the software requirements from this enterprise in meetings during the semester. At the end of the semester all teams make a technical presentation of the application developed to the teachers involved and a commercial presentation with the participation of some members of the enterprise. This experimental environment allows the development of different skills in students, which in a teaching classic way would be difficult to achieve.

1.3. Goals and Research Strategy

This thesis aims to contribute for the empirical software engineering knowledge in educational context. For this, a conceptual model of experimentation (framework of experimentation) was developed, and an experimental academic environment was created.

The research questions are of major importance, as stated by Yin "Defining the research questions is probably the most important step to be taken in a research study, (...)" (Yin, 2014). The demonstration cases and the thesis as a whole aim to provide understanding on answers to the following research questions:

- How to create an educational environment to support empirical research in software engineering?
- How to Conduct Empirical Software Engineering in educational context?

These are the main research questions, however, in each of the demonstration cases specific research questions are referred to.

The goals of this thesis are:

- **Goal 1**: Design one educational experimental environment to support empirical research in Software Engineering.

  With this goal, it is intended to characterize and articulate an educational environment based on curricular units of courses in Information Technology area to support empirical research in software engineering, simulating business contexts of software development. It is intended that the set of curricular units involved is aligned with the nature of the topics under research.

- **Goal 2**: Evolve Basili's experimentation framework to be adopted in educational environment.

  With this goal, we intend to extend the framework originally created by Victor Basili in order to use it in project work carried out by students of courses in the Information Technology area to support empirical research in software engineering. It is also intended to consolidate the framework through the analysis of projects carried out by students in the software engineering area.

- **Goal 3**: Demonstrate the adoption of the educational experimental environment in empirical research of the software process topics.

  With this goal, it is intended to concretize concrete cases (real cases) of research in software engineering topics that demonstrate how the educational environment and the experimentation framework support empirical research approaches. It is intended to use real contexts of curricular units within the scope of the Integrated Master in Engineering and Management of Information Systems at the School of Engineering of the University of Minho.

According to Robert Yin, "a case study investigates a contemporary phenomenon (the "case") in its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident" (Yin, 2014). The case study lends itself to investigations of contemporary social phenomena in which

the researcher cannot manipulate relevant behaviors that influence and/or change their object of study. The method allows the researcher to deal with a wide variety of evidence, from document analysis, field visits, interviews, and participatory observation (Yin, 2014). Using this research methodology, we must look to the past, collect the data to make an analysis and then generate results.

By the other hand, in a demonstration case, the researcher participates in the experiment. The experiment itself allows to collect data for a certain purpose. In this research strategy, the research goals are experience oriented. In the research of this thesis, we deal with three educational contexts, the students, the teachers, and the unit courses.

In this thesis, we call demonstration case to the classification of the case study proposed by Yin in which the researcher assumes an explicit role in conducting the work associated with the development of the case; *i.e.*, in the demonstration case, the researcher participates in the case, not assuming the role of a mere observer and therefore not being an independent and disinterested agent in the realization of the case. For all this, this thesis considers a demonstration case research strategy.

1.4. Structure of this Document

The thesis is structured in eight chapters. Each chapter starts with a section of introduction and ending with a section of conclusion; between those, come the sections pertinent to that chapter thematic. Obviously, this does not happen for chapters 1 and 8.

The eight chapters of this document and their main content are:

Chapter 1: Introduction. This chapter introduces the areas of research, the goals and research strategy, and the document structure. The areas of research are empirical software engineering in educational context, frameworks of experimentation and experimental educational environments. All topics are described in greater detail further on in the thesis.

Chapter 2: Empirical Software Engineering. This chapter introduces the origins, concepts, characteristics, and research in the empirical software engineering area. An overview of the main empirical methods collected from the literature is presented. All phases of the experimentation process in software engineering are presented with some detail. The main approaches related to the experimentation process are also presented in this chapter.

Chapter 3: Software Process and Project Management. This chapter presents an overview of the software development processes models. These software processes are classified into traditional, plan-driven, iterative and change-driven, and agile software development models. The concepts and the main project management approaches are also presented in this chapter. This chapter also presents an overview of the software metrics. Special focus is given to Use Case Points and Function Points Analysis metrics.

Chapter 4: Empirical Software Engineering in Teaching. This chapter presents the common characteristics, issues, and challenges in empirical software engineering in an educational context. The advantages and disadvantages of using students as research subjects are described. This is done in opposite to the professionals. The relevance, objectives, and architecture of our experimentation framework is also presented in this chapter. This section ends with a demonstration case using our framework. This chapter also describes the main educational experimental environments found in the world. After this, it is presented our experimental environment and the results obtained from our Project Based Learning approach using this environment.

Chapter 5: Demonstration Case 1 – Use Case Points. This chapter presents the first demonstration case. This case is an empirical study in the software metrics area. After the goals of the demonstration case, a synopsis of the Use Case Points method is presented. In addition to the detailed description of this metric, in this section, it is described an overview of the past effort estimation methods. The analysis and discussion of the Use Case Points method application and results are also presented. In this chapter is also presented the analysis and results of the experimentation framework using this empirical study.

Chapter 6: Demonstration Case 2 – Function Points. This chapter presents the second demonstration case. This case is also an empirical study in the software metrics area. After the goals of the demonstration case, a synopsis of the Function Points Analysis method is presented. In this section, an overview, advantages and flaws, and objectives of this metric is presented. The Function Points Analysis method is described in detail in order to understand the application of the method. The analysis and discussion of the Function Points Analysis method application and results are also presented. In this chapter is also presented the analysis and results of the experimentation framework using this empirical study.

Chapter 7: Demonstration Case 3 – Software Risks. This chapter presents the third demonstration case. This case is an empirical study in the software project management area. After the goals of the demonstration case, a synopsis of risks in software development projects is presented. In this section, it is also presented the software risks collected from the literature, the importance and benefits of risk management, the methodology followed to compare risks and a comparative risk analysis in software projects. In this chapter is also presented the analysis and results of the experimentation framework using this empirical study.

Chapter 8: Conclusions. This chapter presents the research contributions of the work performed. It presents guidelines for future work and research in order to expand and solidify knowledge about Empirical Software Engineering in educational contexts and experimentation frameworks.

Chapter 2

Empirical Software Engineering

2.1. Introduction

In this chapter, an overview of the empirical software engineering is presented. After the introduction, we present the main empirical studies concepts. We also present the main research methods applied in software engineering area. We present some taxonomies to classify empirical studies and an overview of the complete SE (Software Engineering) experimentation process. The chapter ends with some conclusions about progress and challenges related with the Empirical Software Engineering (ESE).

We address some special features of the ESE. ESE is a sub-field of software engineering which aims at applying empirical theories and methods for the measuring, understanding, and improvement of the software development process in real software companies (Jaccheri & Osterlie, 2005). This definition extends the concept for ESE proposed by Basili *et al.*, when they said that "experimentation is performed in order to help us better evaluate, predict, understand, control, and improve the software development process and product" (Basili et al., 1986).

Juristo and Moreno argue that "experimentation refers to matching with facts the suppositions, assumptions, speculations and beliefs that abound in software construction" (Juristo & Moreno, 2001). They refer also that "software construction is supported by and uses a host of ideas: we apply techniques that we trust to output a given result; we believe that so many people will be able to complete project; we expect development time to be shorter using a given tool; we assume that the quality of the final product will be better if we use a particular development process" (Juristo & Moreno, 2001). In this sense, the authors argue that we need to work with facts rather than assumptions in order to make the software engineering a real engineering discipline.

There is a growing interest in empirical study in software engineering, as evidenced by the growing number of publications incorporating empirical methods (Zelkowitz & Wallace, 1998) and increasing investment in empirical research (*e.g.* NSF's CeBASE project (Boehm & Basili, 2002)). Using empirical

studies to study software development under realistic conditions can provide validation for mature technologies, assessing the effectiveness of proposed development tools and methods in various environments and identification of the problems present in less mature technologies.

In the early nineties, the empirical methods applied in software engineering were basically restricted to quantitative studies (mostly controlled experiments). The concept of experimental software engineering has moved to empirical software engineering when a range of qualitative methods have been introduced, from observational to ethnographical studies. In a broad sense, an empirical investigation (synonym of empirical study) is a process that aims to discover something unknown or to validate hypotheses that can be transformed in generally valid laws (Visaggio, 2008). Experimentation in software engineering, as with any other experimental procedure, involves an iteration of a hypothesis and test process.

It is important to be able to evaluate new techniques and methods in a structured way before they are introduced in the software process (Höst, 2002). Empirical methods have gained increased attention in software engineering; there are dedicated conferences such as the International Conference on Evaluation and Assessment in Software Engineering (EASE, 2021), and there are dedicated international journals such as the Empirical Software Engineering (EMSE, 2021).

Software engineering researchers seek better ways to develop, maintain and evaluate software. They are motivated by practical problems, and key objectives of the research are often quality, cost, and timeliness of software products (Shaw, 2002). Generally, Software Engineers present demonstrations of its technologies as a form of evaluation. Demonstrations can provide proof of concepts or incentives to study a question further. However, these demos merely illustrate a potential, therefore depend on the observers' imagination and their willingness to extrapolate. Rarely, demonstrations produce a solid evidence (Tichy, 1998).

Thus, to overcome this lack of evidence and solidify a body of knowledge about the technologies proposed not only based on intuition, judgments or opinions, Software Engineers should observe phenomena, formulate theories, and test them. Experiments are tests of a particular theory. Computer scientists and practitioners defend their lack of experimentation with a wide range of arguments. Some arguments suggest that experimentation is inappropriate, too difficult, useless, and even harmful (Tichy, 1998).

Experimental studies are really expensive and difficult to carry out, they impose the use of additional resources not always easily available on the organization, such as developers, time and hardware/software resources. Carry out this kind of studies is complex in any scientific discipline. Thus, the Software Engineering

research community have no justification to evaluate a much smaller percentage of its proposals than in other areas such as Physics, Medicine, Biology and Psychology.

Although there are some exhaustive experimental studies in the computer science literature, this is not the general rule. The need of experimental rigor in software engineering has already been stressed by authors like Zelkowitz, Sjoberg or Tichy. These three authors base this affirmation on a study of the papers published in several software system-oriented journals. According to Zelkowitz, over 30% of papers had no experimental validation and only 10% of the papers that presented some experimentation followed a formal approach (equivalent to experimentation in other disciplines) (Zelkowitz & Wallace, 1998).

Sjøberg *et al.* analyzed in detail 103 scientific articles published in leading software engineering journals and conferences in the decade from 1993 to 2002 that reported controlled experiments in which individuals or teams performed one or more software engineering tasks. In this study was evaluated 5.453 scientific papers and only 1.9% of the papers had reported the use of experiments to evaluate its proposals (Sjøberg et al., 2005). The main difference between the results obtained in Sjoberg *et al.* and Zelkowitz is that the first and more recent, observe only the use of controlled experiments, while the second author uses a broader definition of experimental study in the selection of the papers, differing in purpose, selection criteria and taxonomies.

A survey of 400 research articles in SE area showed that of those that would require experimental validation, 40% had none, compared to 15% in other disciplines (Tichy, Lukowicz, Prechelt, & Heinz, 1995).

Experimentation is the focal point of the scientific method. The fact that in the field of Software Engineering the subject of inquiry is information rather than energy or matter (in Physics) makes no difference in the applicability of the traditional scientific method (Tichy, 1998). If other disciplines can benefit from the execution of experimental studies, why Software Engineering cannot?

According to Pfleeger, no science can advance without good experimentation and measurement (Pfleeger, 1999). Common wisdom, intuition, speculation, and proofs of concepts are not reliable sources of credible knowledge. On the contrary, progress in any discipline involves building models that can be tested, through empirical study, to check whether the current understanding of the field is correct (Basili, Shull, & Lanubile, 1999).

Thus, there is an increasing understanding in the software engineering community that empirical studies are needed to develop or improve processes, methods, and tools for software development and maintenance (Sjøberg et al., 2005). Another important aspect is that empirical studies have been used to

assess human behavior in multidisciplinary areas such as social sciences. In this context, we can justify the use of experimentation as a means of evaluating the influence of the human factor in software processes, as the software engineering is strongly influenced by human factor (Wohlin et al., 2012; Wood, Daly, Miller, & Roper, 1999).

Tichy presents some arguments traditionally used to reject the usefulness of experimentation in SE area (Tichy, 1998). Table 2.1 shows a summary of fallacies and rebuttals about computer science experimentation collected by that Tichy' s paper.

Table 2.1: Summary of fallacies and rebuttals about computer science experimentation (adapted from (Juristo & Moreno, 2001))

Fallacy	**Rebuttal**
Traditional scientific method is not applicable.	To understand the information process, computer scientists must observe phenomena and formulate and test explanations. This is the scientific method.
The current level of experimentation is good enough.	Relative to other sciences, the data show that computer scientists validate a smaller percentage of their claims.
Experiments cost too much.	Meaningful experiments can fit into small budgets; expensive experiments can be worth more than their cost.
Demonstrations will suffice.	Demos can provide incentives to study a question further. Too often, however, these demos merely illustrate a potential.
There is too much noise in the way.	Fortunately, techniques can be used to simplify variables and answer questions.
Experimentation will slow progress.	Increasing the ratio of papers with meaningful validation has a good chance of accelerating progress.
Technology changes too fast.	If a question becomes irrelevant quickly, it is too narrowly defined and not worth spending a lot of effort on.
You will never get it published.	Smaller steps are still worth publishing because they improve our understanding and raise new questions.

Juristo and Moreno identify other difficulties that preventing the growth of the SE experimentation, namely, (Juristo & Moreno, 2001):

- Software developers are not trained in the importance and meaning of the scientific method. As they are unfamiliar with the scientific method, Software Engineers do not understand the leading role played by experimentation in validating theories and converting them into facts;
- Software developers are unable to easily understand how to analyze the data of an experiment or how they were analyzed by others because they are lacking the (statistical) training;
- A lack of experimental design and analysis books for SE;

- Empirical studies conducted to check the ideas of others are not very publishable;
- The existence of huge number of variables that influence software development;
- It is difficult to get global results in SE. However, it is possible to determine under what circumstances one option is better than another;
- The existence of human factor on software development. SE is not a discipline whose result is independent of practitioners. Different people applying one and the same artefact can yield different results. This can be a strong obstacle to generalizing the results yielded by empirical testing;
- The existence of huge amount of money moved by the software market today. Companies are continuously developing new, increasingly complex and, ultimately, more expensive software systems.

Perhaps some training concerning the scientific method in engineering, would help Software Engineers to realize that the experimentation is important in software construction. We, the research community, need to awareness the SE community that is important debating facts and claims supported by data rather than suppositions and beliefs.

2.2. Empirical Studies Concepts

In this section, we present the main concepts involved in the empirical studies. In order to better understand these concepts, we present their application to an example of experimentation in the software engineering area.

Research in empirical software engineering should aim to acquire general knowledge about which technology (process, method, technique, language, or tool) is useful for whom to conduct which (software engineering) tasks in which environments (Sjøberg et al., 2005). As scientists, we apply scientific investigative techniques to gain more understanding of what makes software "good" and how to make software well (Pfleeger, 1999).

According Basili *et al.*, an empirical study, "in a broad sense, is an act or operation for the purpose of discovering something unknown or of testing a hypothesis, involving an investigator gathering data and performing analysis to determine what the data mean" (Basili et al., 1999). This covers various forms of research strategies, including all forms of experiments, qualitative studies, surveys, and archival analyses. In this thesis, we consider empirical study as synonym of experimental study. However, we distinguish the terms experimental study and experiment. When an experimental study is a controlled experiment or quasi-

experiment, we just call experiment. According Visaggio, an **empirical investigation**, synonym of **empirical study**, "in a broad sense, is a process that aims to discover something unknown or to validate hypotheses that can be transformed in generally valid laws"(Visaggio, 2008).

An **experiment** is a test of a theory about the operation of a system or process, where there are input and output variables. The purpose of an experiment is to provoke controlled and deliberate changes in input variables to check for changes in output variables (see Figure 2.1).

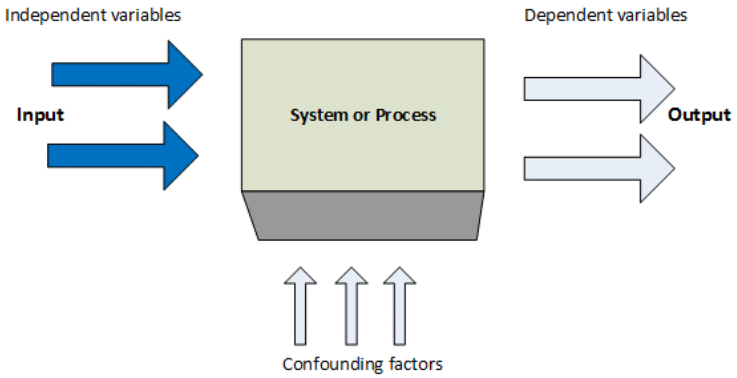


Figure 2.1: General perception of an experiment

A **theory** is a possible explanation of some phenomenon. Any theory consists of a set of hypotheses. A **hypothesis** is a theoretical attempt or an assumption that what we think about it and explains the behavior of that we want to explore. The causes that determine the observed events determine the hypothesis (Visaggio, 2008).

The main hypothesis of an experiment is called **null hypothesis** and states that there is no statistically significant relationship between the cause and the effect. The main goal of the experiment is then reject the null hypothesis.

In one experiment, the input variables are called **independent variables** or **factors**. These are the variables over which the researcher has control, and they are used to cause disturbances in the system or process.

The output variables are called **dependent variables** or **results**. These are the variables that the experiment will measure to check the variations caused in the process or system through the changes caused in the independent variables.

There is also the context or non-controlled variables. These variables are those that, in some way disturb the process or system but for which there is no adequate control. On these variables, we seek to

mitigate its effects by homogenizing samples or groups. These variables are called **confounding factors**. It is hence crucial to try to identify the factors that otherwise may affect the outcome in an undesirable way. These factors are closely related to the threats against the validity of the empirical study (Wohlin, Höst, & Henningsson, 2003). When the effect of one factor cannot be properly distinguished from the effect of another factor, the two factors are confounded. For example, if expert Software Engineers tested tool A and novice Software Engineers tested tool B, we cannot tell if the higher quality software produced by the experts was the result of their experience or of using tool A. Confounding factors can affect the internal validity of the study (Kitchenham, Pickard, & Pfleeger, 1995).

For each **factor** is set a range of values to be assigned during the experiment with the aim to verify the resulting output in the system. Each combination of values of the factors to be tested in the experiment is called **treatment**. An empirical study observes the effect of change on one or more factors. The observations that are carried out with an investigation are consequence of treatments; each treatment is attributed to particular values of each factor. These are expressed through independent variables. The values that the independent variables can assume are called **levels** (Visaggio, 2008).

To better understand the main concepts on an empirical study, we present an illustrative example. For example, we want to compare two tools, X and Y, for programming using object-oriented paradigm and we want to use the M1 and M2 methods to verify the results. In this context, we have:

- Two factors: (1) software tools for programming, and; (2) different methods of performance assessing of a programming tool;
- For the factor, software tools for programming, we have the levels: A and B;
- For the factor, methods of performance assessing of a programming tool, we have the levels: M1 and M2;
- That the following treatments are applicable:
  - Treatment A1: tool X and method M1;
  - Treatment A2: tool X and method M2;
  - Treatment B1: tool Y and method M1;
  - Treatment B2: tool Y and method M2;
- As quality dependent variable we could define the quality of a small or medium software application developed in each of the programming tools and evaluated by each method;
- As context variable, we could indicate the experience of subject in software programming languages.

In addition to the concepts already presented, there are others that are equally important in an empirical study, namely:

- **Objects**, that are the tools used to verify the *cause - effect* relationship existing in an experiment. These objects are the instruments or instrumentation of the experiment;
- **Participants**, that are the individuals selected for composition of a representative sample of a target population for which we wish to generalize the results of the experiment. As greater, the diversity of individuals in the target population, as greater the number of individuals needed to form a representative sample. Are generally selected for convenience in software engineering and divided into groups (or teams). The type of participants can vary from novice students without experience to practitioners experienced in the study domain;
- **Experimental trial**, that is according to the treatments, the combination of objects and participants (or groups). It is also denominated as **internal replication** in (Juristo & Moreno, 2001).

According to the previous example and assuming that, we have 20 participants chosen for convenience, 10 with experience and 10 without experience in software programming languages, we have:

- As objects the initial configuration of the software programming tool A and B and the procedures (steps) to apply M1 and M2 methods (these instructions can be printed in a paper or can be a software tool);
- As experience is a non-controllable variable, we want to mitigate their influence forming two teams (Team 1 and Team 2): each team with 5 experienced participants and 5 without experience. If the experience was an independent variable, we could form two teams, one with participants with experience and other with participants without experience in software programming tools;
- The possible experimental trials:
  - Treatment A1, Team 1 and Treatment B2, Team 2;
  - Treatment A1, Team 2 and Treatment B2, Team 1;
  - Treatment A2, Team 1 and Treatment B1, Team 2;
  - Treatment A2, Team 2 and Treatment B1, Team 1.

2.3. Overview of Empirical Methods

In this section, the empirical research methods applied to experimentation are discussed. We present some taxonomies to classify empirical studies. Finally, we present some overviews of the entire SE experimentation process.

There are two main types of research paradigms having different approaches to running empirical studies. Qualitative research is concerned with studying objects in their natural setting. A qualitative researcher attempts to interpret a phenomenon based on explanations that people bring to them (Denzin & Lincoln, 2011). This type of research is well defined in social sciences. In this context, is a broad term that describes research that focuses on how individuals and groups view and understand the world and construct meaning from their experiences. It is mainly directed to the narrative. Qualitative research begins with accepting that there is a range of different ways of interpretation. It is concerned with discovering causes noticed by the subjects in the study and understanding their view of the problem at hand. The subject is the person, which is taking part in a study in order to evaluate an object (Wohlin et al., 2003).

Qualitative research in SE is usually the same meaning of the social sciences, seeking to focus on understanding the phenomenon of research in its naturally occurring environment. In general, the main techniques used in SE is the observation of participants, interviews, and artefact analysis. The data collected from these experiments are usually composed of text, graphics or even images, etc. Thus, for example, an inquiry to determine why productivity is higher with a new Integrated Development Environment (IDE) and gather data directly from users would be a qualitative study. This study would be concerned with things like the programming language integrated and how users are familiarly with it.

Quantitative research is based on numerical representation of observations for the purpose of describing and explaining a phenomenon. This type of research is mainly concerned with quantifying a relationship or to compare two or more groups (Creswell & Creswell, 2017). Quantitative research aims to get a numerical (quantitative) relationship between several variables or alternatives under examination. Thus, the aim is to identify a *cause - effect* relationship. The quantitative research is often conducted through setting up controlled experiments or collecting data through case studies (Wohlin et al., 2003). Quantitative research begins with the collection of data, followed by the application of various descriptive statistical methods and inference. For example, we would be able to determine how to improve programmer productivity using a new IDE by means of a quantitative investigation. The data collected in this sort of studies are always numerical

values (programmer productivity in this case) to which mathematical methods can be applied to yield formal results.

Qualitative research can be used with quantitative research to achieve a deeper understanding of the causes of a social phenomenon or help to create new research questions. It is possible for qualitative and quantitative research to investigate the same topics but each of them will address a different type of question. For example, a quantitative investigation could be launched to investigate how much a new inspection method decreases the number of faults found in test. To answer questions about the sources of variations between different inspection groups, we need a qualitative investigation. Quantitative research is appropriate when you want to test the effects of a treatment, while a qualitative research is appropriate to find out why the results from a quantitative research are as they are (Wohlin et al., 2003).

The two approaches should be regarded as complementary rather than competitive (Wohlin et al., 2012). Qualitative research is generally considered exploratory and inductive, while quantitative research is usually confirmatory and deductive.

The use of a multi-method approach, i.e., the use of a combination of quantitative and qualitative research methods, provides benefits in terms of more robust conclusions, development and investigation of research hypotheses in an evolutionary manner, and increased understanding of research results (Wood et al., 1999).

Wood *et al.* demonstrated an application of the multi-method approach in an empirical investigation of object-oriented technology. Their research program consisted of a set of structured interviews with practitioners of object-oriented technology, followed by a wide-scale questionnaire survey, and concludes with a set of three controlled laboratory experiments which investigated one of the key findings from the exploratory interview and questionnaire phases (Wood et al., 1999).

The empirical method selection depends of the prerequisites research, its purpose, the availability of resources and how to intends to analyze the collected data (Wohlin et al., 2012).

In SE area, when we think of an experiment, we often think of a roomful of subjects, each being asked to perform some task. The task is usually followed by data collection and then analysis. This is certainly a type of experimentation. But there are other approaches as well, each of which can be grouped into four general categories (Adrion, 1993; Zelkowitz & Wallace, 1998; Zelkowitz, Wallace, & Binkley, 2003):

1. **Scientific method**. Scientists develop a theory to explain a phenomenon; they propose a hypothesis and then test alternative variations of the hypothesis. As they do so, they collect data to verify or refute the claims of the hypothesis.
2. **Engineering method**. Engineers develop and test a solution to a hypothesis. Based upon the results of the test, they improve the solution until it requires no further improvement. It is an evolutionary paradigm.
3. **Empirical method**. A statistical method is proposed as a means to validate a given hypothesis. Unlike the scientific method, there may not be a formal model or theory describing the hypothesis. Data is collected to verify the hypothesis. It is a revolutionary paradigm.
4. **Analytical method**. A formal theory is developed, and results derived from that theory can be compared with empirical observations.

The engineering and empirical methods can be seen as variations of the scientific method since both lead with hypothesis. Traditionally, the analytical method is used in formal areas of Electrical Engineering and Computer Science. The engineering method is probably the dominant method in the industry. The common thread of these methods is the collection of data on either the development process or the product itself. All these methods can be used in software engineering for different purposes.

Some authors propose what they call experimental research methods or technology validation models applied to software engineering. Can we consider method or model such proposals seek to define a taxonomy for categorization the studies performed in software engineering? Thus, we present the taxonomy classification of empirical methods discussed in (Galliers, 1991; Travassos & Barros, 2003; Zelkowitz et al., 2003).

Based on Galliers work, Mandic *et al.* identifies commonly used empirical research methods in software engineering, characterizing them in terms of desirable experimental criteria. According to Mandic *et al.*, the following methods are usually combined: *observational studies*, *pre-experiment studies*, *quasi-experiments*, *controlled experiments*, and *surveys* (Mandić, Markkula, & Oivo, 2009). This empirical research methods are characterized by:

1. **Observational studies** (also known as field, correlational, multiple-case, and single-case studies): in these studies, the researcher does not expose objects or participants in any treatment. Data is usually collected in the field without exposure to any treatment. Data may be collected through visual observation, interviews, data collection forms, or from historical data. In the context of the multi-method approach observational studies may be used to characterize, baseline, and/or identify

relationships. Observational studies are seen as being relatively good for theory generation, have high external validity and are relatively cheap to organize.

2. **Pre-experimental study**: in a pre-experimental study, a treatment is applied to only one case, for example, an organization, a project, a single group of subjects, a unique subject, etc. It is usually conducted in the field and a comparison is made against a baseline. The term 'pre-experimental' is used to denote a study that, does not satisfy the basic criteria of an experiment, rather than being a study conducted before an experiment. Usually, a pre-experimental study is conducted as a research of a new technology introduced in an organization's environment for test how useful is when compared to current practice. For example, evaluating the introduction of object-oriented technology into a pilot project by comparison against a company baseline.
3. **Quasi-experimental study**: in a quasi-experimental study, no random assignment of subjects/objects to treatments. One or more treatments may be applied to two or more groups, direct comparison amongst groups is performed, and usually conducted in field settings. The major restriction is that researchers are often unable to select and randomly assign participants or projects to the conditions. Consequently, representative samples of the study population are not likely to be achieved and comparisons cannot be made based on equivalent groups without some form of study manipulation. Depending on the restrictions, a quasi-experimental study can be conducted under representative conditions where the available level of control is minimal, or in a controlled laboratory conditions. Quasi-experimental studies are characterized as having good internal validity, good potential for theory conformation, and some potential for replication.
4. **Controlled experiment**: in a controlled experiment, a random assignment of subjects, explicit experimental design, with one or more treatments to two or more groups, direct comparison amongst groups is performed. Usually, controlled experiments are conducted in a laboratory setting. All variables that may confound the results should be kept constant through the participants to a high level of control can be achieved. Different effects are compared with subjects randomly assigned to different treatment levels of the independent variables. Controlled experiments provide the best hope for internal validity and theory confirmation. They are also relatively easy to replicate. Additional subjects do not add significantly to the cost of the study.
5. **Survey**: in a survey, some kind of structured sample of subjects is applied. A survey is usually conducted through questionnaires and/or interviews. It is used to identify knowledge of a particular

group of people on an object of study, for example, their views on a particular technology or beliefs about software engineering practices in their organizations. Knowledge elicited can be used to describe a population from sample data or to identify general relationships. Surveys have high external validity, good potential for theory generation, are straightforward to replicate and are relatively cheap to organize.

According Zelkowitz *et al.*, the classification of studies performed in software engineering is through technology validation methods for academia and industry that differ in goals (Zelkowitz et al., 2003). Research methods applied by the research community (academia) can be considered as exploratory, in the researchers' attempts to understand and develop new technology. Industry, on the other hand, wants methods that work, so their techniques are more confirmatory, showing that a given method does indeed have the desired properties. While researchers seek to compare efficiency between technologies, the industry seek to become more competitive (by reducing costs, deadlines, reworking or improves the quality of your process/product), does not always using the most efficient technology.

In (Zelkowitz & Wallace, 1998; Zelkowitz et al., 2003) the empirical research methods are classified in:

1. **Observational methods**: in these methods the data are collected as a project develops. There is relatively little control over the development process unless on the use of new technology being introduced.
2. **Historical methods**: these methods involve an analysis of collected data to discover what happened during the development of a previously developed project. Thus, the data collection is performed on projects that have already been finalized, i.e., the data already exist, only it is necessary to analyze what has been collected.
3. **Controlled methods**: these methods involved careful study of alternative strategies to determine the effectiveness of one method as compared to other methods. This is the more traditional concept when one thinks of an 'experiment'. In these methods are expected statistical validity of the results for multiple instances of an observation. This is the classic method of experimental design in other scientific disciplines.
4. **Formal methods**: these methods involve using a formal model to describe a process. Ultimate validation depends upon using another validation method to determine whether the formal model agrees with reality. The only kind of formal method would be the theoretical analysis.

5. **Informal methods**: these methods are generally ad hoc and do not provide significant results that the technique under study provides the benefits that are claimed.

For each method, there is a subset of types of studies that we can find either in academia or in industry, but their discussion is beyond the scope of this thesis. Table 2.2 shows the types of studies for each of the five methods according (Zelkowitz et al., 2003).

Table 2.2: Validation Methods according to Zelkowitz *et al.* (Zelkowitz *et al.*, 2003)

Method	Academia	Industry
Observational	Case study	Case study
	Project monitoring	Project monitoring
	Field study	Field study
Historical	Legacy data	Legacy data
	Literature search	Literature search
	Lessons learning	Education
	Static analysis	Feature benchmark
		Expert opinions
Controlled	Dynamic analysis	Pilot study
	Replicated experiment	Demonstrator projects
	Simulation	Replicated projects
	Synthetic environment	Synthetic benchmark
Formal	Theoretical analysis	Theoretical analysis
Informal	Assertion	Vendor opinion
	No validation	External

Depending on the purpose of the evaluation, whether it is techniques, methods, or tools, and depending on the conditions for the empirical investigation, Wohlin *et al.* proposes four major different types of empirical research methods (Wohlin et al., 2012; Zelkowitz et al., 2003). These methods are classified in:

1. **Experiment**: they are usually performed in a laboratory environment (*in vitro*) which provides a higher level of control but can be carried out under normal conditions (*in vivo*). Experiments are sometimes referred to as research-in-the-small, since they are concerned with a limited scope and most often are run in a laboratory setting. They are suitable to confirm the theories, to confirm the conventional knowledge, to explore relationships, to assess the predictive models or to validate the measures. The greatest strength of the experiments is in the control over the process and the variables and the possibility to be repeated. When experimenting, subjects are assigned to different treatments at random. The objective is to manipulate one or more variables and control all other variables at fixed levels. The effect of the manipulation is measured and based on this a statistical

analysis can be performed. In some cases, it may be impossible to use true experimentation, in these cases we may have to use quasi-experiments. Quasi-experiments are used when it is impossible to perform random assignment of the subjects to the different treatments.

2. **Case study**: they are used for monitoring projects, activities, or assignments. Case study research is sometimes referred to as research-in-the-typical. It is described in this way due to that normally a case study is conducted studying a real project and hence the situation is 'typical'. Data is collected for a specific purpose throughout the study. Based on the data collection, statistical analyses can be carried out. The case study is normally aimed at tracking a specific attribute or establishing relationships between different attributes. The level of control is lower in a case study than in an experiment. A case study is an observational study while the experiment is a controlled study. The case study is the preferred strategy when the questions "how" and "why" are placed, when the researcher has little or no control over events and when the focus is on a contemporary phenomenon within a real context. Case studies can be complemented by two other types of study: exploratory and descriptive. Regardless of the type of study, researchers must take great care to design and execute a case study in order to overcome the traditional criticism of the method (Yin, 2014).
3. **Survey**: the survey is by referred to as research-in-the-large (and past), since it is possible to send a questionnaire to or interview many people covering whatever target population we have. Thus, a survey is often an investigation performed in retrospect, when, for example, a tool or technique has been in use for a while in an organization and we want to evaluate it in some aspect. The primary means of gathering qualitative or quantitative data are interviews or questionnaires. These are done through taking a sample that is representative from the population to be studied. The results from the survey are then analyzed to derive descriptive and explanatory conclusions. They are then generalized to the population from which the sample was taken. Surveys are very similar to the census, differing primarily in the fact that the surveys typically examine a sample of a population, and a census usually implies an enumeration of the entire population.
4. **Post-mortem analysis**: this type of analysis is also conducted on the past as indicated by the name. It would be possible to study any part of a project retrospectively using this type of analysis. Thus, this type of analysis may be described as research-in-the-past-and-typical. It can hence be viewed as related to both the survey and the case study. The *post-mortem* may be conducted by

looking at project documentation (archival analysis) or by interviewing people, individually or as a group, who have participated in the object that is being analyzed in the *post-mortem* analysis.

The last two methods are both concerned with research-in-the-past, although they have different approaches to studying the past.

According to the taxonomy proposed by Travassos and Barros, empirical studies that can be run in software engineering can be classified as (Travassos & Barros, 2003):

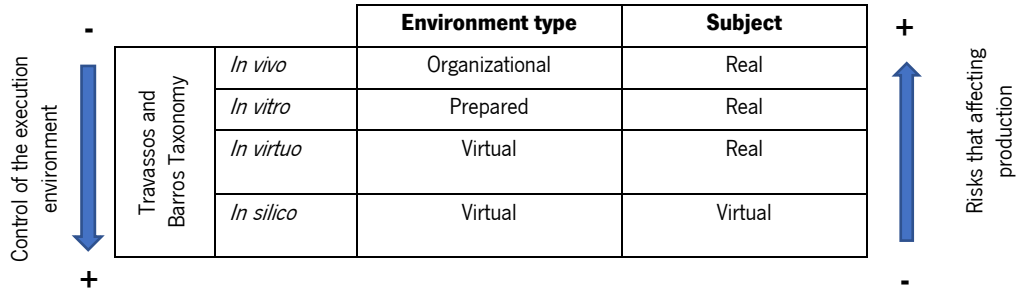
1. ***In vivo* experiments**: such experiments involve people in their own environments. In software engineering, experiments executed in software development organizations throughout the software development process and under real circumstances can be characterized as *in vivo*. They are studies with high influence in the supply chain of an organization and therefore they are considered of the greatest risk. A failure in the execution of the study may reflect in the intermediate product of a process and result in quality loss of the final product, generating rework, additional costs, and additional deadlines. Unlike an *in vitro* study in which the environment is prepared in these studies the environment is the process itself where tasks are performed.
2. ***In vitro* experiments**: such studies are executed in a controlled environment, such as a laboratory or a controlled community. In software engineering, most *in vitro* experiments are executed in universities or among selected groups of a software development organization. They are the most performed studies in software engineering area. In these studies, researchers assemble situations similar to real environments in prepared ("laboratories") environments, to observe how developers perform certain task. In this context a prepared environment is an environment in which some of its variables can be controlled or at least observed. Usually, *in vitro* studies involve manageable risks because they are not affecting any productive process within an organization. But they are also the most difficult studies, as it is difficult for organizations, regardless of their nature, academic or industrial, align the performance of these studies with their organizational objectives. For example, to perform a study in a course, the technology that we want study must be part of the syllabus of that course. In addition, they are also required to allocate physical re-sources (such as rooms, laboratories, tables) and the necessary hardware and software for environment composition where the study will be performed. About the subject selection, we can refer that is a difficult task, especially in industrial environments. It is not easy to take developers from their tasks and allocate

them to perform a study. There are problems ranging from schedule and activities allocation to unexpected project delays.

3. ***In virtuo* experiments**: such experiments involve the interaction among participants and a computerized model of reality. In these experiments, the behavior of the environment with which subjects interact is described as a model and represented by a computer program. In software engineering, these studies are usually executed in universities and research laboratories and are characterized by small groups of subjects manipulating simulators. The risks associated with this type of study are similar to those of an *in vitro* study. However, as the environment it is a model, their behavior can be predicted and customizable. In these studies, the developers perform their tasks, and they will only be influenced by the behavior described in the environmental model, this feature almost never achieved in *in vitro* studies. A risk associated with this type of study that can compromise the validity of these studies is related to the completeness and correctness of the model used to describe the environment.
4. ***In silico* experiments**: these studies are characterized for both the subjects and real world being described as computer models. In this case, the environment is fully composed by numeric models to which no human interaction is allowed. Due to the need of a large amount of knowledge, *in silico* studies are still very rare in software engineering, being limited to areas where subject participation is not an experimental study issue or intelligent agents can replace human subjects. For instance, we can find *in silico* studies applied to software usability experimentation, such as software performance characterization. The risks associated with this type of study are close to what we observe in *in virtuo* studies. In these studies, the risks associated with the selection and participation of subjects are eliminated but are introduced risks relating to the correctness and completeness of the models used for the description of human behavior performing certain tasks, most often much more complex than the models to describe the environments in which the studies are performed.

Table 2.3 shows the relationship between the increase control of the execution environment and increased risks affecting the production process. We can see that in *in vivo* studies have the highest relative risks to the production process with less control over the environment variables. While at the other side we have the *in silico* studies with greater environmental control and less impact on the production process. The *in vitro* and *in virtuo* studies are in an intermediate level, the main difference between them is in environmental control.

Table 2.3: Controls and risks associated with experimental studies according to the Travassos and Barros taxonomy (Travassos & Barros, 2003)



	Environment type	Subject
In vivo	Organizational	Real
In vitro	Prepared	Real
In virtuo	Virtual	Real
In silico	Virtual	Virtual

The taxonomies presented above are not the only existing forms of experimental studies classification. Indeed, in 1996 Basili (Basili, 1996) and Kitchenham (Kitchenham, 1996b) published different works about classification of experimental studies in software engineering area. Each taxonomy defines a perspective for classification and therefore an approach to the experimental studies implementation on Software Engineering area.

2.4. Experimentation Process in Software Engineering

A premise of Process Management is that "*the quality of a system or product is highly influenced by the quality of the process used to develop and maintain it*" (SEI, 2010). Based on this premise, the processes have been widely used and improved in many areas of engineering, such as Chemical Engineering, Industrial Engineering, and the Software Engineering is not an exception. From this reasoning, we can assume that the quality of the results of an experimental study is related to the quality of the process used to run it.

According to Shull *et al.*, "*a well-defined process can be observed and measured, and thus improved*" (Shull, Carver, & Travassos, 2001). Processes can be used to capture the best practices for dealing with a given problem. The adoption of processes also allows for dissemination of effective work practices to occur more quickly than the building up of personal experience. An emphasis on process helps software development become more like engineering, with predictable time and effort constraints, and less like art (Rombach, 2000).

In light of these considerations, there is a growing concern for the adoption of a process for performing experimental studies in Software Engineering (Amaral, 2006; Juristo & Moreno, 2001; Wohlin et al., 2012).

In this section, we briefly discuss the experimentation processes proposed by Wohlin *et al.*, Juristo and Moreno, and Amaral.

Wohlin *et al.* define a waterfall model to describe the experimentation process (Wohlin et al., 2012). Although, the authors argue that the process is not supposed to be a 'true' waterfall model because it is not assumed that an activity is necessarily finished prior to that the next activity is started. The order of activities in the process primarily indicates the starting order of the activities.

The experiment process proposed by Wohlin *et al.* consists of five activities or steps: (1) Experiment scoping; (2) Experiment planning; (3) Experiment operation; (4) Analysis & Interpretation, and (5) Presentation & Package (see Figure 2.2).

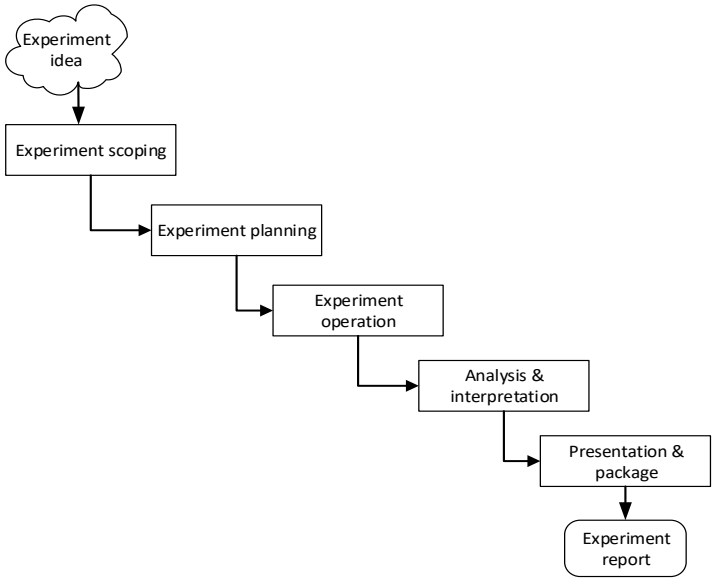


Figure 2.2: Overview of the experiment process proposed by Wohlin *et al.* (Wohlin *et al.*, 2012)

According Wohlin *et al.* the starting point for an experiment is insight, and the idea that an experiment would be a possible way of evaluating whatever we are interested in. In other words, we must realize that an experiment is appropriate for the question we are going to investigate.

As we can see in Figure 2.2 the experiment process can be divided into the following main activities. *Scoping* is the first step, where we scope the experiment in terms of problem, objective, and goals. *Planning* comes next, where the design of the experiment is determined, the instrumentation is considered and the threats to the experiment are evaluated. *Operation* of the experiment follows from the design. In the operational activity, measurements are collected which then are analyzed and evaluated in *analysis* and

interpretation. Finally, the results are presented and packaged in *presentation and package* (Wohlin et al., 2012).

Figure 2.3 presents an overview of the experiment process including the activities. The process is partly iterative, and it may be necessary to go back and refine a previous activity before continuing with the experiment. The main exception is when the operation of the experiment has started, then it is not possible to go back to the scoping and planning of the experiment. This is not possible since starting the operation means that the subjects are influenced by the experiment, and if we go back there is risk that it is impossible to use the same subjects when returning to the operation phase of the experiment process.

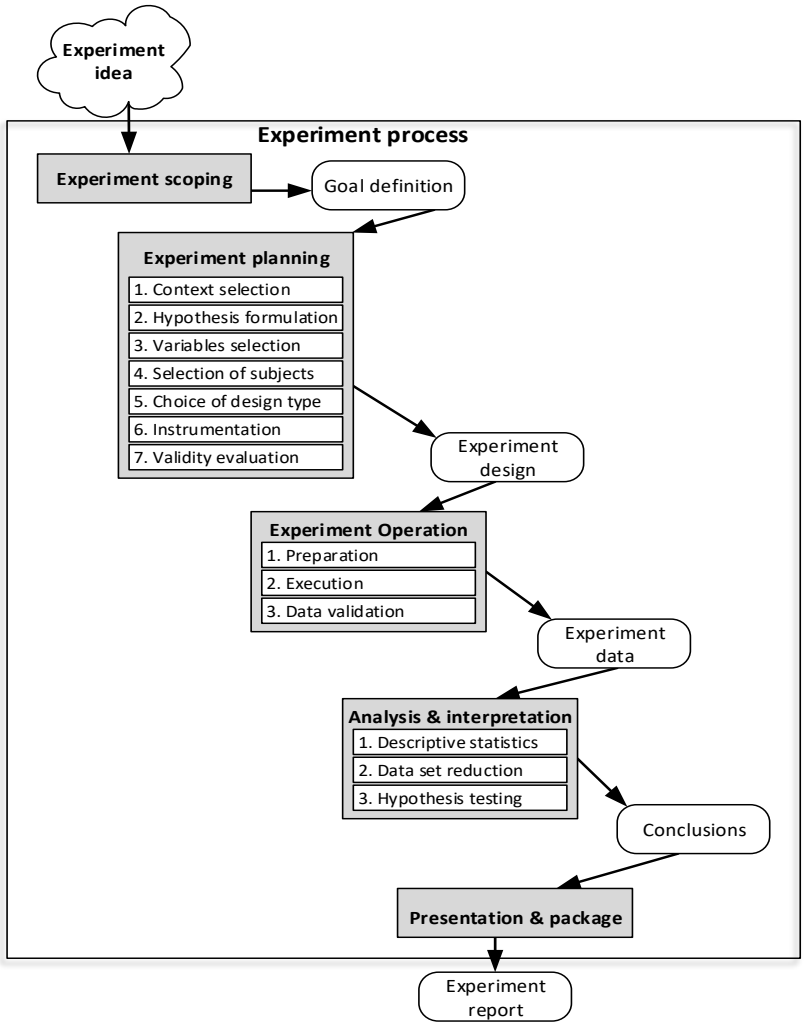


Figure 2.3: Overview of the experiment process and artefacts (Wohlin *et al.*, 2012)

In subsequent subsections will be discussed with some detail the activities or steps of the experiment process proposed by Wholin *et al.* (Wohlin et al., 2012).

2.4.1. Experiment Scoping

The first activity is scoping. The hypothesis has to be stated clearly. It does not have to be stated formally at this stage, but it has to be clear. Furthermore, the objective and goals of the experiment must be defined. The goal is formulated from the problem to be solved. If these objectives and goals are not properly set, it can cause a lot of rework or the experimental study cannot be used for what it was originally set. This step aims to clarify the initial information (reason, purpose, organization that is promoting research, among others) about the experimental study. It is important to keep the overall goal in mind. There are many possible goals of an experimental study, including validation, finding, among others.

In order to capture the scope, some researchers (Amaral, 2006; Juristo & Moreno, 2001; Kitchenham et al., 2002; Sjøberg et al., 2005; Wohlin et al., 2012) suggest the use of GQM (Goal/Question/Metric) approach defined in (Basili, Caldiera, & Rombach, 2002b; Basili, 1992) which defines a framework with following constituents:

- Object of study (what is studied?),
- Purpose (what is the intention?),
- Quality focus (which effect is studied?),
- Perspective (whose view?), and
- Context (where is the study conducted?).

In the context of the scope definition of an experimental study, the object of study is the entity studied. For instance, products, processes, resources, models, metrics, or theories. The purpose defines what is the intention of the experimental study. For instance, the purpose can be to evaluate the impact of two different techniques or to characterize the learning curve of an organization. The quality focus is the primary effect under study, such as cost, reliability, among others. The perspective presents the point of view under which the results of the experimental study are interpreted. For example, we can have perspective of the developer, project manager, client, or researcher, that is, the perspective taken for which you want to evaluate the experiment. The context is the environment in which the experimental study is performed. The context defines briefly what people are involved in experimental study (participants) and what software artifacts (objects) are being used in the experiment. Participants can be characterized by the experience, team size, among others. The objects may be characterized, for example, by size, complexity, priority, or application domain.

According Wohlin *et al.*, the context of an experimental study is composed of the conditions under which it is performed (Wohlin et al., 2012). The context can be characterized according to the following dimensions:

- *In vivo*, *in vitro*, *in virtuo* and *in silico* (Travassos & Barros, 2003);
- Students or professionals: characterizes the participants of the experimental study;
- Classroom problem or real problem: set the experimental study is being studied;
- Specific or general: characterizes if the results of the experimental study are valid for a specific context or for a general domain of the Software Engineering.

At the end of this stage (activity), the context, hypothesis, goals, and purposes of the experimental study should be very clear to anyone who is designing the experimental study.

2.4.2. Experiment Planning

The planning activity is where the foundation for the experiment is laid. This activity describes how an experimental study will be performed. As in other activities of software engineering, experimental studies should be designed, and plans must be followed to obtain the desired control during the execution of the experimental study. The result of the experimental study can be affected or even invalidated if not planned properly.

The experiment planning activity is responsible for describing the products, resources and processes involved in the study, including: the population being studied, the rationale and technique for sampling from that population, the process for allocating and administering the treatment and the methods used to reduce the bias, and determine the sample size (Kitchenham et al., 2002).

The first step of experiment planning activity is the *context selection*. In this step the context of the experiment is determined in detail. This includes personnel and the environment, for example, whether the experiment is run in a university environment with students or in an industrial setting.

The second step of experiment planning activity is the *hypothesis formulation*. In this step the hypothesis of the experiment is stated formally, including a null hypothesis and an alternative hypothesis. The researcher wishes to prove that the treatment planned in the experiment has a cause relation on the dependent variable(s). The conceptual hypothesis to validate is divided into two operative ones:

- Null Hypothesis, H_0: assumes that there is no difference between treatments for the dependent variable(s). The intention is to reject this hypothesis with the highest possible significance;

- Alternative Hypothesis, H_a H_1, H_2, etc: suggests that there is a significant difference between the two treatments. It specifies the assertion in favor of the rejected null hypothesis.

For example, if the purpose of the experimental study is to compare two techniques of inspection requirements A and B. Then we can have:

H_0: There is no difference between the techniques A and B.

H_a: There is difference between the techniques A and B.

H_1: The technique A is more efficient than the technique B.

H_2: The technique B is more efficient than the technique A.

In hypothesis definition, we must pay attention to the risks associated with method used in hypothesis testing. These risks try to translate the possible errors that can occur during the hypotheses verification. Among these errors are generally considered *Type I Error* and *Type II Error*.

The *Type I Error* occurs when the statistical test rejects the null hypothesis and this could not be rejected. This type of error indicates the presence of a relationship when this relationship does not exist. A means of evaluating this error is through their probability of occurrence, defined as (Wohlin et al., 2012):

$$P(\textit{Type I Error}) = P(H_0 \text{ is rejected} \mid H_0 \text{ is true})$$

The *Type II Error* occurs when the statistical test does not reject the null hypothesis and it should be rejected. This type of error indicates the absence of a relationship when this relationship could be effectively demonstrated. As in case of a *Type I Error*, a means of evaluating this error is through their probability of occurrence, defined as:

$$P(\textit{Type II Error}) = P(H_0 \text{ is not rejected} \mid H_0 \text{ is false})$$

A means of evaluating the error made during in the hypotheses verification is through the *power* of the statistical test to be applied. The *power* of the statistical test is defined as the probability that the test will correctly rejecting the null hypothesis. The *power* is defined as:

$$\text{Power} = P(H0 \text{ is rejected} \mid H0 \text{ is false}) = 1 - P(\text{Type II Error})$$

Thus, the statistical test with higher *power* must be chosen. The knowledge of the power of the statistical test can influence the planning, implementation, and results of the experimental study. In (Dybå, Kampenes, & Sjøberg, 2005) is discussed the relevance of the statistical *power* applied in empirical software engineering research and it is presented a systematic review (Biolchini, Mian, Natali, & Travassos, 2005;

Dyba, Kitchenham, & Jorgensen, 2005; Kitchenham, 2004; Kitchenham, Dyba, & Jorgensen, 2004) of the statistical power of the studies performed by the Software Engineering community in the decade 1993-2002.

The third step of experiment planning activity is the *variables selection*. In this step we define the independent (inputs) and dependent variables (outputs) to be used in experimental study and should be derived directly from the hypothesis formulated. The independent variables are those that can be controlled in an experimental study, which can be used to limit the scope of an experimental study and to differentiate experimental studies. The dependent variables are the factors that are expected to change or have a difference as a result of the application of changes in the independent variables. An important issue regarding the variables is to determine the values the variables actually can take. This also includes determining the measurement scale, which puts constraints on the method that we later can apply for statistical analysis.

The fourth step of planning experiment activity is the *selection of subjects*. In this step the subjects of the study are identified. It is in this step that we take the decision of the objects allocation and participant's selection. To generalize the results, the selection should be representative for that population. To obtain a sample, we must begin by defining a target population. The target population is the group or the individuals to whom the study applies. Ideally, a target population should be represented as a finite list of all its members. It is very important to define a representative sample, because, if we do not have a representative sample, we cannot claim that our results generalize to the target population.

Once we are confident that our target population is appropriate, we must use a rigorous sampling method. The methods used to define the sample of a population are defined as probabilistic and non-probabilistic methods. The probabilistic method is one in which every member of the target population has a known, non-zero probability of being included in the sample. The aim of this method is to eliminate subjectivity and obtain a sample that is both unbiased and representative of the target population. It is important to remember that we cannot make any statistical inferences from our data unless we have a probabilistic sample. The non-probabilistic method is used when the participants are chosen because they are easily accessible, or the researchers have some justification for believing that they are representative of the population. This type of method runs the risk of being biased (that is, not being representative of the target population), so it is dangerous to draw any strong inferences from them. Certainly, it is not possible to draw any statistical inferences from such samples. The detail of some techniques of the probabilistic and non-probabilistic methods is available in (Kitchenham & Pfleeger, 2002).

In (Kitchenham et al., 2002) are defined some guidelines for the activity of participants selection:

- Identify the population from which the subjects and objects are drawn;
- Define the process by which the subjects and objects were selected;
- Define the process by which subjects, and objects are assigned to treatments.

The fifth step of planning experiment activity is the *choice of design type*. In this step we must design a plan to systematically manipulate the independent variables and observe the dependent variable, and the operational context of the study, *i.e.*, physical, intellectual, and cultural description of the environment in which the study is will run.

Based on the statistical assumption, that is, the measurement scales and in which objects and participants are available for use, we can choose the experimental design. During the experimental design we must determine how many tests we must perform to make sure that the treatment effect is visible. A design of an experiment describes how the tests are organized and run. The design and the statistical analysis are closely related because the choice of design affects the analysis and vice versa.

In order to make the results more conclusive and meaningful, the "standard" approach to studies in empirical software engineering is the "comparative group experiment". However, this approach requiring a huge number of subjects results in issues involving the expense, timeliness, and applicability of results. Although uncommon, Harrison discusses the use of experimental designs using a single subject experiment, arguing about facilities and risks associated with this type of experimental design (Harrison, 2000). An experimental design well done forms the basis to allow the experimental study replication (Wohlin *et al.*, 2012).

The general design principles to be considered in an experiment design are randomization, blocking and balancing, and the most of experiment design use some combination of these (Wohlin et al., 2012). One of the most important design principles is randomization. All statistical methods used to analyze the data require that observations are from independent random variables. The randomization applies on the allocation of the objects, subjects and in which order the tests are performed. It is used to average out the effect of a factor that may otherwise be present and also used to select subjects that is representative of the population of interest.

Sometimes, we have a factor that has, probably, an effect on the response, but no one is interested in this effect. Blocking is used to systematically eliminate the undesired effect in the comparison among treatments. Within one block, the undesired effect is the same and we can study the effect of the treatments

on that block. This technique increases the precision of the experiment, and can be used when the effect of the factor is known and controllable (Wohlin et al., 2012).

If the treatments are assigned in such a way that each treatment has equal number of subjects, it will have a balanced design. This technique serves to simplify and strengthen the statistical analysis of data, but it is not entirely necessary (Wohlin *et al.*, 2012).

The sixth step of planning experiment activity is the *instrumentation*. The overall goal of the instrumentation is to provide means for performing the experiment and to monitor it, without affecting the control of the experiment. The instrumentation does not affect the outcome of the experiments, if it occurs then the results are invalid. We must identify and prepare suitable objects, develop guidelines, if necessary, and define measurement procedures (Wohlin *et al.*, 2012).

The instruments for an experiment are of three types, namely, objects, guidelines, and measurements instruments. The objects may be, for example, specification or code documents, and it is important to choose objects that are appropriate. The guidelines are needed to guide the participants in the experiment, this includes for example, process descriptions and checklists. The measurement instrument is used to collect data, this made, generally, by forms or interview. The measuring task to be used is to prepare forms and interview questions and validate these forms and questions with some people having similar skills to the subjects in the experimental study (Wohlin *et al.*, 2012). Amaral characterizes the instruments in artifacts of the experimental study and artifacts of the software (Amaral, 2006). Basically, experiment artifacts are guidelines and software artifacts, the objects.

The seventh and last step of planning experiment activity is the *validity evaluation*. As a part of the planning, it is important to consider the question of validity of the results we can expect. Before the results are presented it is important to assess how valid the results are. The concepts of validity refer to the best available approximation to the truth or falsity of the statements.

An appropriate validity refers to the results which should be valid for population of interest. First, the results should be valid for the population from which the sample was taken. Second, may be of interest to generalize the results to a larger population (Wohlin *et al.*, 2012). Validation consists of checking each form for correctness, consistency, and completeness. As part of the validation process, in cases where such checks reveal problems, the people who filled out the forms are interviewed (Basili & Weiss, 1984).

Threats to validity are influences that may limit our ability to interpret or draw conclusions from the study's data (Perry, Porter, & Votta, 2000). Researchers have a responsibility to discuss any limitations of

their study (Kitchenham et al., 2002). There are at least four major classes of validity that can be used to protect our studies from these threats: conclusion, internal, construct and external validity (Visaggio, 2008; Wohlin *et al.*, 2012).

The conclusion validity is concerned with the relationship between the treatment and the outcome of the experiment. We have to judge if there is a relationship between the treatment and the outcome. It must be ensured that there is a statistical relationship with a meaning. The risk is that the researchers do not draw the correct conclusion about relations between the treatment and the outcome of an experiment.

The internal validity is concerned with the validity within the given environment and the reliability of the results. This class of validity is concerned with factors that may affect the dependent variables without the researcher's knowledge (Wohlin et al., 2003). Internal validity means that changes in the dependent variables can be safely attributed to changes in the independent variables (Perry et al., 2000). Internal validity refers to how we infer that a relationship between variables is casual or that the lack of a relationship implies the lack of a cause. Internal validity is the basic minimum without which any experiment is uninterpretable. One way of assessing the existence of internal validity is answer the following question: *Did in fact the experimental treatments make a difference in this specific experimental instance?* (Campbell & Stanley, 1963).

The construct validity is a matter of judging if the treatment reflects the cause construct and the outcome provides a true picture of the effect construct (Wohlin et al., 2012). Construct validity means that the independent and dependent variables accurately model the abstract hypotheses (Perry et al., 2000). The construct validity is related to the relationship between the concepts and theories behind the experiment and what is measured and affected (Wohlin et al., 2003).

The external validity is a question of how general the findings are. Many times, we would like to state that the results from an experiment are valid outside the actual context in which the experiment was run (Wohlin et al., 2012). External validity asks the question of generalizability: *To what populations, settings, treatment variables, and measurement variables can this effect be generalized?* (Campbell & Stanley, 1963). The risks involved in this class of validity are: wrong experimental subject, wrong environment and performances, and wrong timing so that the results are affected from changed characteristics of the original experiment (Visaggio, 2008).

According to Carver *et al.*, in software engineering, the understanding of the threats of each type of validity is influenced by three factors: people, processes and products (Carver, VanVoorhis, & Basili, 2004). In this sense, when designing an experimental study, threats to validity must consider each of these factors.

A controversial issue on threats to validity is the use of students as subjects as equivalent to professional individuals, a threat to the external validity. It is convenient to use students as subjects in experimental studies because they are cheap, plentiful, and easily managed. However, there is a common skepticism about the ability to generalize results of these studies to industrial environments (Harrison, 2000).

According to Host *et al.*, software engineering students may be used instead of professional software developers under certain conditions (Höst, Regnell, & Wohlin, 2000). In (Berander, 2004) is showed that requirements prioritizations made by students and customers in projects seem to be more similar to industry. Moreover, Carver *et al.* point out that the use of students as subjects have helped in the development of new technologies and in the debugging of development technologies and experimental protocols for later use in industry. The authors suggest an approach based on observation and training to students gaining experience. With this, the main goal is decrease the differences between university students and industrial professionals when they run studies in a classroom environment (Carver, Shull, & Basili, 2003).

Höst *et al.* argue that it is essential to understand the factors that differentiate students from industrial professionals and thus obtain a better understanding of the validity of the results using students as subjects (Höst et al., 2000), since it is no clear how well the results of students generalize to software engineers professionals (Harrison, 2000). Empirical Software Engineering using students versus professionals will be better explained in chapter 4.

In conclusion, we can argue that the planning is a crucial step in an experiment to ensure that the results from the experiment become useful. Poor planning may ruin any well-intended study.

2.4.3. Experiment Operation

After design and plan the experimental study, the next task is to execute it in order to collect the data that will be analyzed. The operation consists in principle of three steps: preparation, execution and data validation (Wohlin et al., 2012).

In the preparation step, we are concerned with preparing the subjects as well as the material needed, for example, data collection forms. When running the experiment, it is vital to monitor the process carefully to ensure that everything is being done according to plan. Errors in experimental procedure at this stage will usually destroy experimental validity. Montgomery suggested that prior to conducting the experiment a few trial runs or pilot runs are often helpful. These runs provide information about consistency of experimental

material, a check on the measurement system, a rough idea of experimental error, and a chance to practice the overall experimental technique (Montgomery, 2012).

It is in this stage that the treatments are applied to all participants. This means that, this is the stage of the experimental study where the experimenter really meets with the participants. In most experimental studies in software engineering there are few other occasions where participants are really involved.

When dealing with people, researchers need to motivate them to participate in experimental studies, because no matter how good the experimental design and data analysis, the results will be invalid if the participants do not have actively participated in the experimental study. Some issues should be taken into account in order to convince them to participate in an experimental study as subjects: obtaining their consent and commitment, inform them about the intention of the experimental study, present results that sensitize them, preparing attractive elements to participate in the experimental study, and prevent that they be disappointed with the experimental study (Wohlin et al., 2012).

The actual execution is normally not a major problem. The main concern is to ensure that the experiment is conducted according to the plan and design of the experiment, which includes data collection.

Data collection from interviews (Hove & Anda, 2005) and questionnaires (Kitchenham et al., 2002; Punter, Ciolkowski, Freimut, & John, 2003) are the most common. The main advantage of using questionnaires is that does not require much effort of the experimenter, because he does not have to take an active part in the collection activity. The main disadvantage is that the experimenter does not have possibility to find directly inconsistencies, uncertainties, and faults in the questionnaires, among others.

In (Karahasanovic et al., 2005) are presented unobtrusive methods of collecting data. These methods are characterized by non-intervention of the participants and researchers in the process of collecting the data. The authors developed a feedback-collection tool to obtain additional during the experiments. According the authors, this additional data are useful to: validate the data obtained from other sources about solution times and quality of solutions; check process conformance; understand problem solving processes; identify problems with experiments; and understand subjects' perception of experiments (Karahasanovic et al., 2005).

When we create forms, it is important to decide if the participants must fill the forms anonymously or not. If no additional studies and therefore there is no real need for experimenter distinguish between different participants, then it may be appropriate use anonymous forms. But this means that there is no possibility to contact the participant if something is not filled in a proper manner. In many cases it is appropriate to prepare

a personal set of tools for each participant because many projects deal with randomness and repetition tests. In this sense, we can assign different treatments to them. However this can also be done when participants are anonymous (Wohlin et al., 2012).

In final step of the experiment operation, we must try to make sure that the actually collected data is correct and provide a valid picture of the experiment. In this sense, after data collecting, it is important to verify that the data was properly collected and if they are reasonable. This means, for example, if the participants understood the questionnaires and thus, they fill them correctly. Another type of error occurs if the participants have not participated in the experimental study seriously and therefore some data should be removed before analysis. One way to verify if the participants understand the experimenter's intentions is to provide, for example, a seminar to present the results (Wohlin et al., 2012).

2.4.4. Analysis and Interpretation

The data collected during operation provide the input to this activity. The data can now be analyzed and interpreted. After collecting the data in the operation activity, you should be ready to write conclusions based on these data collected. To write valid conclusions, these data should be interpreted. The first step in the analysis is to try to understand the data by using descriptive statistics. These provide a visualization of the data. The descriptive statistics help us to understand and interpret the data informally (Wohlin et al., 2012).

The interpretation of the data leads us back to our original research questions. At this time, we must understand and explain the limits of the study: what conclusions can we draw? Where are we limited in drawing conclusion? What might have influenced our results? It is also important try to explain what questions were answered and do not simply present the data. We should also discuss the practical significance of the results (Perry et al., 2000).

The next step is to consider whether the data set should be reduced, either by removing data points or by reducing the number of variables by studying if some of the variables provide the same information. Specific methods are available for data reduction (Wohlin et al., 2012).

Basically, there are two approaches for presenting and analyzing data: quantitative analysis and qualitative analysis. Quantitative analyses, as the name suggests, deal mainly with comparing numeric data. The comparisons are typically aimed at rejecting or not rejecting a null hypothesis. Two of the tools used in quantitative analysis are hypothesis testing and power analysis. Qualitative analysis tends to use data that is

less readily quantified: observations, interviews, among others. These techniques tend to be used when we want to understand people´s perspectives of a situation (Perry et al., 2000).

Thus, after having removed data points or reduced the data set, we are able to perform a hypothesis test, where the actual test is chosen based on measurement scales, values on the input data and the type of results we are looking for.

One important aspect of this activity is the interpretation. That is, we have to determine from the analysis whether the hypothesis was possible to reject. This forms the basis for decision-making and conclusions concerning how to use the results from the experiment, which includes motivation for further studies, for example, to conduct an enlarged experiment or a case study (Wohlin et al., 2012).

2.4.5. Presentation and Package

The last activity is concerned with presenting and packaging of the findings. In this activity the data, artifacts and conclusions about the experimental study are recorded and the policies how it would be available to the Software Engineering community are defined. This includes primarily documentation of the results, which can be made either through a research paper for publication, a lab package for replication purposes or as part of a company's experience base. Independently, we must take some time after the experiment to document and present it in a proper way.

In this last activity is important to make sure that the lessons learned are taken care of in an appropriate way. Moreover, an experiment will never provide the final answer to a question, and hence it is important to facilitate replication of the experiment. A comprehensive and thorough documentation is a prerequisite to achieve this objective. Having said that, the use of lab packages should done with care since using the same experimental design and documents may carry over some systemic problems and biases from the original experiment (Wohlin et al., 2012). Therefore, experiments packages containing all the artifacts and documents produced and used throughout the execution of experimental studies need to be built (Amaral, 2006).

The issue of packaging allows the repetition and confirmation of the results by other researchers. In the case of isolated studies is difficult to understand how to generalize the results and, therefore, how to evaluate their real contribution to Software Engineering community.

Unfortunately, in Software Engineering, too many studies tend to be isolated and are not replicated, either by the same researchers or by others (Basili et al., 1999). To have scientific validity, experimental studies should allow its replication and proof.

2.5. Approaches Related to Experimentation

One of the main expectations related to the experimentation in the Software Engineering is its role as a facilitator agent to software process improvement (Ott, Kinnula, Seaman, & Wohlin, 1999) and in the technology transfer from academia to industry (Shull et al., 2001; Sjøberg et al., 2002).

From this perspective, some approaches proposed in Software Engineering are related to the application of a methodology of knowledge acquisition through data collection and analysis. In general, these approaches seek to apply, in essence, a continuous cycle with the following steps:

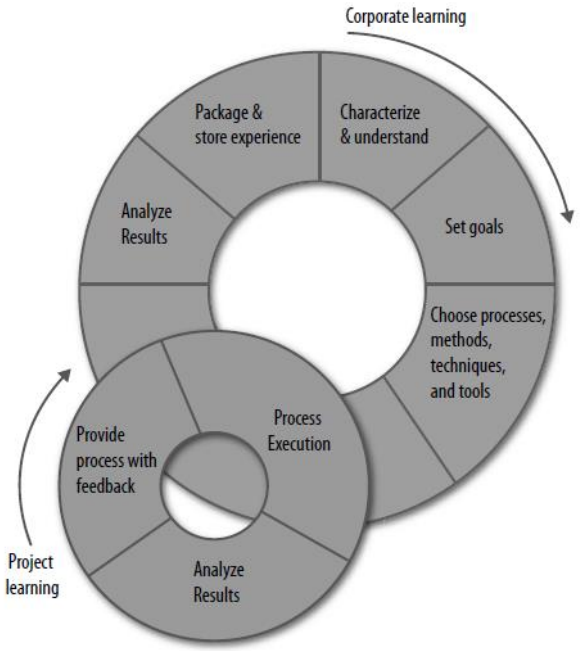
- Characterize the process, the environment and basic objectives of a technology, process or organization;
- Set from the goals, which will be measured on the product or process;
- Acquire the data collection or feedback from the developer, project or process;
- Compare the current practices with new ones or detect deficiencies;
- Analyse the data collection;
- Propose a forthcoming improvement to be incorporated through new practices or acquiring a new technology.

Related with the software process improvement, we must highlight the *Quality Improvement Paradigm* (QIP) (Basili, 1985; Basili & Green, 1994), the *Experience Factory* (Basili, 1989; Basili et al., 1992; Basili, Caldiera, & Rombach, 2002a; Basili & Seaman, 2002), the *Knowledge Dust and Pearls* (Basili et al., 2001) and the GQM (Basili, 1992; Basili et al., 2014) approaches.

The QIP was developed as a process for applying the scientific method to software engineering in an industrial environment. The QIP consists of six basic steps (Basili, 2011):

1. Characterize the current project and its environment with respect to the appropriate models and metrics. (*What does our world look like?*)
2. Set quantifiable goals for successful project performance and improvement. (*What do we want to know about our world and what do we want to accomplish?*)
3. Choose the process model and supporting methods and tools for this project. (*What processes might work for these goals in this environment?*)

4. Execute the processes, construct the products, and collect, validate, and analyze the data to provide real-time feedback for corrective action. (*What happens during the application of the selected processes?*)
5. Analyze the data to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements. (*How well did the proposed solutions work, what was missing, and how should we fix it?*)
6. Package the experience in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and save it in an experience base to be reused on future projects. (*How do we integrate what we learned into the organization?*)



7.

Figure 2.4: Quality Improvement Paradigm by Basili (Basili, 2011)

The QIP is a double-loop process, as shown by Figure 2.4. Research interacts with practice, represented by project learning and corporate learning based upon feedback from application of the ideas. Each one of the steps of this approach evolved over time based in formal experiments performed in Flight Dynamics Division of NASA/GSFC by Basili and his colleagues (Basili, 2011).

The QIP is related to the *Experience Factory* concept by defining a full cycle for process improvement. *Experience Factory* is an infrastructure aimed at capitalization and reuse of life cycle experience and products. It is a logical and physical organization, and its activities are independent from the ones of the development

organization. The *Experience Factory* provides an organizational schema for collecting experiences on reuse of empirical results, for analyzing them and generalizing the knowledge contained (Visaggio, 2008). In addition to the passive storage of experimental data, the *Experience Factory* may process requests of the current project providing relevant information of similar projects (Basili et al., 2002a).

The current schema of the *Experience Factory* organization is shown in Figure 2.5. The *Experience Factory* collects experiences and empirical validations on data related to development processes in various contexts: costs, benefits, risks, and improvement initiatives. For clearness, the *Experience Factory* shown in Figure 2.5 is briefly described. Further details can be found in references (Basili, 1989) and (Basili et al., 2002a) even though the schema is slightly different.

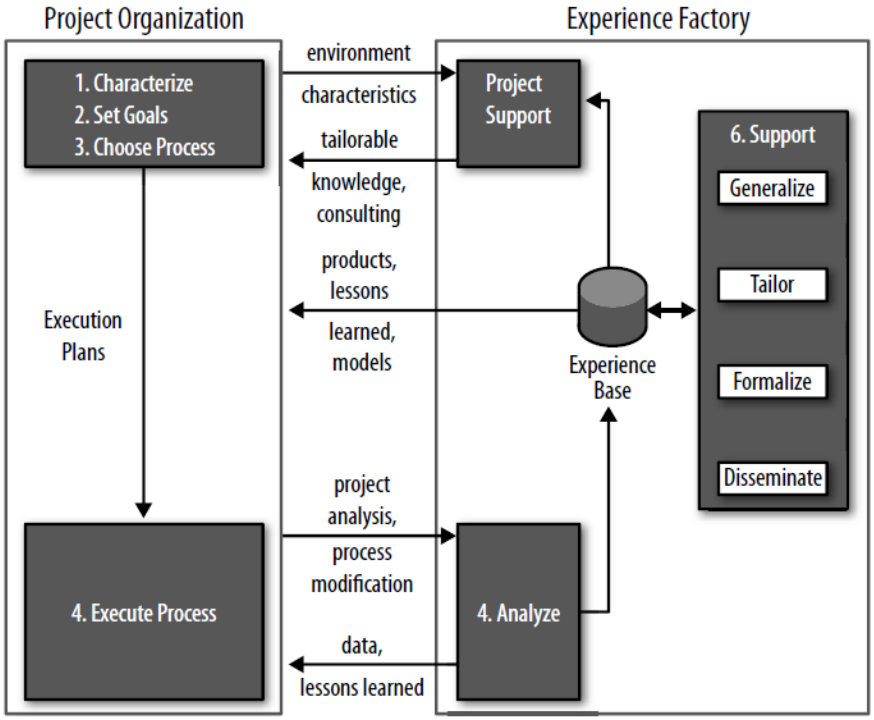


Figure 2.5: The Experience Factory by Basili (Basili, 2011)

Once a project begins, goals are defined; also, resources and the project context are characterized. This information allows us to extract existing experience packages from the *Experience Base*, which are explicated in products, lessons learned, and models. Either with or without support, the Project Manager defines the process he/she intends to use and derives the project execution plans. During project execution, data collected by the analyzer are registered in the project database. They are synthesized in order to search for other experience packages that can improve project execution with respect to the project goals. At the end

of the project, the data collected are compared to those of existing experience packages and formalized to provide further validation to the package or to extend their content. Following a project, the new empirical evidence can be integrated with the ones existing in the experience base and eventually lead to generalization of knowledge. In this way the experience base represents assets of knowledge to diffuse and socialize.

Knowledge Dust and Pearls is an approach to the knowledge management of software engineering research organizations. This new approach is influenced by the QIP and *Experience Factory* ideas. This approach is a kind of *Experience Factory* that addresses issues of short-term knowledge acquisition for organizational memory building. This approach captures the knowledge *dust* that employees use and exchange on a daily basis and immediately, with minimal modifications, makes it available throughout the organization. In parallel, the knowledge dust is analyzed and synthesized and transformed into knowledge *pearls*, which represent more sophisticated, refined, and valuable knowledge items that take longer time to produce. Further details about this approach can be found in reference (Basili et al., 2001).

The GQM approach comes as recognition that collecting data is not enough, you need to know what to do with them. Thus, the data collection needed to be goal-driven. This led to the development of the GQM approach to help us organize the data around a particular study (Basili et al., 2014; Basili & Weiss, 1984).

The GQM paradigm is a mechanism for defining and evaluating a set of operational goals, using measurement. It represents a systematic approach for tailoring and integrating goals with models of the software processes, products and quality perspectives of interest, based upon the specific needs of the project and the organization (Basili, 1992).

The goals are defined in an operational, tractable way by refining them into a set of quantifiable questions that are used to extract the appropriate information from the models. The questions and models, in turn, define a specific set of metrics and data for collection and provide a framework for interpretation (Basili, 1992).

The GQM paradigm was originally developed for evaluating defects for a set of projects in the NASA/GSFC environment. The application involved a set of case study experiments (Basili & Weiss, 1984), it was then expanded to include various types of experimental approaches, including controlled experiments (Basili & Selby, 1984; Basili et al., 2014).

Although the GQM was originally used to define and evaluate goals for a particular project in a particular environment, its use has been expanded to a larger context. It is used as the goal setting step in an evolutionary improvement paradigm tailored for a software development organization, the QIP, and an

organizational approach for building software competencies and supplying them to projects, the *Experience Factory* (Basili, 1992).

The GQM approach was continued to evolve, for example, by defining goal templates. In the TAME (*Tailoring A Measurement Environment*) project at the University of Maryland, Basili and Rombach developed an improvement oriented software engineering process model that uses the GQM paradigm to integrate the constructive and analytic aspects of software development. The model provides a mechanism for formalizing the characterization and planning tasks, controlling and improving projects based on quantitative analysis, learning in a deeper and more systematic way about the software process and product, and feeding the appropriate experience back into the current and future projects (Basili & Rombach, 1988).

Regarding to technology transfer from the research community to industry we have seen a great skepticism from the industry side. They have some difficult to accept the benefits of apply the experiment studies results performed in educational environments. In this sense, Software Engineering researchers have three main challenges to face for conduct more realistic experimental studies (Harrison, 2000; Sjøberg et al., 2002):

1. Treat tasks of size, complexity, and duration closer to reality found in the actual software processes in the organizations (Höst et al., 2000).
2. Select participants that represent the target population, considering that some studies show that under certain conditions students and professionals do not differ significantly (Carver, Jaccheri, Morasca, & Shull, 2003; Höst et al., 2000).
3. Conduct experimental studies in more realistic environments with more realistic tasks and participants representing the target population. Most studies cannot play an experimental environment with an industrial environment configuration with a supporting technology (processes, methods, techniques, tools, etc.). Traditional pen and paper based exercises used in a classroom setting are hardly realistic for dealing with relevant problems of the size and complexity of most contemporary software systems (Sjøberg et al., 2002).

Shull *et al.* propose a technology transfer methodology iterative based on execution of experimental studies (Shull et al., 2001). According to the authors, there are many factors that influence the technology transfer and does not make it a trivial task. No matter how good an idea is on its own merits, there are many other factors that influence its usefulness: budget and effort constraints, practical usefulness, etc. Many of these factors simply cannot be assessed and controlled in a laboratory environment. These factors can explain

why studies of new software development processes, inserted into industrial processes, are high risk activities (Shull et al., 2001).

Shull *et al.* based on the premise that the definition of technology transfer processes requires iteration to separate these factors and test them in smaller groups. Thus, we have a better chance that the results generate a real understanding of the influence of the organizational environment variables in the use of technology. A second reason for adopting an iterative approach is ensure that the fundamental issues are addressed in previous steps before we implement in a particular environment (Shull et al., 2001).

The studies of each stage of the proposed methodology are closely related to the risks that can be assumed and the desired control of the environment variables at a given time, and are (see Figure 2.6): (1) Feasibility Study; (2) Observational Study; (3) Case Study – Use in real lifecycle; (4) Case Study – Use in Industry.

The feasibility studies, sometimes referred to as "quasi-experimental designs" have a plan for execution and data collection, although at this point it is difficult to control all possible variables. At this point, we seek to explore the fundamental issues that led to the creation of the technology and not its details. The concern is with the generation of hypotheses that can be tested for evaluation of technology to be transferred. Although the control of the variables at this point can be difficult, the risks assumed in this type of studies are low compared to other types of studies because they can be running outside of production environments (laboratories). Classroom environments are well suited to feasibility studies. Although their results cannot be applied directly to industrial developers, running studies in the classroom allows new concepts to be tested before using them with expensive developers from industry. The concern at this stage is to evaluate if it is worthwhile to spend the resources required to develop the technology studied.

In observational study, the experimental subject performs some tasks while being observed by an experimenter. At this point, the collect data aims to provide us with information about how a particular task is performed with the technology being studied. Thus, the researchers can gain a better understanding of how a new technology is applied and what the possible difficulties in practice. Questions about training needs can be considered and evaluated from the results of the feasibility studies. Although these studies are also performed in laboratories, there is now the variables control in an attempt to isolate and test the factors that influence the use of technology in question. These studies have higher associated risks than the feasibility studies because it is difficulty to reproduce an environment with controlled variables.

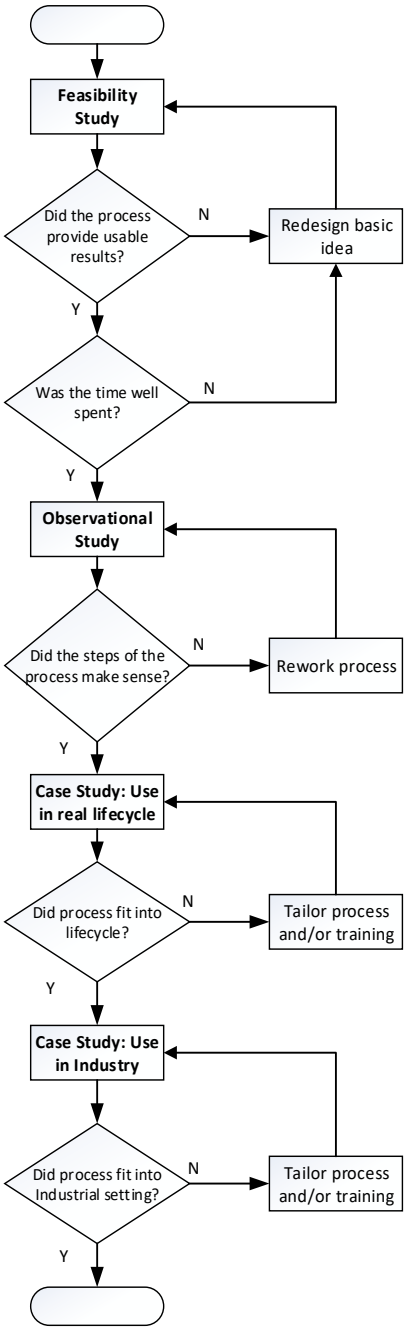


Figure 2.6: Technology transfer methodology (Shull *et al.*, 2001)

In "Case Study – Use in real lifecycle", we can pre-suppose that there are already sufficient indications that the investigated technology is effective. However, its use was only done in controlled environments, making it difficult to predict the human behavior in the use of technology in task execution in the productive process of an organization. Case studies are costly because individuals need to be trained. Thus, the risks associated with this type of study are mainly related to the cost, lack of environmental control and human

behavior. Although these studies are performed in the organizational environment, there is still the isolation of the organizational productive processes. Such studies serve as test cases of technology use within the organizational environment. From the interaction between technology and organizational environment, problems and deficiencies can be identified and addressed, considering all knowledge built in the analysis of data from previous steps. At this point, one can conclude that the adoption of a technology can be prohibitive for certain organizational environments.

In "Case Study – Use in industry", the study will be performed in the production process of an organization. The risks of this type of study are the highest since ranging from training cost, lack of variables control and human behavior to the negative impact on the performing tasks in the supply chain of an organization. All the above steps have sought to minimize those risks associated with the introduction of new technology. The latest assessments can be made in order to identify any points not covered yet and if the observable results are inside of expected results, so that action can be taken if necessary.

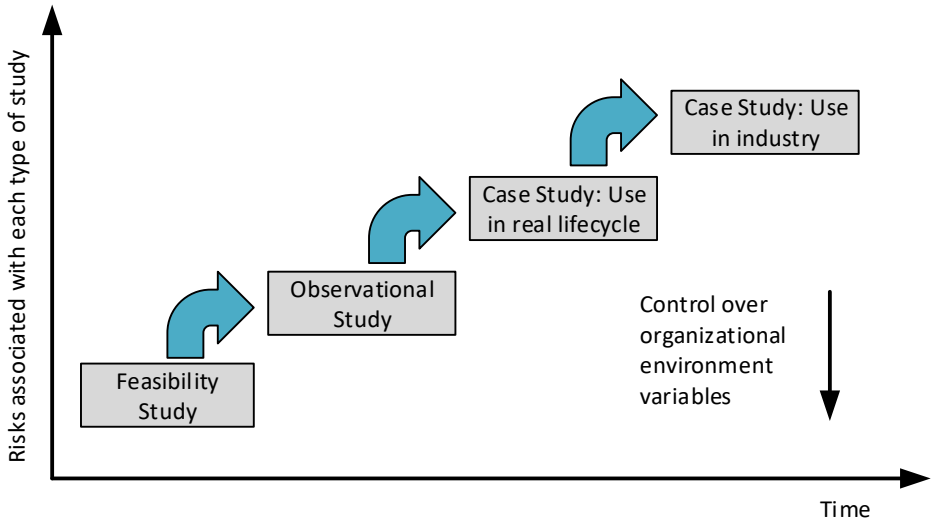


Figure 2.7: Risks associated with each type of study over the time

Figure 2.7 presents the increased risks associated with the implementation of the different types of studies over time. Also observed is the fact that, in general, as the risks increase, control over organizational environmental variables decreases.

However, the use of iterative approaches brings several other new challenges for researchers. Since the iterations imply several replications of the same experimental study. When the term replication is used, it is generally assumed that involve to replicate the study without any changes. In the scope of this thesis, we will consider a replication to be a study that is run, based on the design and results of a previous study,

whose goal is to either verify or broaden the applicability of the results of the initial study. If a researcher wished to explore the applicability of the results in a different context, then the design of the original study may be slightly modified but still considered a replication (Shull et al., 2001).

Based on the experience gained from *Readers' Project* (Shull et al., 2002; Shull et al., 2004), a bilateral project supported by the Brazilian (CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico) and American national science agencies (NSF - National Science Fundation) were investigated replications and transfer of experimental know-how issues in experimental studies performing.

According to Shull even when the experiment packages are effectively specified and exist, researchers find difficulties to understand and select an appropriate study to replicate it. The main difficulty is to understand the concepts underlying the techniques under study and to master the knowledge involved in running the experiment. This problem can be thought of as the difficulty in transferring experimental know-how between the original experimenters – the knowledge providers – and the replicators – the knowledge users (Shull et al., 2002).

Moreover, hardly in an experimental study replication will be possible reproduce all variables from the original study. The environment and the sample population will be different, and perhaps, it will be necessary adapt the instruments to extend the applicability of the original study.

Thus, Shull *et al.* draw attention to develop mechanisms to understanding and defining the experimental knowledge transfer process and evolution of the artifacts of an experiment package, through collaborative structures to facilitate interaction between researchers (Shull et al., 2002).

In the field of Empirical Software Engineering, a central challenge is to efficiently share experimental knowledge between replicators in order to facilitate replication comparability. Absorption tacit knowledge was a major source of difficulty during the replications done so far in the *Readers' Project* (Shull et al., 2004).

In Shull *et al.*, is realized an adaptation to the knowledge evolution spirally model of Nonaka-Takechi (Nonaka & Takeuchi, 1995) for Software Engineering experimentation context. The authors called *Experimentation Knowledge-Sharing Model* (EKSM) to this model adaption (Shull et al., 2004).

In summary, the EKSM acts iteratively on the knowledge sharing issues among researchers in each of its steps. In the first step, tacit knowledge is shared through socialization among researchers. In the second step this tacit knowledge, when possible, is explained and embodied on the experiment package. Finally, the explicit knowledge can be improved and internalized, and thus, it can be used in the experiment package evolution.

2.6. Conclusion

In this chapter we were presented topics related to experimentation in Software Engineering, including:

- The definition of key concepts and terms (hypothesis, factor, treatment, etc.) used in the software engineering experimental studies;
- An overview of qualitative and quantitative research methods used in Software Engineering experimental research, elucidating its main objectives and differences;
- An overview of experimentation process proposed by Wholin *et al.* (Wohlin et al., 2012), showing its main tasks and issues, such as, the threat to validity, hypotheses formulating, the realism in experimental studies and the participants selection;
- The discussion of some taxonomies to classify experimental studies in Software Engineering collected from the literature. We present also, some examples of this taxonomies;
- The discussion of some approaches related to the Software Engineering experimentation, used as facilitators of process improvement and technology transfer.

Finally, it is noteworthy that due to the specific characteristics of experimentation applied in Software Engineering area, such as, hardware/software resources and the availability of participants, the experiment operation step is the most critical activity of the experimentation process.

After experiment planning step, a failure in the operation experiment step commits all study results and, as a rule, invalidates them. The main risks are related to the lack of participants to conduct the experimental study. Thus, some lessons learned in the *Readers' Project* should be considered when we perform software engineering experimental studies, such as: (1) run pilot studies to anticipate deviations from the experimental plan and adapt it to these deviations; (2) train the researchers in the object of the study and in experimental knowledge necessary to replicate a particular study; (3) select external reviewers (individuals not involved in the planning and performing of the study) to revise the artefacts produced. This must done, before actually performing the study and thus anticipate deviations and adjust the experimental plan.

Chapter 3

Software Process and Project Management

3.1. Introduction

In the subsection 3.2 of this this chapter, an overview of the software development processes is presented. In a chronological way the different software development models will be presented and detailed. Thus, in this section we can see an overview about evolution of the most recognizes processes models used in software development. These software development models are classified into traditional, plan-driven models, and iterative and change-driven models. In addition, the background and fundamentals of agile software development are defined in the last part of this section.

In the subsection 3.3 of this chapter, an overview of the main project management approaches is also presented. The PMBOK (*Project Management Body of Knowledge*) (PMI, 2021) shall be presented with more detail. Throughout the research work of this thesis, there were several themes that focused on this area of knowledge, which is why it is addressed in some detail in this chapter.

In the subsection 3.4 of this chapter, an overview the software metrics is presented. This subject is discussed in some detail, given that in the chapters five and six we will present two demonstration cases based on two methods of software metrics, namely, the Use Case Points (UCP) and the Function Points (FPs). Thus, much of the research work of this thesis falls into this area.

Before starting to expose the concepts involved in all these research areas, we present a formal definition for literature review. A systematic literature review is defined by Kitchenham and Charters as, "a means of evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest. Systematic reviews aim to present a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology" (Kitchenham & Charters, 2007). The authors present this

formal definition in the report with comprehensive guidelines for systematic literature reviews appropriate for software engineering researchers, including PhD students.

The literature review includes in this research focused on three important subjects: software development models, project management approaches and software metrics. There are several links between these areas of knowledge.

Firstly, in order to get an overview of evolution in software development models a literature review was carried out in this issue. This study provides the concepts and the background of the most popular software standards and process models.

In order to get an overview in the most popular project management approaches, PMBOK, PRINCE2 and the main agile project management approaches will be presented. In this subsection, we present an ontology-based approach, namely PROMONT (Project Management Ontology) that intends to summarize all major project management standards and tools in one integrated reference model. This artifact can help us to understand the main concepts involved in the project management area.

The notion of software metrics and the main metrics used over time are presented based on the collection and analysis of the most relevant documents in this area. As we will see below, there are several metrics found in the scientific literature in this area.

3.2. Software Development Processes

It is crucial to distinguish the concept of a process model and a software method. According to Boehm, "a process model differs from a software method (often called a methodology) in that a method's primary focus is on how to navigate through each phase (determining data, control, or ''uses" hierarchies; partitioning functions; allocating requirements) and how to represent phase products (structure charts; stimulus-response threads; state transition diagrams)" (Boehm, 1988). Whereas, "a process model provide guidance on the order (phases, increments, prototypes, validation tasks, etc.) in which a project should carry out its major tasks" (Boehm, 1988).

The primary function of a software life cycle models is to "serve as a high-level definition of the phases that occur during development" (Abran, Bourque, Dupuis, Moore, & Tripp, 2004). Boehm refers that the main function of software development process models is to "determine the order of the stages involved in software development and evolution and to establish the transition criteria for progressing from one stage to

the next" (Boehm, 1988). In this sense, they are not aimed at providing detailed definitions, but at highlighting the key activities and their interdependencies.

During the history different models and approaches have been suggested for tackling the complexity and uncertainty of software development. Table 3.1 depicts the evolution of software process models in the past decades. As can be seen in the Table 3.1, it has also been suggested that the evolution of software development models originates from the problems of ad hoc programming that, at first, led towards traditional plan-driven models and towards iterative change-driven models of software development. The original meaning of the Latin term 'ad hoc' refers to a methodology that has been designed for a special purpose ('ad hoc - "for the special purpose or end presently under consideration"). In Latin, ad hoc literally means "for this". However, in this context, as often in software engineering literature, the term 'ad hoc' is used to refer to the low degree of methodological discipline (Basili & Reiter, 1981).

Table 3.1: The Evolution of Software Process Models

	Year	Model
Ad hoc	1950	Code-and-Fix Model
	1960	Stagewise Model
Plan-Driven	1970	Waterfall Model
	1975	Prototyping Model
	1981	Transform Model
	1992	V-Model
Iterative and Change-Driven	1975	Iterative and Incremental Model
	1981	Evolutionary Model
	1986	Spiral Model
	2001	Agile Software Development

Before the 1970's the development of software was based on ad hoc programming. In these early days of the software development, a very simple model was used, namely, code-and-fix model. This type of model consists of two steps: first, write some code and second, fix the problems in that code (Boehm, 1988). Basic problems with this model had led to the need to explicit sequencing of the phases of software development. In particular, the need to design prior to coding, to define requirements prior to design, and the

need for early preparation for testing and modification were identified (Boehm, 1988). One of the first models to rise to that challenge was the stagewise model as early as in the middle of the 1950s (Benington, 1983). This model stipulated that software be developed in successive stages and comes with the need to develop large software systems.

According to the literature we can classify the software processes models in: (1) plan-driven and (2) iterative and change-driven. The plan-driven models of processes have more emphasis on defining the scope, schedule and costs of the project upfront including an early fixing stage and extensive documentation of the end product requirements. Whereas iterative and change-driven models have more focus in the overall lifecycle model in which the software is built in several iterations in sequence.

Software development life cycles, traditional processes models and agile processes are another approach to classify the software processes models. There are many software development life cycles available, however, this literature review only covers the most used when designing and developing software systems. The most well-known are: (1) waterfall model, (2) V-model, (3) transform model, (4) evolutionary model, (5) prototyping model, (6) spiral model and (7) iterative and incremental model (see Table 3.1)

Waterfall, classic software development life cycle (phases and activities), prototyping, Rapid Application Development (RAD) and evolutionary models is another possible classification of the software processes models. In this classification, the evolutionary models include the incremental development, the spiral development, the Object-oriented models (e.g., RUP (Rational Unified Process)) and the Agile Software Development (ASD) models.

Early classification of the software process models is followed in this thesis. In the following subsections, the evolution of software development process models is discussed in more detail.

3.2.1. Plan-Driven Models for Software Development

The plan-driven approaches of software development have been defined as document-driven, code-driven, and traditional process models (Boehm, 1988). One common characteristic of these approaches could also be the recurrence of the software development phases only once during the development process, i.e., with only hints of interactivity (Larman & Basili, 2003). In the following sections of this thesis, the process models of this category will be referred to as traditional software development.

The two-step process model of code-and-fix, used in the early days of software development, resulted in difficulties that necessitated explicit sequencing of the phases of software development (Boehm, 1988). In

particular, the need to design prior to coding, to define requirements prior to design, and the need for early preparation for testing and modification were identified (Boehm, 1988). One of the first models to rise to that challenge was the stagewise model as early as in the middle of the 1950s (Benington, 1983). This model evolved from the problems caused by the increasing size of software programs, which could not be handled by a single programmer (Benington, 1983).

In 1968, the North Atlantic Treaty Organization (NATO) Science Committee held a software engineering conference in Garmisch, Germany, where the "software crisis", or "software gap" was discussed (NATO Science Committee, 1969). Perceived problems in software development, the key conference recommendations sought to standardize the software development process with emphasis on quality, costs, and development practices (Lycett, Macredie, Patel, & Paul, 2003). After this, the software development has since come to rely on a methodical approach. In 1970, as a refinement of the stepwise model, Royce introduces the waterfall model (Royce, 1970). In that year, Winston Royce, publishes an article where introduces his "personal views about managing large software developments" and it has since evolved into a concept consisting of the sequential phases of requirements analysis, design, and development (Larman & Basili, 2003). According to Boehm, the waterfall model provided two main advances over the stepwise model: "recognition of the feedback loops between stages, and a guideline to confine the feedback loops to successive stages to minimize the expensive rework involved in feedback across many stages" and "an initial incorporation of prototyping in the software life cycle" (Boehm, 1988). The waterfall model has become adopted for most software acquisition standards in government and industry. However, this model has solved various core problems in software development, it also includes features not appropriate for every software development context. Boehm argues that "a primary source of difficulty with the waterfall model has been its emphasis on fully elaborated documents as completion criteria for early requirements and design phases" (Boehm, 1988). These problems led to the formulation of alternative process models.

The V-Model can be considered a variation of the waterfall model which emphasizes traceability between the requirements, design, and implementation. The original V-Model includes similar phases to the waterfall model, but its phases are not defined as a linear activity but form a V-shape (Bruegge & Dutoit, 2010). In 1997, with the publication of the development standards for IT (Information Technology) systems of the Federal Republic of Germany, the V-Model entered into force as standard for all civil and military federal agencies (KBSt, 2004).

As we can see in Figure 3.1, the coding phase is situated in the intersection of the V, while the requirement gathering, system analysis, software design and module design form the system verification of the V-Model, and the acceptance testing, system testing, integration testing and unit testing form the system validation of the V-Model. The latter version of the V-Model, with public name V-Modell® XTV-Model is a Software development Life Cycle (SDLC) that emphasizes the concept of "Verification and Validation". In each step of development in V-Model, there will be a corresponding testing phase that will be validating such a process. Testing Phases will be planned in parallel with the development of the stage which they are supposed to be tested against and will be joined at the bottom by the actual coding process, hence the name V-Model. It is also considered to be an extended form of Waterfall Model since one step cannot be done without the completion of a previous process first. Each verification process in V-Model SDLC and their parallel testing counterpart.

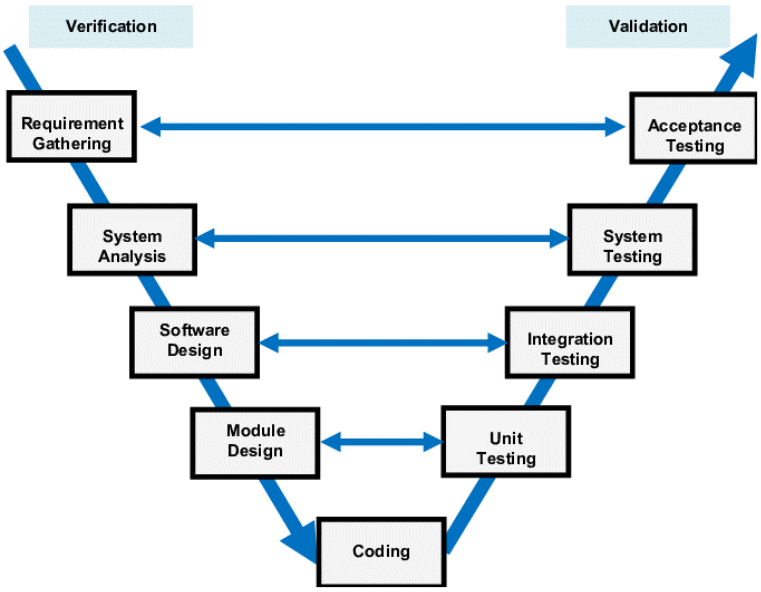


Figure 3.1: V-Model Software Development

The latter version of the V-Model has been extended to cover the entire system life-cycle and aims to be compatible with standards such as CMMI and to increase the scalability and adaptability of the model. In addition, the later version of the V-Model also perceives the possibility of conducting a series of subsequent V-cycles which increases the possibility of applying the model in a more iterative manner (KBSt, 2004).

More commonly, it has been argued, that "no life-cycle scheme, even with variations, can be applied to all system development" (McCracken & Jackson, 1982). On the other hand, according to the survey study of Fitzgerald (1998), despite numerous existing software development methodologies, as much as 60% of

software development organizations do not apply any development methodologies. An additional problem has been identified in using a disciplined approach to software development which is that, "rather than focusing on the *end* (the development of software), developers become pre-occupied with the *means* (the software development method)" (Fitzgerald, 1998). In practice, the result may be the disparity between the organizational software development process and its actual implementation in the software development teams. The findings of the Fitzgerald study suggest that practitioners will not adopt formalized methodologies in their prescribed form and, indeed, that they may be modifying and omitting aspects of methodologies in a very pragmatic and knowledgeable fashion (Fitzgerald, 1997).

Another dilemma identified among plan-driven approaches to software development, is the pursuit of certainty. The up-front requirements definition, and locking of the project scope, leads to contracts and decisions based on estimations of costs, time, and resources. However, such estimates have been found to be highly prone to uncertainty (Morien, 2005). Nonetheless, the success of software projects is often measured against these estimates as it may be appealing, from the viewpoint of both an acquirer and supplier, to agree fixed costs, scope, and schedule for the project up-front. However, it has been stated that "certainty is a myth and is the most uncertain part of any project" (Morien, 2005). In fact, it could be argued that the quest for certainty, in both time and money, may not only fail to pay off in these respects but may also seriously affect the quality of the end product. In his study, Morien concludes that the traditional approaches to system development have proven unsuccessful too often and the agile development approaches, that emphasize lean, change-oriented development, have been demonstrated to be effective (Morien, 2005).

It can be argued that the plan-driven models of software development can and should be applied in a dynamic way by repeating the phases or even the entire process, if necessary. However, the original purpose of these process models was not to welcome changes during the development, but rather to try to fix factors, such as scope, time, and money, up-front in order to eliminate change which was considered a risk factor.

3.2.2. Iterative Change-Driven Models for Software Development

Software development processes, developed after the waterfall model, seem to have the common aim of enabling, at least to some degree, the evolution of product requirements during the process of software development. This contributed one main modification to the earlier software development models: the adoption of the iterative and incremental approach. According to Larman, "iterative development is an

approach to building software (or anything) in which the overall lifecycle is composed of several iterations in sequence. Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test. The goal for the end of an iteration is an iteration release, a stable, integrated and tested partially complete system" (Larman, 2004). Incremental development involves adding functionality to a system over several releases, i.e., a repeated delivery of a system into the market or production. Thus, one incremental delivery may be composed of several iterations. A development approach where the system is developed in several iterations is called Iterative and Incremental Development (IID) (Larman, 2004). Interactive prototyping is an example of the iterative model. Prototyping involves the construction of a model of a system or object that contains only essential features (Pressman & Maxim, 2020). The prototype can serve as a first system created quickly and provides timely feedback on the feasibility of an application design' s and specifications.

Even though agile software development has recently brought the IID approach of developing software into the spotlight, the history of these approaches is, in fact, considerably longer (Larman & Basili, 2003). Many of the earlier change-driven approaches have adopted the ideologies of prototyping, for example, where the first early prototype gradually evolves into the final software product with no formal specifications or co-operation with the customer (McCracken & Jackson, 1982). Among the first models that focused on increasing the possibility of determining product improvements throughout the development process, was the evolutionary development model. This concept was first introduced in 1981 (Gilb, 1981) and has been expanded by Gilb (Gilb, 1988, 2005). This method suggested an iterative development approach in which the product increment was understood as a delivery to the real customer rather than a prototype (Gilb, 1981). While evolutionary delivery also lacks plans for future deliveries, it does attempt to capture feedback to guide future deliveries. This is in contrast to "pure" incremental delivery where the plan is drafted for several future deliveries and feedback is not the sole driving force (Larman, 2004).

The evolutionary model was followed by the transform model (Balzer, Cheatham, & Green, 1983), which is also based on the iterative development model and on adjusting the product during the development. The transform model, however, had a strong emphasis on product specifications due to its ideology of focusing on automatic transformation of specifications into code (Boehm, 1988). This approach had its origin in the problems of the earlier software development models producing "spaghetti code", which was difficult to modify and maintain (Boehm, 1988).

The spiral model, proposed by Barry Boehm in 1988, typically consists of four iteratively repeatable steps: 1) determining the objectives, alternatives, and constraints, 2) evaluating alternatives, and identifying and resolving risks, 3) development and verification, and 4) planning the next phase (Boehm, 1988). Boehm defined the spiral model as a risk-driven approach for software development. In the spiral model, the iteratively evaluated strategy for resolving the risks of the next spiral has an effect on the choice of the software development approaches to be adopted (Boehm, 1988). Depending on the risks, the spiral model then allows the adoption of any mixture of development approached, such as prototyping or elements from the specification-oriented waterfall approach modified to incremental development. According to Boehm, the risk-driven approach also means that the results of each risk analysis activity has an effect on the amount of time and effort allocated to the different development activities in the following spiral, while also influencing the required level of completeness, formality, or granularity of product specifications (Boehm, 1988).

Agile software development, which emerged in the middle 1990s, can also be classified as an iterative and change-driven software development approach. The common feature of agile methods is the recognition that software development cannot be considered to be a defined process, but rather an empirical (or nonlinear) one due to the constant changes that are welcomed during the development of the software product (Williams & Cockburn, 2003). It could be argued that at present there is no common agile process model with specified phases, but there is rather a set of fundamentals (Agile Alliance, 2001) common to the methods claiming to be agile. However, Extreme Programming (XP) (Beck & Andres, 2004), which is probably the best-known among the first agile methodologies, contains an underlying process model for agile software development that has been adopted and adapted by its successors. Figure 3.2 illustrates how Beck has compared the agile development model of XP with the waterfall model and with the iterative processes (Beck, 1999).

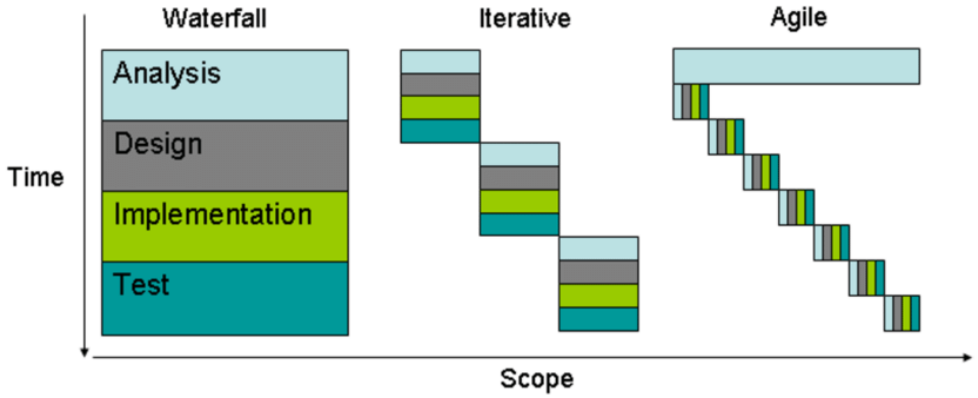


Figure 3.2: Process Models in Comparison (Beck, 1999)

The simplified illustration of the different software development models (Figure 3.2) provides an overview of the suggested differences between the models. According to Beck, XP aims at blending the activities of analysis, design, implementation, and testing, a little at a time, throughout the entire software development process (Beck, 1999). In Figure 3.2, we can see the evolution of the Waterfall Model and its long development cycles (analysis, design, implementation, test) to the shorter, iterative development cycles within, for example, the Spiral Model to Extreme Programming's blending of all these activities, a little at a time, throughout the entire software development process (Beck, 1999).

According to Larman, "in modern iterative methods, the recommended length of one iteration is between one and six weeks", (Larman, 2004) whereas the "incremental deliveries are often between three and twelve months" (Larman, 2004). Each iteration includes production-quality programming, not just requirements analysis, for example. And the software resulting from each iteration is not a prototype or proof of concept, but a subset of the final system. The principles of agile development suggest a short (i.e., from two weeks to two months) duration of the development iterations. Evolutionary development model also promotes relatively short delivery cycles of few weeks (Larman, 2004). Similarly as in the evolutionary model, agile methods also consider the term "iterative" as referring to evolutionary advancement of the product rather than just rework (Larman & Basili, 2003).

Agile software development is discussed in more detail in the next subsection.

3.2.3. Agile Software Development

The Agile Software Development (ASD) models arise to face the growing competition of companies to present solutions in a shorter period of time and at a lower cost. The software systems need to be developed continuously (evolutionary), with very fast deliveries (e. g. 2 to 4 weeks), in very uncertain environments regarding requirements and priorities. The application of these models intends to solve this problem. In business world, the time is the priority. Thus, with ASD models, it is intended to ensure a fast and continuous delivery, minimizing all the documental, analysis and design work.

Agile development methods apply "timeboxed iterative and evolutionary development, adaptive planning, promote evolutionary delivery, and include other values and practices that encourage agility-rapid and flexible response to change" (Larman, 2004). These methods appear in opposition to the so-called "heavyweight" or "monumental" methodologies, recognized for being bureaucratic. The "Light" methods try to reach a useful compromise between "no process" and "too much process". A lower document orientation,

less external work to role creation for each task and a greater code orientation (the main part of the documentation is the source code itself) are the main differentiating characteristics revealed by the "light" methods.

The light methods are adaptive rather than predictable, where requirements are constantly changing, and the iterative process is focused on the short term. These methods are "people-oriented" rather than "process-oriented". Thus, these methods put workers first, and the process is self-adaptive. Another feature of these methods is that they emphasize the production of high-value results, based on rapid adaptation to both external and internal events. These methods are rooted in a non-linear and non-deterministic perspective, on the edge of chaos, where planning is tenuous and control impossible – it is the world of adaptation, not optimization.

The idea of lightweight software development is that the method should only have a handful of practices and rules, simplifying the process. Alternatively, it could just have rules that are easy to follow. This puts it in direct contrast with 'heavyweight' software development which is built on complex methods with lots of rules. Thus, one question we can ask is: should we all use light methods? The answer is, it depends, if you have a team motivated with workers with great quality and with great trust between them, and if they are good with uncertain and volatile requirements, and if the teams are smaller, then it is a good idea to follow these methodologies.

The emergence of agile methodologies can be said to have begun in the mid-1990s, when software methodologies and techniques such as Extreme Programming (XP) (Beck, 1999), Scrum (Schwaber, 1995), eXtreme testing (Jeffries, 1999), Crystal Family of Methodologies (Cockburn, 1998), Dynamic Systems Development Method (DSDM) (Stapleton, 2003), Adaptive Software Development (ASD) (Highsmith, 2000), and Feature-Driven Development (FDD) (Coad, Luca, & Lefebvre, 1999) began to emerge. Many of these models are based on Object-Oriented technology. The emergence of agile methodologies is defined in more detail in, for example, (Abrahamsson, 2001, 2003).

The ideologies of agile software development can be traced back to lean manufacturing in the 1940s as well as agile manufacturing in the early 1990s. Lean manufacturing is based on the fundamentals of short-cycle time, reduced setup, multi-skilling and flow being in place while driving out waste in time, activity, inventory and space (Ross & Francis, 2003). The essence of the agile approach in manufacturing has been summarized as "the ability of an enterprise to thrive in an environment of rapid and unpredictable change" (Gould, 1997). Gould identified the elements of agility, and how they relate to the other buzzwords of our

time. The debate between the actual differences of lean and agile lasted sometime in the manufacturing sector (e.g., (James, 2005)), the central ideologies of both can be found in the fundamentals and methodologies of agile software development. In his paper, James claims that “there are those who believe that the relative success of lean will not be decided in academic journals but in the industrial marketplace for ideas because it has a solid content and track record and is delivering real results for people in a huge range of industries” (James, 2005). Thus, the author argues that it is the marketplace and not the academics who will make the choice between lean and agile. For example, in Lean Software Development (Poppendieck & Poppendieck, 2003) the lean principles are integrated with agile practices.

In software development, the agile movement was launched in 2001 when the various originators and practitioners of these methodologies met to identify the common aspects of these methods that both combined old and new ideas, and clearly shared some particular ideologies in common. As a result, the Manifesto for Agile Software Development was drafted and the term "agile" was chosen to combine the methods and techniques that would share the values and principles of agile software development (Agile Alliance, 2001). The values and principles of the Agile Manifesto (Agile Alliance, 2001) set out the central elements of agility that should be embedded in any method claiming to be agile. The agile manifesto emphasizes the agile values listed below on the left, while the items listed below on the right are still considered valuable too:

“Individuals and interactions	over	processes and tools
Working software	over	comprehensive documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan”

The twelve principles of agile software development (Agile Alliance, 2001) are: 1) the highest priority is to satisfy the customer through early and continuous delivery of valuable software, 2) the welcoming of changing requirements, even late in development, for the benefit of the customer’ s competitive advantage, 3) frequent delivery of working software, the release cycle ranging from a couple of weeks to a couple of months, with a preference for a shorter timescale, 4) daily collaboration of business people and developers throughout the project, 5) building of projects around motivated individuals by offering them an appropriate environment and the support they need, and trusting them to get the job done, 6) emphasis on face-to-face conversation for conveying information and within a development team, 7) working software is the primary

measure of progress, 8) agile processes promote a sustainable development pace for the sponsors, developers, and users, 9) continuous attention to technical excellence and good design enhances agility, 10) simplicity is essential for maximizing the amount of work not having to be done, 11) self-organizing teams give best results in terms of architectures, requirements, and designs, 12) regular reflection of teams on how to become more effective, and tuning and adjusting its behavior accordingly. The principles of agile software development can be considered as fundamental ideologies that should be embedded in the practices of any software development method claiming to be agile.

The core features of agility that should be embedded in any true agile method have been further specified as follows: iterative development of several cycles, incremental development, ability and permittance of teams to self-organize and determine the management of work, and emergence of processes, principles, and work structures during the project (Boehm & Turner, 2003a). In addition, the active involvement of users in requirements and planning, and the importance of tacit knowledge are identified as further important elements of agile software development (Boehm & Turner, 2003a).

Essentially, many of the ideologies behind the agile software development methods are not, nor have they been claimed to be new. Many of these ideologies and related agile software development methodologies have roots in, for example, the preceding iterative methodologies (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003) and agile and lean industrial product development (Poppendieck & Poppendieck, 2003). In addition, it has been widely acknowledged prior to the agile movement that the different methods of software development are far from being neutral and universally applicable (Malouin & Landry, 1983). Benington, among many others, has earlier considered top-down programming and specification as highly misleading and dangerous, as it assumes that enough detailed knowledge is available up-front to precisely know the objectives before producing a single line of code, and because it erroneously parallels the software development to the manufacturing industry (Benington, 1983). Furthermore, the positive effect of regular employee involvement in operating decisions and a high degree of responsibility for overall performance in high team spirit, loyalty, and motivation have also already been recognized among production workers (Deming, 1990). Neither has the iterative or incremental mode of software development been invented only by agile proponents, but it has a long history in software development (Larman & Basili, 2003). However, the agile software development approach has accomplished a novel mixture of old and new software development principles that have been gaining increasing interest among practitioners and researchers alike. Williams and

Cockburn suggest that the novelty of agile software development is, "if anything, the bundling of the techniques into a theoretical and practical framework" (Williams & Cockburn, 2003).

Abrahamsson *et. al* did a valuable comparative review of the agile software development methods. The authors conclude that "although agile software development methods have caught the attention of software engineers and researchers worldwide, scientific research still remains quite scarce" (Abrahamsson, Oza, & Siponen, 2010). They performed a comparative review from the standpoint of using the following features as the analytical perspectives: project management support, life-cycle coverage, type of practical guidance, adaptability in actual use, type of research objectives and existence of empirical evidence. The results that they found show that agile software development methods cover, without offering any rationale, different phases of the software development life-cycle and that most of these methods fail to provide adequate project management support. Moreover, quite a few methods continue to offer little concrete guidance on how to use their solutions or how to adapt them in different development situations (Abrahamsson et al., 2010).

In conclusion, the fundamentals of agile software development propose a very different view to the certainty aspect in the software development process, compared to the plan-driven approaches (see subsection 3.2.1.). In agile software development, the uncertainty of schedule, scope and budget of any software development project can be considered as a baseline assumption. Thus, agile software development methodologies can be regarded as a means of responding to the uncertainty of software development, rather than as a means of achieving certainty.

In the last years, there has been a considerable discussion in scientific forums both in favor and against agile methodologies. The early agile methodologies, especially, received criticism for the lack of scientific evidence (Abrahamsson, Salo, Ronkainen, & Warsta, 2002; Lindvall et al., 2002), and their suitability only for software development contexts where small and co-located teams were producing non-safety-critical products with volatile requirements (Williams & Cockburn, 2003).

Since the early days of agile software development, an increasing amount of interest has been paid to agile methods, by both practitioners and researchers, thus creating a growing body of empirical data on the different aspects of agile software development. Apart from the individual methods and practices of agile software development, problematic issues have arisen, such as the scalability of agile software development for large and multisite projects (Eckstein, 2004; Lindvall et al., 2004) and the compatibility of agile methods with existing standards (Lycett et al., 2003; Paulk, 2001; Reifer, 2003). Another issue that the international community has paid attention to is the organizational and business aspects of agility (Baskerville, Mathiassen,

& Pries-Heje, 2005; Coplien & Harrison, 2005; Oleson, 1998). Accordingly, the early agile methods and techniques have been evolving and are being updated, e. g., XP (Beck & Andres, 2004), Scrum (Schwaber, 2004; Schwaber & Beedle, 2002), Crystal (Cockburn, 2005), Test-Driven Development (TDD) (Beck, 2003), and DSDM (DSDMConsortium, 2003).

Currently, as more empirical evidence on the agile methodologies is available, it seems that the main arguments for and against their use is nowadays not so much about their benefits, but rather about the need to extend their scope and adapt them to organizations with established and mature plan-driven processes (Boehm & Turner, 2005). For instance, it has been suggested that one major problem in adopting agile methodologies can be found in balancing the currently dominating engineering ideologies and methodologies of manageable, predictable and repeatable processes with agile software development methods, which again embrace self-organization, process adaptation and constant changes (Lycett et al., 2003). Balancing the two approaches has been suggested in order to benefit from their strengths, and to compensate for their weaknesses (Boehm & Turner, 2003b). Bohem and Turner in this paper (Boehm & Turner, 2003b) present a risk-based approach for structuring projects to incorporate both agile and plan-driven approaches in proportion to a project's needs.

In the early days of its implementation, there was some confusion regarding the relationship between ad hoc coding and agile software development. It was proposed that one reason for this confusion is the piecemeal approach of agile software development (Highsmith & Cockburn, 2001). For instance, quality in design in agile software development is prioritized in ongoing design done in smaller chunks instead of massive up-front design of the system (Highsmith & Cockburn, 2001). In fact, the existing agile methodologies, such as Scrum for agile project management and XP for implementation of software, all seem to propose a rather disciplined approach to conducting the tasks of software development. In addition, studies (Kähkönen & Pekka, 2004; Nawrocki, Walter, & Wojciechowski, 2001; Paulk, 2001) indicate that by adopting different agile methods and practices, individual agile software development teams can accomplish a methodology that meets with the goals of CMMI level 2. However, there still seems to be a need to extend agile methodologies in order to meet, for example, CMMI requirements related to more organizational level practices.

In the last years, some early agile methodologies have been evolving and are being updated, for example, Extreme Programming (XP) (Beck & Andres, 2004) and Scrum (Schwaber, 2004). These changes involved organizational and business aspects of agility.

As it is the most agile and widely used, some more details about Extreme Programming (XP) will be presented next. Thus, XP is one of the numerous agile frameworks applied by IT companies. But its key feature is emphasis on technical aspects of software development, this distinguishes XP from the other approaches. XP was introduced by the Software Engineer Ken Beck in the 90s with the goal of finding ways to write high-qualitative software quickly and being able to adapt to customers' changing requirements. The XP framework normally involves five phases or stages of the development process that iterate continuously: planning, designing, coding, testing and listening (Beck & Andres, 2004).

XP involves seven interconnected processes, namely: (1) Identify (new) requirements with the customer; (2) Write (new) story lines and validate them with the customer; (3) Prioritize requirements. Select first from the list; (4) Write tests (5) Programming and testing; (6) Deliver to customer; (7) Review with the customer: skip to (1). The pair programming practice is widely used in the process of the step 5. Such a development process entails the cooperation between several participants, each having his or her own tasks and responsibilities. Extreme programming (XP) puts people in the center of the system, emphasizing the value and importance of such social skills as communication, cooperation, responsiveness, and feedback.

3.3. Project Management Approaches

This section begins with some essential definitions about project management collected from the literature. Thus, this section first provides a short overview of the theorical basis of project management, based on several bibliographic references. It then describes the importance of project management from an organization' s perspective, and the meaning of a project and project management based on project management literature. We then describe the Project Management Body of Knowledge (PMBOK) as one of project management process used extensively in industry.

Firstly, the main definition that we must know is the definition of project. According to the PMBOK, "a project is a temporary endeavor undertaken to create a unique product, service, or result" (PMI, 2021). The temporary nature of projects indicates a beginning and an end to the project work or a phase of the project work. Projects can stand alone or be part of a program or portfolio. Project and operation are two concepts which are sometimes confused. Thus, projects are temporary in nature, while operations are ongoing. Projects have definitive start dates and definitive end dates. The project is completed when the goals and objectives of the project are accomplished. Operations involve work that is continuous without an ending date

and often repeat the same process (PMI, 2021). Every project must work within the triple constraint combination of time, money, and quality.

According to Kerzner, "a project can be considered to be any series of activities and tasks that:

- Have a specific objective, with a focus on the creation of business value, to be completed within certain specifications;
- Have defined start and end dates;
- Have funding limits (if applicable);
- Consume human and nonhuman resources (i.e., money, people, equipment);
- Are multifunctional (i.e., cut across several functional lines)" (Kerzner, 2017).

Kerzner also argues that the result or outcome of the project can be unique or repetitive and must be achieved within a finite period of time. Because companies have very limited resources, care must be taken that the right mix of projects is approved (Kerzner, 2017).

Although there is no single definition of a project, there are four dimensions of projects that most project management writers have described, based on (Pinto & Kharbanda, 1995). These four dimensions are:

- Projects are constrained by a finite budget and time frame to completion; that is, they typically have a specific budget allocated to them as well as a defined start and completion time;
- Projects comprise a set of complex and interrelated activities that require effective coordination;
- Projects are directed toward the attainment of a clearly defined goal or set of goals;
- To some degree, each project is unique.

Project Management is the application of knowledge, skills, tools and techniques to project activities to meet specific project requirements (PMI, 2021). According, Pinto and Kharbanda, project management "is the dynamic process of leading, coordinating, planning and controlling a diverse and complex set of processes and people in the pursuit of achieving project objectives" (Pinto & Kharbanda, 1995).

Koskela and Howell argue that project management has generally been seen without an explicit theory (Koskela & Howell, 2002). They contend that it is actually possible to precisely point out the underlying theoretical foundation of project management as espoused in the PMBOK of Project Management Institute (PMI). This foundation can be divided into a theory of project and a theory of management. The product-oriented process of the PMBOK that specify and create the project product refer to the theory of project. The

process of initiating, planning, executing, controlling and closing the project of the PMBOK refer to the theory of management (Koskela & Howell, 2002).

In their description of the theory of project, Koskela and Howell refer to J. R. Turner' s theoretical view of project management (Turner, 2009). First, Turner claims that project management is about managing work; secondly, work can be managed by dividing the total work effort into smaller chunks of work – these are called tasks in the PMBOK; and finally, the divided tasks are related by sequential dependence (Turner, 2009).

In describing the other theoretical basis of project management as result of analyzing the PMBOK, Koskela and Howell suggest that project management is based on three narrow theories of management: management-as-planning, the dispatching model and thermostat model (model of management control). The first is evident from the structure and emphasis of the PMBOK. The second is apparent from the discussion of execution int PMBOK. The third is very clearly embodied in the closed loop of planning, execution and controlling. The planning process provide a plan that is realized by executing process, and variances from the baseline or requests for change lead to corrections in execution or changes in further plans (Koskela & Howell, 2002).

The software engineering can be viewed according two basic dimensions: process management and project management. Processes management are pertinent techniques and tools applied to a process to implement and improve process effectiveness, hold the gains, and ensure process integrity in fulfilling customer requirements. On the other hand, project management involves the planning, monitoring, and coordinating of people, processes, and events that occur as software evolves from a preliminary concept to full operational deployment (Pressman & Maxim, 2020).

After the definition of project and project management, we continue the description of the meaning and importance from organization' s perspective, based on project management literature. Projects can be viewed as critical stepping stones for organizational growth and productivity (Pinto & Kharbanda, 1995). Organizations worldwide are seeking ways to reduce bureaucracy and increase productivity and job satisfaction. Increasingly, project management processes are being used to create highly integrated organizations, controlled by project teams responsible for planning, controlling, coordinating and improving their own work (Kezsbom & Edward, 2001). Based on the increasing interest in project management, there has also been an explosion in the literature on project management.

Excellence in project management is defined as a continuous stream of successfully managed projects. Any project can be driven to success through formal authority and strong executive meddling. But in order for a continuous stream of successfully projects to occur, there must exist a strong corporate commitment to project management, and this must be visible (Kerzner, 2017).

The successful project consists of four factors, described in (Pinto & Kharbanda, 1995), projects being on time, on budget, performs as expected and is accepted by the customers. In order for the customer to accept the project, the project managers must devote additional time an attention to maintaining close ties with and satisfying the demands of the external clients. This requires the project managers to adopt an outward focus in their efforts. They are not just the managers of the project activities, but also the company' s sales representatives to the client base.

When we begin to view project management as a technique for implementing overall corporate strategy, it is clear that the importance of project management and, hence, project managers cannot be underestimated. Project management becomes a framework for monitoring corporate progress as it further provides a basis on which the skillful manager can control the implementation process. No wonder, then, that there is growing interest in the project manager' s role within the corporation (Pinto & Kharbanda, 1995).

Despite the emergence of new project management approaches, the failure rate still remains high, especially in the Information Technology (IT) industry. Research investigating project management in IT has not been reassuring. The Standish Group of Dennis, Massachusetts, conducted a lengthy and thorough study of IT projects and determined that (Pinto, 2019):

- 18% of IT application development projects are canceled before completion;
- 43% of the remaining projects face significant cost and/or schedule overruns or changes in scope;
- IT project failures cost U.S. companies and governmental agencies an estimated $1.2 trillion each year, with California the leading waster of IT projects funds at $164 billion annually.

Worldwide, the values are not either encouraging, estimates for canceled IT projects range as high as $3 trillion, or 4.7% of the total Gross domestic product (GDP) on the planet, more than the entire economic output of Germany (Krigsman, 2012).

Pinto mentions 10 signs of pending IT project failure (Pinto, 2019), namely:

1. Project managers do not understand users' needs.
2. Scope is ill defined.
3. Project changes are poorly managed.

4. Chosen technology changes.
5. Business needs change.
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost.
9. Project lacks people with appropriate skills.
10. Best practices and lessons learned are ignored.

To avoid project failure, it is critical to recognize the warning signs, including the inability to hit benchmark goals, the piling up of unresolved problems, communication breakdowns among the key project stakeholders, and escalating costs. Such red flags are sure signals that an IT project may be a candidate for termination (Pinto, 2019). Pressman and Maxim refers that "to avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a commonsense approach for planning, monitoring, and controlling the project" (Pressman & Maxim, 2020).

In a study of 250 large software projects between 1998 and 2004, Capers Jones found that "about 25 were deemed successful in that they achieved their schedule, cost, and quality objectives. About 50 had delays or overruns below 35 percent, while about 175 experienced major delays and overruns, or were terminated without completion" (Jones, 2004). Although the success rate for present-day software projects may have improved somewhat, our project failure rate remains much higher than it should be.

The number and complexity of projects undertaken by organizations is on the rise globally. In its 2017 Job Growth and Talent Gap report (PMI, 2017b), the Project Management Institute (PMI) predicts that employers will need to fill nearly 2.2 million new project-oriented roles each year through 2027. In these report, PMI refers that "the global economy has become more project-oriented, as the practice of project management expands within industries that were traditionally less project-oriented, such as health care, publishing and professional services" (PMI, 2017b). With more projects to manage and more intricacy within those projects, project managers are increasingly turning to tried-and-true methodologies to help them stay organized and maximize workflow efficiency. Each project management approach works best for certain kinds of projects. According to a Hubstaff survey from 2021, 39% of the companies surveyed in 2021 said that their organization implemented hybrid project management practices (Hubstaff, 2021). In today's project management world, forward-thinking managers and leaders do not adhere to a single methodology, they

become well-versed in many of them, and they learn how to mesh together various practices in order to accommodate whatever the project calls for. We will present below the most popular project management approaches.

Traditional project management approaches

The most popular traditional project management approaches are:

- **Waterfall** - Perhaps the most common way to plan out a project, the Waterfall method is a simple sequential approach. Each project phase must be completed before beginning the next one, leading to the end deliverable. These project plans can be easily replicated for future use. The Waterfall system is the most traditional method for managing a project, with team members working linearly towards a set end goal. Each participant has a clearly defined role, and none of the phases or goals are expected to change. Applying this approach, we should plan projects fully, then executing through phases. Waterfall project management works best for projects with long, detailed plans that require a single timeline. Changes are often discouraged (and costly). Requirements, system design, implementation, testing, deployment (service) or delivery (product) and maintenance are the typical stages of Waterfall project management (Royce, 1970).
- **Critical Path Method (CPM)** - Similar to the Waterfall methodology, the critical path method is a sequential approach that allows project managers to prioritize resources, putting more emphasis and investment into the most important work and rescheduling lower-priority tasks that may be slowing down the team. Critical path is a method for modeling projects where all necessary factors involved are input in the project and then the optimal timeline for completing it will output. Factors to input in the model include time estimates, task dependencies, milestones or deliverables, and any hard deadlines set by clients or stakeholders. The foundation of both critical path analysis and critical path method is that we cannot start a task until you finish the previous one. CPM was developed by the USA private sector in the 1950s.This methodology is related to the PERT (Program Evaluation and Review Technique) concept.
- **Critical Chain Project Management (CCPM)** - This methodology focuses on the resources needed for each task in the project. Using this approach, the project manager identifies and allocates resources for the most crucial, high-priority tasks, the “critical chain”, and builds in buffers of time around these tasks to ensure the project’s main deadlines are met.

Agile project management approaches

Agile project management is a collaborative methodology comprising short development cycles called "sprints" that incorporate feedback as the project progresses in an effort to embrace flexibility and continuous improvement. The methodology was developed by 17 people in 2001 as an optimized approach for software development (Agile Alliance, 2001). This methodology is characterized by building products using short cycles of work that allow for rapid production and constant revision. Agile project management focuses more on team collaboration and less on a hierarchical leadership structure. The most popular agile project management approaches are:

- **Scrum** - This practice disperses the traditional responsibilities of the project manager among the team members, with a Scrum master serving as a leader and facilitator. Scrum is an approach to managing complicated projects that may have to adapt to changes in scope or requirements. By emphasizing productivity, focus and collaboration, Scrum teams build high-quality deliverables quickly and can more easily adapt to change (Bonnie, 2021). The main focus of this methodology is enabling a small, cross-functional, self-managing to deliver fast.
- **Kanban** - Suited for projects with priorities that can frequently change, Kanban is similar to Scrum but progresses continuously, rather than in predefined sprint periods. Work is pulled in when needed and when capacity allows. Unlike Scrum, Kanban is less time-based and more focused on to-do lists. The Kanban methodology was designed to provide a workload-centric method of project management. In other words, its emphasis is on managing multiple deliverables across a team so that no one member is overworked or overwhelmed. The main focus of this methodology is improving speed and quality of delivery by increasing visibility work in progress and limiting multi-tasking.
- **Extreme Programming (XP)** - This approach was developed specifically for software engineering. It is ideal for clients who are not 100% sure what they need from the end product, and therefore need many opportunities for testing and feedback. It is a project management methodology designed to improve the quality of software and the development team's ability to adapt to customer needs. Work is done in short cycles (AKA sprints) with frequent iterations and constant collaboration with stakeholders. The main focus of this methodology is doing development robustly to ensure quality.
- **Adaptive Project Framework (APF)** - This approach is also suited to IT projects requiring a high level of flexibility and adaptability. It was developed by industry expert Robert Wysocki and is laid out step by

step in his book, *Adaptive Project Framework: Managing Complexity in the Face of Uncertainty* (Wysocki, 2010). It is a project management methodology that grew from the idea that most IT projects cannot be managed using traditional project management methods. Work is done in stages and evaluated after each one. This methodology focuses on the proposition that, since most current IT projects evolve as they go and begin with uncertain requirements, traditional project methods are not applicable (Freedman, 2010).

- **Lean** - This project management methodology focuses on streamlining and cutting waste. The goal is to do more with less and deliver value to the customer using less manpower, money, and time. This methodology is also known as "lean manufacturing" or "lean production". The main focus of this methodology is streamlining and eliminating waste to deliver more with less.

PRINCE2 (Projects IN a Controlled Environment) (AXELOS, 2017) and PMBOK (Project Management Body of Knowledge) (PMI, 2021) are two project management references of global use. Both resulted from the experiences gained in thousands of projects.

PRINCE2 is one of "the most widely used methods for managing projects in the world. It is a structured project management method based on experience drawn from thousands of projects and from the contributions of countless project sponsors, project managers, project teams, academics, trainers, and consultants" (AXELOS, 2017). The main premise of PRINCE2 is project management controlled that leaves nothing to chance. PRINCE2 is a project methodology for managing projects characterized by a product-based planning approach. Especially popular with the United Kingdom government, it gives teams greater control of resources and the ability to manage risk effectively.

The PRINCE2 method addresses project management with four integrated elements of principles, themes, processes, and the project environment (see Figure 3.3). The complete structure of the PRINCE2 is showed in Figure 3.3. The principles are the guiding obligations and good practices which determine whether the project is genuinely being managed using PRINCE2. There are seven principles and unless all of them are applied, it is not a PRINCE2 project. The themes describe aspects of project management that must be addressed continually and in parallel throughout the project. The seven themes explain the specific treatment required by PRINCE2 for various project management disciplines and why they are necessary. The processes describe a progression from the pre-project activity of getting started, through the stages of the project lifecycle, to the final act of project closure. Each process has checklists of recommended activities, products, and related responsibilities. The project environment is related with the fact that organizations often want a

consistent approach to managing projects and tailor PRINCE2 to create their own project management method. This method is then embedded into the organization's way of working (AXELOS, 2017). PRINCE2 is not a "one size fits all" solution. It is a flexible framework that can readily be tailored to any type or size of project.

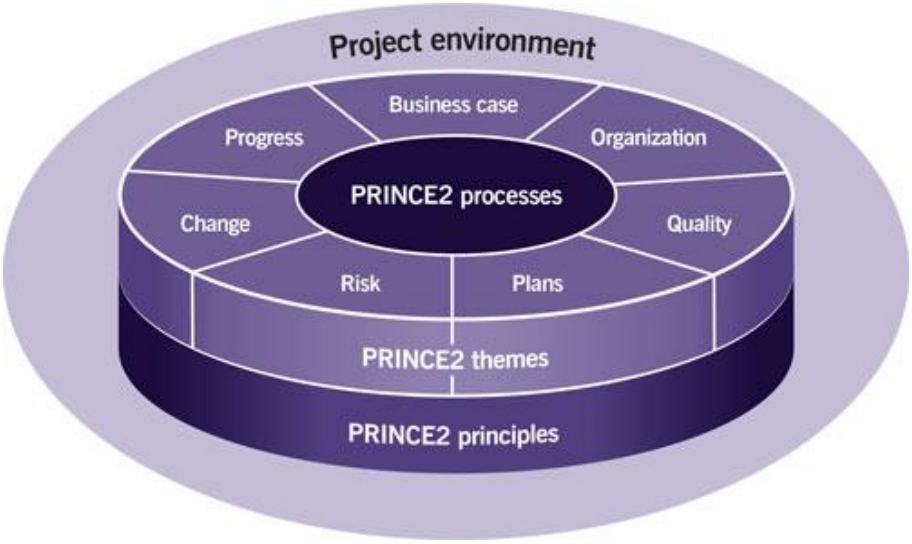


Figure 3.3: The structure of PRINCE2 (AXELOS, 2017)

PRINCE2 is designed so that it can be applied to any type of project, taking account of its scale, organization, geography, and culture. The full description of the principles, themes and processes of the PRINCE2 is beyond this thesis and full detail can be found here (AXELOS, 2017).

After a brief presentation of the PRINCE2 project management reference model, we will provide a more detailed description of the PMBOK. Thus, in 1983, the first version of the body of knowledge in project management was published by the Project Management Institute (PMI, 2022). PMI is a not-for-profit membership association for the project management profession, with thousands of members and credential holders in many countries which was founded in 1969. PMI is the "leading professional association for project management, and the authority for a growing global community of millions of project professionals and individuals who use project management skills" (PMI, 2022). PMI offerings include globally recognized standards, certifications, online courses, thought leadership, tools, digital publications, and communities.

PMBOK describes the sum of knowledge within the profession of project management. The full project management body of knowledge includes knowledge of proven traditional practices that are widely applied, as well as knowledge of innovative and advanced practices that have seen more limited use, and includes both published and unpublished material (PMI, 2021).

The purpose of the PMBOK is to identify and describe the knowledge and practices applicable to most projects most of the time with a widespread consensus about their value and usefulness. It also provides a common lexicon within the profession and practice for talking and writing about project management (PMI, 2021).

Based on the PMBOK, projects are often implemented as a means of achieving an organization' s strategic plan. Operations and projects differ primarily in that operations are ongoing and repetitive while projects are temporary and unique. In other words, a project is a temporary endeavor undertaken to create a unique product or service. Temporary means that every project has a definite beginning and a definite end. Unique means that the product or service is different in some distinguishable way from all the other products or services. For many organizations, projects are a means to respond to those requests that cannot be addressed within the organization' s normal operational limits.

In the PMBOK, project management is defined as the application of knowledge, skills, tools, and techniques to project activities to meet project requirements. Project management is an integrative endeavor, an action, or failure to take action, in one area will usually affect other areas. To help in understanding the integrative nature of project management, and to emphasize the importance of integration, the PMBOK describes project management in terms of its component processes and their interactions.

According to the PMBOK, projects are composed of processes. A process is "a set of interrelated actions and activities performed to achieve a pre-specified product, result or service. Each process is characterized by its inputs, the tools and techniques that can be applied, and the resulting outputs" (PMI, 2021). Project processes are performed by people and generally fall into one of two major categories: project management processes – describe, organize, and complete the work of the project; and product-oriented processes – specify and create the project' s product. Project management processes are applicable to most projects most of the time. Product-oriented processes are typically defined by the project life cycle and vary by application area. Project management processes and product-oriented processes overlap and interact throughout the project.

As we can see in Table 3.2, the PMBOK describes 49 project management processes that are grouped into ten knowledge areas, namely, project integration management, project scope management, project schedule management, project cost management, project quality management, project resource management, project communications management, project risk management, project procurement management and project stakeholder management (PMI, 2017a).

Knowledge Areas	Project Management Process Groups				
	Initiating Process Group	**Planning Process Group**	**Executing Process Group**	**Monitoring and Controlling Process Group**	**Closing Process Group**
4. Project Integration Management	4.1 Develop Project Charter	4.2 Develop Project Management Plan	4.3 Direct and Manage Project Work 4.4 Manage Project Knowledge	4.5 Monitor and Control Project Work 4.6 Perform Integrated Change Control	4.7 Close Project or Phase
5. Project Scope Management		5.1 Plan Scope Management 5.2 Collect Requirements 5.3 Define Scope 5.4 Create WBS		5.5 Validate Scope 5.6 Control Scope	
6. Project Schedule Management		6.1 Plan Schedule Management 6.2 Define Activities 6.3 Sequence Activities 6.4 Estimate Activity Durations 6.5 Develop Schedule		6.6 Control Schedule	
7. Project Cost Management		7.1 Plan Cost Management 7.2 Estimate Costs 7.3 Determine Budget		7.4 Control Costs	
8. Project Quality Management		8.1 Plan Quality Management	8.2 Manage Quality	8.3 Control Quality	
9. Project Resource Management		9.1 Plan Resource Management 9.2 Estimate Activity Resources	9.3 Acquire Resources 9.4 Develop Team 9.5 Manage Team	9.6 Control Resources	
10. Project Communications Management		10.1 Plan Communications Management	10.2 Manage Communications	10.3 Monitor Communications	
11. Project Risk Management		11.1 Plan Risk Management 11.2 Identify Risks 11.3 Perform Qualitative Risk Analysis 11.4 Perform Quantitative Risk Analysis 11.5 Plan Risk Responses	11.6 Implement Risk Responses	11.7 Monitor Risks	
12. Project Procurement Management		12.1 Plan Procurement Management	12.2 Conduct Procurements	12.3 Control Procurements	
13. Project Stakeholder Management	13.1 Identify Stakeholders	13.2 Plan Stakeholder Engagement	13.3 Manage Stakeholder Engagement	13.4 Monitor Stakeholder Engagement	

Table 3.2: Project Management Process Group and Knowledge Area Mapping (PMI, 2017a)

The needs of a specific project may require one or more additional Knowledge Areas, for example, construction may require financial management or safety and health management. Table 3.2 maps the Project Management Process Groups and Knowledge Areas.

The same processes can also be organized into the following five process groups - initiating process, planning process, executing process, monitoring and controlling process, and closing process. Initiating processes authorize the project or phase. Planning processes define and refine objectives and select the best of the alternative courses of action to attain the objectives that the project was undertaken to address. Executing processes coordinate people and other resources to carry out the plan. Controlling processes ensure that project objectives are met by monitoring and measuring progress regularly to identify variances from plan so that corrective action can be taken when necessary. Closing processes formalize acceptance of the project or phase and bring it to an orderly end.

The full description of the knowledge areas, the project management process groups including all processes of the PMBOK is beyond of this thesis and full detail can be found here (PMI, 2017a, 2021).

Today one can find several approaches that aim at collecting project management knowledge in some kind of standardized data model which can be used to implement project management software and to exchange project data. In order to perform project management activities, we can use different methodologies according to our needs and standards. Instead of creating a project plan manually, companies use project management software that supports most important tasks that appear in project management processes. Those software solutions emerged in the 1980's and have unfolded their full potential in the last decade (Abels, Ahlemann, Hahn, Hausmann, & Strickmann, 2006).

Projects have limited duration, are conducted by a specially assigned organization, and tackle a new task, thus dealing with a certain degree of uncertainty and risk. To meet these special requirements and risks, numerous methods for project management have been devised and successfully employed. Contemporary Project Management Software (PMS) systems implement these methods and facilitate project planning, execution, and control in daily business. Surprisingly, however, there is no common international data standard for the structure and exchange of project data in a broad use. (Abels et al., 2006).

Microsoft Office Project is one of the most often used solutions available today (Microsoft, 2022). Although it is not based on an official standard, it can surely be considered as a de-facto standard because of its market position. However, *Microsoft Office Project* does not have an open structure but uses a proprietary data model which is not defined by an independent body. Furthermore, it only focuses on a small

subset of what is typically understood as project management in general. So, for example, the *Microsoft Office Project*'s XML schema MSPDI (Microsoft Project Data Interchange) only addresses three out of the ten knowledge areas, namely scope management, schedule management and cost management. The remaining areas are not supported or only in small parts. It is therefore not sufficient to use it as the only reference when designing new project management software (Abels et al., 2006).

PROMONT is another project management approach provided by Abel *et. al* (Abels et al., 2006). PROMONT is an ontology-based approach with 32 classes that intends to summarize all major project management standards and tools in one integrated reference model. It offers extending definitions of project management issues aimed at supporting interoperability of project management systems, processes, and organizations. In particular, PROMONT offers a formal approach to define relationships and conditions between different terms that are used in project management. Thus, it is well suited for communication and integration management the most crucial issues in projects (Abels et al., 2006).

3.4. Software Metrics

Throughout history, the need to measure has always been a concern. In ancient Egypt and in ancient Greece, there are several testimonies of measuring instruments and measuring processes. Lord Kelvin has a well-known expression, he said “when you can measure what you are speaking about and express it in numbers, you know something about it”. Tom DeMarco also has a well-known expression, he said “you cannot control what you cannot measure”. Therefore, finding and applying metrics to assess size and software development effort is very important in today's highly competitive days.

Measuring software size is not the same as measuring temperature, humidity, pressure, time, volume, etc. Software is different, it is an intellectual product, and it is not material. Why measure software size? As in any other field of endeavor, size, and scale matter. The larger the requirements for a new software system, the more expensive, risky, and difficult it will be to deliver. Similarly, the more an organization depends for its success on delivering software, the greater the challenges of managing project teams to deliver software on time, efficiently, and to acceptable quality. Thus, the ability to measure and understand the influence of size can be critical for the performance of a software-producing organization. Specifically, software size is a key parameter to measure and control the following tasks (Symons, 2020).

Pressman and Maxim clarify the concepts of *measure*, *measurement*, and *metric*. Thus, “when a single data point has been collected (e.g., the number of errors uncovered within a single software component), a

measure has been established. In other words, a *measure* provides a quantitative indication of a product or process attribute. *Measurement* occurs as the result of the collection of one or more data points (e.g., a number of component reviews and unit tests are investigated to collect measures of the number of errors for each). A *software metric* relates the individual measures in some way (e.g., the average number of errors found per review or the average number of errors found per unit test)" (Pressman & Maxim, 2020). A Software Engineer collects measures and develops metrics so that indicators will be obtained. An indicator is a metric or combination of metrics that provides insight into the software process, a software project, or the product itself (Pressman & Maxim, 2020). In other words, a *metric* is a quantitative measure of the degree to which a product or process possesses a given attribute.

Based on Roche research, Pressman presents the activities of the measurement process (Pressman, 2010). Thus, Roche suggests a measurement process that can be characterized by five activities (Roche, 1994):

1. Formulation – the derivation of software measures and metrics appropriate for the representation of the software that is being considered;
2. Collection – the mechanism used to accumulate data required to derive the formulated metrics (i.e., survey, observation, experimental study, etc.);
3. Analysis – the computation of metrics and the application of mathematical tools;
4. Interpretation – the evaluation of metrics results in an effort to gain insight into the quality of the representation.
5. Feedback – recommendations derived from the interpretation of product metrics transmitted to the software team.

Fenton and Bieman define the scope of software metrics. According to the authors, software metrics is a term that embraces many activities, all which involve some degree of software measurement (Fenton & Bieman, 2015):

- Cost and effort estimation;
- Productivity measures and models;
- Data collection;
- Quality models and measures;
- Reliability models;
- Performance evaluation and models;

- Structural and complexity metrics;
- Capability-maturity assessment;
- Management by metrics;
- Evaluation of methods and tools.

There are a set of techniques currently in use for each facet of measurement. In their book "Software Metrics: A Rigorous and Practical Approach", Fenton and Bieman explore this set of techniques in detail (Fenton & Bieman, 2015).

Preliminary estimates of effort always include many elements of insecurity. Reliable early estimates are difficult to obtain because of the lack of detailed information about the future system at an early stage. However, early estimates are required when bidding for a contract or determining whether a project is feasible in the terms of a cost-benefit analysis. Since process prediction guides decision-making, a prediction is useful only if it is reasonably accurate (Fenton & Bieman, 2015).

Measurements are necessary to assess the status of the project, the product, the process, and resources. By using measurement, the project can be controlled. By determining appropriate productivity values for the local measurement environment, known as calibration, it is possible to make early effort predictions using methods or tools. Measurement can be used throughout a software project to assist in estimation, quality control, productivity assessment, and project control (Pressman & Maxim, 2020).

Estimation of resources, cost, and schedule for software development requires experience, access to good historical information (e.g., process and product metrics), and the courage to commit to quantitative predictions when qualitative information is all that exists. Estimation carries inherent risk, and this risk leads to uncertainty. Project complexity, project size, and the degree of structural uncertainty all affect the reliability of estimates (Pressman & Maxim, 2020).

Some cost estimation methods and tools are too difficult to use and interpret to be of much help in the estimation process. Numerous studies have attempted to evaluate cost models. Research has shown that estimation accuracy is improved if models are calibrated to a specific organization (Kitchenham, 1996a). Estimators often rely on their past experience when predicting effort for software projects. Cost estimation models can support expert estimation. It is therefore of crucial interest to the software industry to develop estimation methods that are easy to understand, calibrate, and use.

Over time, several metrics of size estimation have been developed with some different specificities. Based on these specificities the organization, more specifically the software managers should select the most

appropriate for the size of a software project they have to build. The main metrics were developed based on the software functions such as: Function Points (Albrecht & Gaffney, 1983), Feature Points (Jones, 1988), Boeing' s 3D Function Points (Whitmire, 1992), Mark II (Symons, 1991), Use Case Points (Karner, 1993), COSMIC Full Function Points (COSMIC, 2020; Symons, 2020). In this section of the literature review we will present in more detail Use Case Points (UCP) and Function Points (FPs) metrics because this metrics were applied in two demonstration cases (see chapters 5 and 6).

Traditional cost models take software size as an input parameter, and then apply a set of adjustment factors or 'cost drivers' to compute an estimate of total effort. In object-oriented software production, use cases describe functional requirements. The use case model may therefore be used to predict the size of the future software system at an early development stage. This thesis presents a demonstration case based on Use Case Points (UCP) method to estimate the software size of the projects performed in educational context. The method is not new but has not become popular although it is easy to learn. One of the reasons that the method has not caught on may be that there are no standards for use case writing. In order for the method to be used effectively, use cases must be written out in full. Many developers find it difficult to write use case descriptions at an appropriate level of detail. One way to solve this problem would be to provide guidelines for writing use cases for estimation purposes. The Use Case Points method was originally developed by karner in 1993 (Karner, 1993).

Rosa and Wallshein (Rosa & Wallshein, 2017) found that knowing initial software requirements at the start of a project provides an adequate but not always accurate estimate of project completion times. To get more accurate estimates, it is also important to know the type of project and the experience of the team (Pressman & Maxim, 2020). Function Points and Use Case Points are estimation techniques that gives us the project size and the effort to build a software project in the early stages of your development. These estimation metrics examine the requirements model with the intent of predicting the “size” of the resultant system. Size is sometimes (but not always) an indicator of design complexity and is almost always an indicator of increased coding, integration, and testing effort. By measuring characteristics of the requirements model, it is possible to gain quantitative insight into its specificity and completeness (Pressman & Maxim, 2020).

In 1993 the Use Case Points method for sizing and estimating projects developed with the object-oriented method was developed by Gustav Karner of Objectory Systems (later acquired by Rational Software Corporation). The method is an extension of Function Point Analysis and Mark II Function Point Analysis (an

adaption of FPA mainly used in the UK) and is based on the same philosophy as these methods. The philosophy is that the functionality seen by the user is the basis for estimating the size of the software.

Karner's work on UCP metric was written as a diploma thesis at the University of Linköping (Sweden). It was based on just a few small projects, so more research is needed to establish the general usefulness of the method. The work is now copyright of Rational Software Corporation and is hard to obtain. However, Schneider and Winters wrote a book that help software engineers and project managers how to implement use cases effectively and to apply the UCP method. The authors deliver a "clearly presented tour of the basics of designing effective use cases organized around a single large case study for an order-processing system" (Schneider & Winters, 2001).

In the years immediately following the creation of the metric by Karner, there is few research published using this metric. In 2002, Bente Anda reports the results from a study conducted to evaluate a method for estimating software development effort based on use cases, the UCP method, by comparing it with expert estimates. The UCP method gave an estimate that was closer to the actual effort spent on implementing the system than most estimates made by 37 experienced professional software developers divided into 11 groups. The author refers that "the results support existing claims that the use case points method may be used successfully in estimating software development effort. They also show that the combination of expert estimates and method based estimates may be particularly beneficial when the estimators lack specific experience with the application domain and the technology to be used" (Anda, 2002).

Damodaran and Washington present a paper that looks at the potential of successful application of the UCP method for estimating the size and effort of software development projects, including the major limitations and offers some possible remedies. The author concludes that UCP method of effort estimation is a very valuable addition to the tools available for the project manager. The method can be very reliable or just as reliable as other effort estimation tools such as COCOMO (COnstructive COst Model), Function Point and Lines of Code (Damodaran & Washington, 2002).

Cost models like COCOMO and sizing methods like Function Point Analysis (FPA) are well known and in widespread use in software engineering. But these approaches have some serious limitations. Counting function points requires experts. The COCOMO model uses lines of code as input, which is an ambiguous measure. None of these approaches are suited for sizing object-oriented or real-time software. Object-Oriented Analysis and Design (OOAD) apply the Unified Modeling Language (UML) to model the future system and use cases to describe the functional requirements. The use case model serves as the early requirements

specification, defining the size of the future product. The size may be translated into a number, which is used to compute the amount of effort needed to build the software.

In 2005, Anda *et al.* investigated in more detail the UCP method. In their research, the authors resorted to four software companies developed equivalent functionality, but their development processes varied, ranging from a light, code-and-fix process with limited emphasis on code quality, to a heavy process with considerable emphasis on analysis, design, and code quality. The authors calculate the effort estimate based on the UCP method for each of the four companies' software projects. The found that their result was close to the actual effort of the company with the lightest development process. Concretely the estimate was 413 hours while actual effort of the four companies ranged from 431 to 943 hours. They concluded that these results show, that the UCP method needs modification to better handle effort related to the development process and the quality of the code (Anda, Benestad, & Hove, 2005).

Ochodek *et al.* investigated the construction of UCP in order to find possible ways of simplifying it. The authors used a cross-validation procedure to compare the accuracy of the different variants of UCP (with and without the investigated simplifications). They performed an analysis based on data derived from a set of 14 projects for which effort ranged from 277 to 3593 man-hours. In their research, the factor analysis was performed to investigate the possibility of reducing the number of adjustment factors. The conclusion of their study was that the UCP method could be simplified by rejecting UAW (Unadjusted Actor Weight); calculating UCP based on steps instead of transactions; or just counting the total number of steps in use cases. Moreover, they proposed two use case based size metrics Transactions and TTPoints. They argue that could be used as an alternative to UCP to estimate effort at the early stages of software development (Ochodek, Nawrocki, & Kwarciak, 2011).

Yavari *et al.* did a study to assessment weak points of use case complexity metrics specially transaction metric in UCP method. In their study, the authors introduced other metrics for determining use case complexity that it can be led to simplify calculating UUCW (Unadjusted Use Case Weight) and it can be also tried to cover the most of aspects of use cases. According to the authors, these metrics can improve accuracy effort estimation and it can provide more information about system under discussed (Yavari, Afsharchi, & Karami, 2011). The limitation of this study is that the authors claim that they were still collecting data from the project to test the proposed method.

In another study, Popovic *et al.* created and cross-compared prediction models for estimating task type efforts by means of UCP size using an online analytical processing model and R packages on a set of 32 real-

world projects, with the goal of facilitating analysis of the correlation between project sizes and effort required to complete task types. In their research, the authors recognized the task types that are most correlated to the UCP size. According to authors, "task types that are most correlated to the UCP size can be estimated with acceptable accuracy for early estimates, while for the other task types one should use other estimation models (*e. g.* expert judgement)". They concluded that, requirements, scoping, functional specification, and functional testing task types have up to two times better estimation accuracies than project effort. The main conclusion of the authors was that using estimates of the most correlated task types and other techniques, such as expert judgment for others, they improved the overall project effort prediction accuracy and decreased the error from 26 to 16% (Popovic, Bojic, & Korolija, 2015).

Iskandar *et al.* applied the UCP method to measure the software size of one knowledge management portal. They called BINUS KMS to that portal. With the results of this metric, the authors intended to help managers to know how better the software size, complexity level and effort to development in numbering. The measurement of software size with UCP method showed that the project had medium software size with score of 108.56 UCPs of estimate effort and it will be developed in 2,064 hours (or in 258 days or 51.6 weeks or 12.9 months) and it had a development cost with a value in Indonesian currency. The authors concluded that, UCP, estimate effort and project value will powerful to help management in order to make decision regarding the implementation of IT software project development in term of time, money and people (Iskandar, Gaol, Soewito, Warnars, & Kosala, 2016).

Another example of UCP metric use was performed by Agtriadi *et al.* on employee software system. This software application realizes the latest personal data and provide accurate employee information for planning, development, welfare, and control of employees. The results of applying the metric showed that the project has small software size, they get a score of 70.34 UCPs. The authors conclude that, UCP method as part of measurement software size can be used to measure the software using internal product attribute which can show the effort, cost, and productivity (Agtriadi, Chandra, Warnars, & Gaol, 2017).

Azzeh and Nassif studied the relationship between project productivity and UCP environmental factors and they found that they have a significant impact on the amount of productivity needed for a software project. The authors designed four studies, using various classification and regression methods, to examine the usefulness of that relationship and its impact on UCP effort estimation. The results that they obtained are encouraging and show potential improvement in effort estimation. Based on the findings, the author refer that is better to exclude environmental factors from calculating UCP and make them available only for

computing productivity. The authors also refers that "the study also encourages project managers to understand how to better assess the environmental factors, as they do have a significant impact on productivity" (Azzeh & Nassif, 2017).

Kurniadi *et al.* applied the UCP method to measure the software size of Student Information Terminals (S-IT). This software application is an independent academic service information system for students, where this service makes it easy for students to obtain academic information in real time with information such as the transcript of academic achievement, finance, course, attendance, exam, lecturer, card examinees and announcements academic, and has the function to directly print the data independently on S-IT devices. The measurement of software size with UCP method showed that the project had a small size, the software had a score of 96.767 UCPs of estimate effort and it will be development in 1,452 hours or equivalent 9 months 1 week and it had a development cost with a value in Indonesian currency. The authors also refer that "the aims of measurement software size S-IT with the UCP is to help make decisions about the implementation of software development application project in terms of the estimated time, costs, and people" (Kurniadi, Hendric, Gaol, & Soewito, 2017).

Another example of UCP metric use was performed by Safrizal *et al.* on academic information system at Satya Negara Indonesia university. This software application deals with academic activities processes, which support the teaching learning process activities. This academic information system serving students carry out lectures and gain value in the form of cards in each semester of study results. In order to rebuild this system, the authors applied the UCP metric to estimate the software size of the actual system to help managers to take better decisions. The software size measurement had a score 86.864 UCPs and it was categorized as small software size project because this value is smaller than 99. In fact, the authors use a scale that categorize the UCP size of project software development in four categories, they consider a small project with a score less than 99 UCPs, a medium project with UCP between 100 and 299, a large project with UCP between 300 and 799 and an extreme project with more than 799 UCP (Safrizal, Warnars, Gaol, & Abdurachman, 2017).

Kurniadi *et al.* applied the UCP method to measure the software size of Online Admission System. This software system is part of the academic information system that exists in college. According to authors this system is useful in the management of new student enrolment. The results of applying the metric showed that the project has medium software size, they get a score of 103.86 UCPs, time work 1,454 hours and it had a development cost with a value in Indonesian currency (Kurniadi, Mulyani, Septiana, & Aulawi, 2018).

Concerned with adjusting the UCP method, Azzeh *et al.* propose a new ensemble construction mechanism applied for software project productivity prediction. According to authors ensemble is an effective technique when performance of base models is poor. They proposed a weighted mean method to aggregate predicted productivities based on average of errors produced by training model. The main goal of their study was to examine the efficiency of ensemble learning for predicting project productivity from environmental factors, and hence improve accuracy of effort estimation based on UCP. The obtained results by this study showed that "the using ensemble is a good alternative approach when accuracies of base models are not consistently accurate over different datasets, and when models behave diversely" (Azzeh, Nassif, Banitaan, & López-Martín, 2018).

Bagheri and Shameli-Sendi proposed another study to improve UCP method. Concretely, they proposed a new approach for cost estimation, based on UCP method, by considering all the existing risks related to software projects. According to authors the UCP method suffers some limitations such as less accuracy, failure to consider software risks, failure to consider software quality aspects, failure to consider different levels of software security, and so on. Thus, the authors proposed a new approach for cost estimation, based on UCP method, by considering all the existing risks related to software projects. The results obtained by this study indicated that the new estimation approach can produce relatively accurate estimates and also declare various aspects of project risks during project estimation. According to authors, the results also provide guidance for organizations that want to develop a software project (Bagheri & Shameli-Sendi, 2018).

Rak *et al.* proposed a new effort estimation model based on use case reuse, called the Use Case Reusability (UCR), intended for the projects that are reusing artifacts previously developed in past projects with similar scope. According to the authors "the UCR model introduces new classification of use cases based on their reusability, and it includes only those technical and environmental factors that according to the effort estimation experts have significant impact on effort for the target projects". The authors conducted a study within industry and academic environments using industry project teams and postgraduate students as subjects to validate the UCR model. According to the authors, the analysis of the results showed that UCR model can be applied in different project environments and that according to the observed mean magnitude relative error, it produced very promising effort estimates (Rak, Car, & Lovrek, 2018).

Silhavy et *al.* did a study to investigate the significance of using subset selection methods for the prediction accuracy of Multiple Linear Regression models, obtained by the stepwise approach. K-means,

Spectral Clustering, the Gaussian Mixture Model and Moving Window are evaluated as appropriate subset selection techniques. The authors concluded that this study proves subset selection techniques as a significant method for improving the prediction ability of linear regression models - which are used for software development effort prediction. They also concluded that the clustering method performs better than the moving window method (Silhavy, Silhavy, & Prokopova, 2018).

Mahmood *et al.* did a systematic review of studies associated with the best practices of UCP and expert judgment–based software development effort estimation techniques. The primary aim and contribution of this systematic review are to support the researchers through an extensive review to ease to other researcher's search for effort estimation studies. The authors have performed a systematic review of studies which are published in the period of 2000 to 2019. They have selected a total of 34 primary studies of UCP and expert judgment–based estimation techniques to report the research questions stated in their review (Mahmood, Kama, & Azmi, 2020).

Another systematic literature review to predicting software effort from UCP was performed by Azzeh *et al.* and published in Science of Computer Programming journal. With this systemic literature review, the authors intend to provide directions and supports for this research area of effort estimation. The objective of their study is twofold: 1) to classify UCP effort estimation papers based on four criteria: contribution type, research approach, dataset type and techniques used with UCP; and 2) to analyze these papers from different views: estimation accuracy, favorable estimation context and impact of combined techniques on the accuracy of UCP. The authors concluded that "there are multiple research directions for UCP method that have not been examined so far such as validating the algebraic construction of UCP based on industrial data. Also, there is a need for standard automated tools that govern the process of translating use case diagram into its corresponding UCP metrics". The authors also refers that, "although there is an increase interest among researchers to collect industrial data and build effort prediction models based on machine learning methods, the quality of data is still subject to debate" (Azzeh, Nassif, & Attili, 2021).

Azzeh *et al.* did a study to examines the impact of data locality approaches on productivity and effort prediction from multiple UCP variables. The results found by this research demonstrated that the prediction models that are created based on local data surpass models that use entire data. Also, the results showed that conforming to the hypothetical assumption between productivity and environmental factors is not necessarily a requirement for the success of locality (Azzeh, Nassif, & Martín, 2021).

In chapter 6 an overview of the Function Points (FPs) is presented from its origin. In this subsection is presented some research that has been published on UCP in conjunction with FPs. Additionally, some studies involving only FPs are also presented.

Some work has been published on UCP in conjunction with FPs. Certain attempts have been made to combine FPs and UCP. A modification of Karner's method to fit the needs of a specific company is discussed by Arnold and Pedross, where UCP are converted to FPs (Arnold & Pedross, 1998). An attempt to map the object-oriented approach into FPs has been described by Thomas Fetke *et al.* (Fetcke, Abran, & Nguyen, 1997), and converting UCP counts to lines of code by John Smith (Smith, 1999). But there does not seem to be much research done on these ideas.

FPs is one of the oldest metrics for software size measurement. It was proposed by Allan Albrecht in the 80s (Albrecht, 1979). This metric is based on functionality as perceived by the user and independent of the technology. In the literature review on this metric, some weaknesses in its application are found, namely:

- Counting FP itself takes a long time;
- Difficult to count consistently without an extensive training;
- Difficulty of using automated tools;
- Difficulty of counting embedded, highly algorithmic modules, and web systems.

After the publication of the metric by Albrecht, there have been several studies that intend to make some adjustments or repairs to the FPs metric. In this sense, Charles Symons was one of the first researchers to make some adjustments to this metric. He did a close examination of the method, and he found some weaknesses. Based on this research, the author proposed a partial alternative for the FPA method. He call "Mark II" to this new approach and it and can be consulted in detail here (Symons, 1988; Symons, 1991).

Whitmire was another researcher to propose some adjustments to FPA metric. He presented a 3D Function Points, a problem-based, technology-independent measure, so called because of its roots in FPs. He presented a discussion of the three-dimensional nature of an application problem. He identified measurable characteristics from each dimension that contribute to overall problem complexity. He provided rules for counting and assigning a level of complexity for the identified characteristics. Finally, Whitmire compared 3D Function Points to four existing problem-based size measures: Function Points, Feature Points, Mark II Function Points, and ASSET (Analytical Software Size Estimation Technique) Function Points (Whitmire, 1992).

Kitchenham and Känsäla did an empirical investigation of Albrecht FPs. Their study suggested that FPs are not well-formed metrics because there is a correlation between their constituent elements. It also suggested "that (for the dataset under investigation) two of the constituent elements were as good at predicting effort as the raw FP count and that the unweighted counts can be reasonable predictors of effort" (Kitchenham & Känsäla, 1993).

In order to evaluate some details of the FPA method, Alan and Robiliard did an empirical study of its measurement process. According to authors FPA was "initially designed on the basis of expert judgments, without explicit reference to any theoretical foundation" (Abran & Robillard, 1996). The results of this empirical study demonstrated that in a homogeneous environment not burdened with major differences in productivity factors there is a clear relationship between FPA's primary components and work-effort. Alan and Robiliard refer that this empirical study also indicates that there is such a relationship for each step of the FPA measurement process prior to the mixing of scales and the assignments of weights (Abran & Robillard, 1996).

In a "Why We Should Use Function Points" paper, Sean Furey refers that "FPs are technologically independent, consistent, repeatable, and help normalize data, enable comparisons, and set project scope and client expectations" (Furey, 1997). In his paper, Furey address these issues from the perspective of a practitioner who uses the International Function Point Users Group's *Counting Practices Manual, Release 4.0* rules for counting FPs.

Kitchenham found some problems with FPs metric. In her opinion, if FPs are used with caution within a specific organization, we will probably have few problems. But if you want to use them for cross-company benchmarking, as the basis for development or support contracts between different companies, or to develop generic estimation models, there is a non-negligible risk that you will encounter problems (Kitchenham, 1997).

Antoniol *et al.* presented and published a method for estimating the size, and consequently effort and duration, of object-oriented software development projects. They defined an adaptation of traditional function points, called "Object Oriented Function Points" (OOFP), to enable the measurement of object-oriented analysis and design specifications. They also constructed tools to automate the counting method. In summary, the authors showed that is possible to apply the concepts of FPs to object-oriented software and that results are accurate and useful in an industrial environment (Antoniol, Fiutem, & Lokan, 2003; Antoniol, Lokan, Caldiera, & Fiutem, 1999; Caldiera, Antoniol, Fiutem, & Lokan, 1998).

Ram and Raju suggest a counting procedure to measure the functionality of an OO system during the design phase from a designers' perspective. They adapted this procedure from Traditional Function Point Counting Procedure (TFPCP). The main aim of their research is to use all the available information during the OO design phase to estimate Object Oriented Design Function Points (OODFP). The novel feature of their approach is that it considers all the basic concepts of OO systems such as inheritance, aggregation, association and polymorphism (Ram & Raju, 2000).

Lokan reports an empirical investigation about the use and practical value of the fourteen *General System Characteristics* (GSCs) and the *Value Adjustment Factor* (VAF). The author concludes that "recording the GSCs may be useful for understanding project cost drivers and for comparing similar projects, but the VAF should not be used: doubts about its construction are not balanced by any practical benefit". Lokan argues that a new formulation is needed for using the GSCs to explain effort (Lokan, 2000).

In order to complement previous research, Antoniol *et al.* performed an empirical validation of OO size estimation models. It should be noted that, in the previous work the authors proposed OOFP, an adaptation of the FPs approach to OO systems. In this research, the authors extended the empirical validation of OOFP substantially, using a larger data set and comparing OOFP with alternative predictors of LOC (*Lines of Code*). The main aim of their research was to gain an understanding of which factors contribute to accurate size prediction for OO software, and to position OOFP within that knowledge. They identified several factors that influence size estimation (Antoniol et al., 2003).

Ceddia and Dick performed an experiment using FPs in educational context. The aim of their work is to perform a FP count of the projects of their students. Due to the large number of projects and the changing scope of projects a method of automatically counting FPs has been devised that uses the output from design tools that students have used. Principally the method counts use cases and database tables. These projects are performed by students of the final year of the Bachelor of Computing course. The students should complete an industry project where they work in teams to build an IT system for an external client. In their activities, the students performed a FP count in design phase and another FP count in the testing stage. The results of these calculations are then compared. In the Ceddia and Dick paper, the authors discusses two issues (a) proposing a metric for project size and (b) automating the production of that metric (Ceddia & Dick, 2004).

In their research, Harput *et al.* present an extension of the Function Point Analysis (FPA) method to object-oriented requirements specifications. Their concerns focus how function points can be reasonably

counted for object-oriented requirements specifications. They found that this cannot be done fully automatically, since several constructs of such a representation can be interpreted in various ways in the spirit of FPA, depending on the context. For applying FPA to object-oriented requirements specifications, the authors defined rules that specify a semi-automatic transformation from an object-oriented requirements model to an FPA model (Harput, Kaindl, & Kramer, 2005).

Zivkovic *et al.* proposed the unified mapping of UML models into FPs. This mapping is formally described to enable the automation of the counting procedure. The authors defined three estimation levels that correspond to the different abstraction levels of the software system. They conclude that the level of abstraction influences an estimate's accuracy. In their research, based on a small data set, they proved that accuracy increases with each subsequent abstraction level. Finally, the authors proposed changes to the FPA complexity tables for transactional functions in order to better quantify the characteristics of object-oriented software (Živkovič, Rozman, & Heričko, 2005).

In another research, Abrahão and Poels present an empirical study that evaluates OO-Method Function Points (OOmFP), a functional size measurement procedure for object-oriented systems that are specified using the OO-Method approach. They conducted a laboratory experiment with students to compare OOmFP with the IFPUG – Function Point Analysis (FPA) procedure on a range of variables, including efficiency, reproducibility, accuracy, perceived ease of use, perceived usefulness, and intention to use. Their results showed that OOmFP is more time-consuming than FPA but the measurement results are more reproducible and accurate. Their results also indicated that OOmFP is perceived to be more useful and more likely to be adopted in practice than FPA in the context of OO-Method systems development (Abrahão & Poels, 2007).

Chamundeswari and Babu proposed an extended FP approach for size estimation of object-oriented software. The authors proposed a new and enhanced approach for OO software size estimation by providing rules that better guide the practitioners. Their paper presents a sample case study describing the applicability of the proposed approach. They found that the developmental size predicted by applying the proposed approach for a set of sample projects correlates well with the size prediction obtained through the existing approaches. They conclude that "the proposed approach provides simple and unambiguous guidelines for the identification of FPA components as well as for the calculation of complexity due to each one of those components, without adversely affecting the accuracy of software size estimation" (Chamundeswari & Babu, 2010).

Irawati and Mustofa proposed a system to measure software functionality using FP method based on design documentation. Their research aims at designing and implementing a system that makes users convenient in analyzing software functionality size based on FP method referring to IFPUG CPM 4.3.1 standards. According to the authors, the system helps users to perform FP analysis in a faster and easier way without sacrificing accuracy. The input for the system is XMI (XML Metadata Interchange) document resulting from software design documentation derived from UML documents. As in other studies in this area, this study also reveals that the more complete UML documents of the software in the project, the more accurate the FP calculation results obtained (Irawati & Mustofa, 2012).

Recently, some metrics have emerged to estimate the effort spent on Web applications development. Typically, these metrics are adaptations of the FP method to the context of Web applications. Barabino *et al.* proposed a Web framework points that is a hybrid methodology, composed of a sizing phase, which follows specific guidelines, and an effort estimation phase, obtained by applying a cost model to the size model of the project to estimate. According to the authors, the sizing of the project takes into account not only usual functional requirements, as in FPA, but also specific elements for developing a Web application through Content Management Framework (CMF) (Barabino et al., 2015). In another research, Rajankrupa and Srinath, presented a study to obtain an estimating method for e-learning projects effort based on FP. In their papers, the authors describes how to estimate the size and effort of the three different types of projects like simple, medium and complex using FPA (Rajankrupa & Srinath, 2015).

Another interesting research using the FPA method carried out in an educational context was carried out by Santos and Oliveira. In their study, the authors combined gamification and evaluation the use of the FPA technique in software quality subjects. In this study, the authors aim to use the concepts of gamification to stimulate the support for teaching and engaging students' motivation in the subject of software quality taught in the graduate course in computer science at Universidade Federal do Pará (UFPA). For this, classes were defined to teach the FPA technique that used elements of games as motivation for the students. With this study, the authors had a double objective, on the one hand, to use of the elements of gamification and, on the other hand, to teach the concepts of the FPA technique to students so that they are better prepared in this area of knowledge (Santos & Oliveira, 2018, 2019).

The concern of Shah *et al.* is the presence of inconsistent states of software artifacts i.e., some of the classes are completely developed, some are partially developed, and some are not developed yet. In their research, the authors proposed a new model, the Function Point Analysis for Software Development Phase

(FPA-SDP) model and then they applied this new model in an empirical study to overcome this challenge. According to the authors, "the results of FPA-SDP model can help software project managers in: (i) knowing the inconsistent states of software artifacts (ii) estimating the actual size of a change request with its complexity level for software development phase" (Shah, Kama, & Ismail, 2018).

In a work performed by Kumawat and Sharma, the FPA method is used for calculating the cost of re-engineering and reproduction applications. After applying the FPA method to the re-engineering and reproduction applications, the authors generated a comparison graph which compares the processing complexity, Unadjusted Function Points (UFPs), and total FPs (Kumawat & Sharma, 2019).

In a paper published in 2019, Hillman and Subriadi did some reflections about FPA method and the current software development needs, namely, the real-time and the multimedia applications. They findings indicate that current software development does not only focus on system functionality, but non-functionality factors also begin to influence the value of a system for the users. According to the authors, "the FPA method is not adequate for current software development, especially in real-time and multimedia applications because of the inability to measure non-functional factors" (Hillman & Subriadi, 2019).

Freitas *et al.* proposed a method to improve reproducibility whilst keeping accuracy for the FPA method. The proposed method is based on a new artifact model called Function Point Tree (FPT). According to authors the FPT model enables a standardized and systematic collection of all data required for FP counting. Based on this new model, the authors developed a new measurement method and call them, Function Point Tree-based Function Point Analysis (FPT-FPA). Freitas *et al.* also implemented a prototype tool to show the feasibility of automation of the proposed method as well as to support its evaluation and then, they conducted an empirical study to evaluate FPT-FPA. According to the authors, their results "show general coefficients of variation lower than the maximum expected for both reproducibility and accuracy when compared to the standard FPA method" (Freitas, Fantinato, Sun, Thom, & Garaj, 2020).

In this subsection we present the main concepts related to software metrics. The main metrics were presented, but special emphasis was given to the UCP and FPs metrics. For these metrics, in a chronological way, the main research produced by the scientific community were presented.

3.5. Conclusion

Compared to the history of humanity, it is very recent, but in terms of the history of information technology, software development processes belong to prehistory. In fact, since the 50s of the last century,

there have been models of software development. To Know these models and to frame their evolution throughout history is crucial, even more so in the case of research work in the technologies and information systems area. After to present the basic concepts involved in software development process, we present an overview of the main models proposed over time by various researchers. These software development models are classified into traditional, plan-driven models, iterative and change-driven models, and agile models.

As is well known, there is always a reluctance to change, and bridges must be built to take advantage of the good that has been done in the past to improve future processes. Currently, as more empirical evidence on the agile methodologies is available, it seems that the main arguments for and against their use is nowadays not so much about their benefits, but rather about the need to extend their scope and adapt them to organizations with established and mature plan-driven processes (Boehm & Turner, 2005).

After to present some essential definitions about project management collected from the literature, we present the main project management approaches. In fact, we present an overview of the PMBOK and PRINCE2 project management guides that are well-knowing by the project managers. For questions that have to do with the research developed in this thesis, greater emphasis was given to the PMBOK. We decided to classify the project management approaches into traditional and agile. For each of them, the chronological framework and the main authors involved were given. The idea was to present the main aspects of these references and contextualized them within the work of the thesis.

For various project of software development, the size, effort, and cost are essential aspects for planning schedules and control software development to achieve the preferred outcome. These aspects require a method to measure software accurately and reliability. In this sense, we present the main software metrics, but special emphasis was done to Use Case Points (UCP) and Function Points Analysis (FPA) metrics. For each of these metrics, the most relevant research works were presented in a chronological way. We present some studies that propose new metrics that result from adaptations of the previous metrics (UCP and FPA metrics) and that are applied to specific contexts, namely, to Web, real-time and multimedia applications.

Chapter 4

Empirical Software Engineering in Teaching

4.1. Introduction

In this chapter, an overview of the Empirical Software Engineering (ESE) in teaching is presented. After the introduction, we present the main issues about the empirical software engineering in educational context. We present also, the advantages and disadvantages of using students and professionals in empirical studies. We present with some detail a framework for classifying empirical studies performed in educational context. Fundamentally based on the work of the previous section the lessons learned are presented. The chapter ends with some conclusions about progress and challenges related with the empirical software engineering in teaching.

We address some special features of the Empirical Software Engineering in teaching. We will present a framework to classify empirical studies performed in our educational environment context. We adapted Basili framework of experimentation (Basili et al., 1986) for educational context.

Empirical studies are important in software engineering to evaluate new tools, techniques, methods, and technologies in a structured way before they are introduced in the industrial (real) software process.

Within this PhD thesis, we developed a framework of a consistent process for involving students as subjects of empirical studies in the software engineering area. In concrete, our experiments with software development teams composed of students that will analyze how RUP (Rational Unified Process) processes can be compliant with the CMMI (Capability Maturity Model Integration), namely in the context of MLs (Maturity Levels) 2 and 3. Additionally, in this environment, we will also analyze the influence of project management tools to improve the process maturity of the teams. Our final goal of carrying out empirical studies with students is to understand its validity when compared with the corresponding studies in real industrial settings.

Unlike other mature disciplines, the field of Software Engineering (SE) continues to lack a research and development infrastructure that supports systematic testing of novel software engineering methods, techniques, and tools.

In the early nineties, Basili *et al.* introduced, for the first time, the concept of *experience factory*. As the authors refer in (Basili et al., 1992) the concept was introduced to "*institutionalize the collective learning of the organization that is at the root of continual improvement and competitive advantage*". Thus, the *experience factory* provides an organizational schema for collecting experiences on reuse of empirical results, for analyzing them and generalizing the knowledge contained (Visaggio, 2008). This scheme was designed based on many years of the Software Engineering Laboratory (SEL) work. Over several years, this well-known laboratory has conducted several studies and experiments for the purpose of understanding, assessing, and improving software and software processes within a production software development environment at the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) (Basili et al., 1992).

With our approach, we do not intend to create a new software engineering laboratory. Instead, we created an educational environment that allows us to conduct empirical studies in the software engineering area by involving students that are enrolled in our current software engineering courses (undergraduate (BA/BSc), graduate (MA/MSc) and doctorate (PhD) degrees). Our goal was to create a platform that allows us to perform empirical evaluations of the tools, techniques, and technologies used in software engineering.

Our intention was to develop a new experience factory approach based on one explicit educational environment. In this stage of our work, we just worked with students as subjects of our first empirical studies, but in future we intend extend these studies using professionals as subjects. We are fully aware that we will face some problems with the validation of the results that obtained in our student-based experiments performed. It is impossible to be sure that techniques evaluated under such circumstances will scale up to industrial size systems or very novel software engineering problems. Even though, Kitchenham *et al* say that "*students are the next generation of software professionals and, so, are relatively close to the population of interest*" (Kitchenham et al., 2002). In the opposite, students in psychology studies are not representatives of the human population as a whole (Rosenthal, 1994).

4.2. ESE in an Educational Context

ESE is a sub-field of software engineering which aims at applying empirical theories and methods for the measuring, understanding, and improvement of the software development process in real software companies (Jaccheri & Osterlie, 2005). This definition extends the concept for ESE proposed by Basili, when he said that "*experimentation is performed in order to help us better evaluate, predict, understand, control, and improve the software development process and product*" (Basili et al., 1986).

It is important to be able to evaluate new techniques and methods in a structured way before they are introduced in the software process (Höst, 2002). Empirical methods have gained increased attention in software engineering; there are dedicated conferences such as the *International Conference on Evaluation and Assessment in Software Engineering* (EASE) (EASE, 2021), and there are dedicated journals such as the *International Journal of Empirical Software Engineering* (EMSE, 2021).

Controlled experiments are the most commonly used empirical methods in software engineering. Sjøberg *et al.* define controlled experiment in software engineering as a "*randomized experiment or a quasi-experiment in which individuals or teams (the experimental units) conduct one or more software engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages, or tools (the treatments)*" (Sjøberg et al., 2005).

Currently, some universities offer courses in the Software Engineering area, as is the case of Norwegian University of Science and Technology (NTNU) (Jaccheri & Osterlie, 2005; NTNU, 2021). This university has a research unit called Information Systems and Software Engineering (ISSE) (NTNU_ISSE, 2021)that perform several empirical studies in the ESE area.

Another university that has a research group in the Software Engineering area is the Lund University in Sweden (SERG, 2021). This university has a Software Engineering Research Group (SERG) known for conducting relevant and rigorous research on the large-scale engineering of software systems, which have significant impact on industry and society. They use, particularly, empirical research methodology for long-term industry-academia collaboration. These two institutions have worked with students as subjects of experiments. These institutions run the experiments out of the courses' context, whereas in our approach the students perform the experiments as part of their regular academic courses.

The SERG is one of the two main research centers for ESE in Sweden. They conduct industry relevant research, published with the highest international standards. They focus their work on Software Startups,

Requirements Engineering, Software Testing, Continuous deployment and experimentation, Industry-academia collaboration, Cognitive aspects of software development, Open source and data ecosystems, Software Process Quality, and Software Management. They have also, some research in study aspects of the Digital Society in cross faculty settings. SERG conduct research with close industry contacts using empirical methods, including surveys, experiments, and case studies. They teach undergraduate and graduate courses in software engineering, which provide knowledge and understanding of large scale industry contexts (SERG, 2021).

The Department of Computer Science of the University of Helsinki created an experimental software laboratory for basic and applied software development research and education. The name of this laboratory is *Software Factory* and they involve researchers, students, and industry partners in their projects (Software Factory, 2021). This infrastructure allows students to use state-of-the-art tools, modern processes, and best practices to prototype and develop great software for businesses in an environment made to support top-level research.

The *Software Factory* is run by the Empirical Software Engineering Research Group (ESE) (ESE Helsinki, 2021)at the University of Helsinki in Finland. This research group address software engineering research problems and challenges with industrial relevance or origin. They emphasize the empirical aspect of the research, in particular by applying research methods that enable them to gain an in-depth understanding of software development.

Software Factory's research trains PhD and Master students, performs basic and applied research in its operating context and performs tests for evaluating different research methods. *Software Factory* offers a PhD student an opportunity to develop his/her capabilities to conduct observation and participant observation, perform interventions and conduct interviews in a scientific manner. The PhD student can perform initial tests to their research design and see where it can be improved before conducting field studies on a large scale (Abrahamsson, 2010). As well Known the research in software engineering requires an authentic environment. At the *Software Factory*, they aim to provide an environment that emulates industry as closely as possible.

Table 4.1 shows a strategic outlook of the *Software Factory* for all involved stakeholders (Kettunen, 2010). According to Petri Kettunen, *Software Factory* offers advances for both academic as well as industrial interest groups in terms of software business, research, and training/education. This goal is aligned with the overall strategy of University of Helsinki for fostering new linkages towards business life since 2010.

Table 4.1: Strategic Outlook of the Software Factory

Interest Group	Learn	Share	Grow
Companies	Global software development	Open Innovation Future skill and competence needs	Business-driven prototypes
Investors	Software product development	Prospects Expectations	Spin-offs
Researchers	Software business	Experiments	Cloud software competences
Students	Professional skills (e.g. teamwork)	Teaching material Lessons learnt	Careers

Based in software technology advances, such as cloud computing and internet-based networked software services, the strategic intent of the *Software Factory* is to promote (Kettunen, 2010):

- Learning: Train skilled and competent software professionals for the software-intensive industry;
- Sharing: Facilitate intensive industrial-academic collaboration and relevant research knowledge creation;
- Growth: Develop innovative software product ideas and prospective software business ventures.

In the *Software Factory*, the team works constantly together just like in a real workplace. There is always a real business demand behind the project, which makes the project context valid for research. At the same time the researchers can observe the team members and even participate in projects if participant observation is considered useful (Software Factory, 2021).

Methods, practices, and tools used in *Software Factory* has some particularities. Researchers can bring their own research equipment (*e.g.* mobile devices that the team members use). Researchers can ask participants to answer questionnaires (paper or web) during the project. All written material (*e.g.* code and documentation) provided by the team is available for the research purposes. Researchers can come to the *Software Factory* room and do direct observation. Researchers can also take part in the projects and do participant observation. Researchers can interview team members and customers after the projects (Software Factory, 2021).

In Brazil, Guilherme Travassos leads an active group in ESE area. *Experimental and Software Engineering Group* (ESE Group) is one of research and development groups of the Systems Engineering and Computer Science Department of the Universidade Federal do Rio de Janeiro (UFRJ/COPPE). ESE Group is a research line of the Laboratory of Software Engineering (LENS) of that university (ESE Group, 2021a).

ESE Group aims at improving software engineering through applying experimentation (scientific method) for the construction and evaluation of software technologies (processes, methods, techniques, and tools). ESE Group also concerns with the field advance, by researching and proposing new models for the planning, implementation, and packaging of software engineering studies. These activities are fundamental for software engineering to increasingly incorporate the principles of engineering. ESE Group believes these are fundamental activities that will contribute to get software engineering more closed to the classical engineering and scientific principles (ESE Group, 2021a).

The current research interests of the ESE Group are (ESE Group, 2021a):

- Experimental Software Engineering:
  - Methodological support for conducting studies in Software Engineering;
  - Environments for Software Engineering and experimentation.
- Experimentation applied to Software Engineering:
  - Search-Based Software Engineering;
  - Ubiquitous systems engineering;
  - Web application engineering;
  - Agility in Software Engineering;
  - Software maintenance and evolution;
  - Verification, validation, and software testing;
  - Software processes for innovation.

As previously mentioned, the ESE group has its focus on the development of technologies (approaches, strategies, guidelines, methodologies, models, processes, among others) to support the planning, execution, and analysis of primary, secondary, and tertiary studies in Software Engineering. The computational environment developed by this research group is called eSEE (Experimental Software Engineering Environment), whose objective is to provide an environment (not necessarily just a set of integrated tools) to organize knowledge about experimentation and to support the conduction of large-scale (and science) studies in Software Engineering area (ESE Group, 2021a).

In (ESE Group, 2021b) they have a glossary of concepts very useful for anyone that do research in this area. For being aware of the intervention area of the ESE Group, we list some research and development projects that the ESE laboratory is involved in (ESE Group, 2021a):

- Softex MPSBR, is a project to improve the software development capacity in Brazilian companies. Its main objective is to develop and disseminate a Brazilian process improvement model (the MPS (Melhoria do Processo de Software) reference model) aiming to establish an economically viable path for organizations, including small and medium-sized companies, to achieve the benefits of process improvement and the use of good software engineering practices in a reasonable amount of time.
- eSEE: Experimentation in Software Engineering and Science on a Large Scale is a project that has the main goal building the eSEE (*Experimental Software Engineering Environment*), a computational infrastructure to support experimentation in Software Engineering and science on a large scale.
- Dogs Project, the main goal of this project is to organize a DevOps software ecosystem to support the collaboration between software developers and IT operations staff, in the context of a startup-centric social environment. In particular, the project will address security, tool interoperability and data portability issues to enable data-driven knowledge transfer between development and operations staff.
- CAcTUS Project, the goal of CAcTUS (Context-Awareness Testing for Ubiquitous Systems) project is to answer questions regarding ubiquitous systems, trying to define a strategy for planning and executing testing procedures considering the potential contexts of use and using measurements.

Another important group of research in ESE area is the International Software Engineering Research Network (ISERN) group. ISERN is a community that believes that the software engineering research needs to be performed in an experimental context. ISERN members argue that by doing this we will be able to observe and experiment with the technologies in use, understand their weaknesses and strengths, tailor the technologies for the goals and characteristics of particular projects and package them together with empirically gained experience to enhance their potential reuse in future projects. The founding ISERN members chose the *Quality Improvement Paradigm* (QIP) as the reference model to provide a common terminology for their cooperation (ISERN, 2021).

In (ISERN, 2021) we can read the "*ISERN Manifesto*" that describes with some detail contents such as: Purpose and Focus, Common Research Framework, Activities, Communication, Members, Membership Application, Meetings and Meeting participation, and Benefits from Network, to researchers and to companies.

The ISERN community holds annual meetings since 1993 in different countries of the world. For instance, in 2019, the ISERN meeting has taken place in Pernambuco, Brazil (ESEIW 2019) and in 2020, previously scheduled for Bari, Italy, it was taken in virtual mode due to the COVID-19 pandemic (ESEIW 2020).

ISERN was initially created by relevant researchers in the ESE area from different parts of the world, namely: Prof. Dr. Koji Torii (Nara Institute of Science and Technology, Japan), Prof. Dr. Dieter Rombach (Fraunhofer IESE & University of Kaiserslautern, Germany), Prof. Dr. V. R. Basili (University of Maryland at College Park, USA), Prof. Dr. Ross Jeffery (University of New South Wales, Australia), Prof. Dr. Giovanni Cantone (University of Roma at Tor Vergata, Italy) and Dr. Markku Oivo (University of Oulu, Finland). Numerous other academic and industrial organizations have joint ISERN since. A list of all current ISERN members can be obtained in (ISERN, 2021).

Fraunhofer Institute for Experimental Software Engineering (Fraunhofer IESE) is one of the founding organizations of Empirical Software Engineering (IESE, 2021). Fraunhofer IESE is one of 75 institutes and research institutions of the Fraunhofer-Gesellschaft. Together they have a major impact on shaping applied research in Europe and contribute to Germany's competitiveness in international markets (Fraunhofer, 2021). As one of the founders and drivers of the ISERN, a unique network of more than 45 international software engineering research groups, Fraunhofer IESE have access to all relevant research groups and experts world-wide in ESE area. Fraunhofer IESE is one of allied institutes of the Department of Computer Science of the Kaiserslautern University. Fraunhofer IESE's mission is to promote experimental software engineering, the best approach for introducing engineering style rigor into business practice (Rombach, 2000).

Empiricism is one of the major competencies of Fraunhofer IESE. They support the customer in all steps of technology evaluation, and they have profound experience in conducting primary studies such as surveys, case studies, experiments, project retrospectives, and post-mortem studies. They support companies in mastering challenges in the areas of "Autonomous Systems", "Industry 4.0", "Smart Farming", and "Digital Healthcare", and offer digital solutions for rural and urban areas. In over 2,000 customer projects, they have already transferred cutting-edge research into sustainable business practices, with current focus topics being "Dependable AI (Artificial Intelligence)", "Digital Ecosystems", "Virtual Engineering", and "System Modernization". Their portfolio comprises quantitative as well as qualitative methods, and triangulation (IESE, 2021).

Fraunhofer IESE perform several services ranging from planning and designing empirical studies, implementing data collection instruments, supervising the conduction of empirical studies, analyzing the

results of empirical studies, documenting empirical studies, and establishing the infrastructure and competencies for conducting your own studies.

Fraunhofer IESE also provides a comprehensive support during the lifecycle of secondary studies, such as systematic literature reviews and meta-analysis approaches. They provide methods that allow synthesizing (i.e., summarizing and generalizing) evidence from individual empirical studies.

Fraunhofer IESE also plays an important role in the technology transfer and piloting. They conduct demand analyses to identify key problems and demands in practice. They guide the development of new technologies by using empirical methods to systematically support the tailoring and transfer of technologies.

Simula Research Laboratory is another research institution that works closely with the Norway universities. Simula was founded in 2001 and has since become a nationally and internationally acclaimed research institution. Simula's main objective is to create knowledge about fundamental scientific challenges that are of genuine value for society. The strong focus on basic research is combined with both teaching of postgraduate students and the development of commercial applications (Simula, 2021).

Simula develops its research work in Communication Systems, Software Engineering and Scientific Computing fields. All activity within the research field of software engineering at Simula is concentrated in the Department of Software Engineering, and this research group focuses on the development of tools and methodologies for the validation and verification of large-scale systems, thereby aiming to ensure greater systems stability. They main activities has three vectors: research (Simula Research Laboratory, Simula Metropolitan, and Simula UiB), education (Simula School of Research and Innovation (SSRI)) and innovation (Simula Garage, Simula Innovation and Simula Consulting) (Simula, 2021).

Simula was organized as a limited company owned by the Ministry of Education and Research of Norway government. Simula's main objectives are:

- To conduct basic and long-term research in the fields of scientific computing, software engineering, machine learning, communication systems, and cryptography;
- To educate students at the master, doctoral, and postdoctoral levels in partnership with Norwegian and international universities;
- To promote the application of research in both private and public sectors.

Simula is involved in many projects with public organizations and industrial settings. In all of these projects, the main objective is to apply research and technology transfer.

As shown, in this section we present the most relevant research institutions that involve students in their empirical studies. Probably, there other institutions that would be present in this section, but our research focus was centred in the best ones.

4.3. ESE using Students versus Professionals

In this section, based on literature review we will describe the strengths/weaknesses of using students versus professionals in the ESE context.

In the survey conducted in (Sjøberg et al., 2005), a total of 5,488 subjects took part in the 113 experiments investigated, eighty-seven percent were students and nine percent were professionals. This survey demonstrates the importance of using students in this context. Table 4.2 shows the subject types divided into the categories involved in all experiments. The authors quantified, also, the topics of the experiments and their subjects, tasks, and environments.

Table 4.2: Subject categories using in experiments (adapted from (Sjøberg *et al.*, 2005))

Subject Category	Reported Subject Types	N	%
Undergraduates	Undergraduates, Bachelors, Third and fourth-year students, Last-year students, Honors and Majors	2969	54.1
Graduates	Graduate students, Students following graduate courses or Master's programs, MSc and PhD students	594	10.8
Students, type unknown	Students in computer science, Students	1203	21.9
Professionals	Developers, Practitioners, Software engineers, Analysts, Domain experts, Business managers, Facilitators, Professionals	517	9.4
Scientists	Professors, Post-doctorates, Staff members of educational institutions	74	1.3
Unknown		131	2.3
	Total	5488	100

The lack of professionals in software engineering experiments is due to the conception of high costs and large organizational effort. Usually, companies do not provide their professionals to carry out empirical studies. Port and Klappholz refer that very little empirical research is done in industry and that, for competitive reasons, even when such research is done, its results are often withheld as proprietary or are too difficult to contextualize without divulging sensitive information (Port & Klappholz, 2004).

In many studies, students are used instead of professional software developers, although the objective is to draw conclusions valid for professional software developers. The differences are only minor, and it is

concluded that software engineering students may be used instead of professional software developers under certain conditions.

Höst *et al.* argue that the main reason to use students as subjects is often that they are available at universities and they are willing to participate in studies as part of courses they attend (Höst et al., 2000). In many cases, it is possible to combine the learning objectives of the courses with the research objectives of the studies.

Tichy refers that software students are much closer to the world of software professionals than psychology students are to the general population (Tichy, 2000). In particular, software graduate students are so close to professional status that the differences are marginal. Software graduate students are technically more up to date than the "average" software developer who may not even have a degree in computing. Software professionals, on the other hand, may be better prepared in the application domain and may have learnt to deal with systems and organizations of larger scale than a student.

Sjøberg *et al.* argue that the main reason of most subjects in software engineering experiments are students is that they are more accessible and easier to organize, and hiring them is generally inexpensive (Sjøberg et al., 2002). Consequently, it is easier to run an experiment with students than with professionals and the risks are low. The bad thing is that the variations among studies conducted with professionals are higher than the variations among students due to the more varied educational backgrounds and working experiences in the professionals.

Jaccheri and Morasca refer that empirical studies are often carried out with students because they are viewed as inexpensive subjects for pilot studies. The authors also argue that the final goal of carrying out empirical studies with students is carrying out empirical studies in industrial organizations and establishing collaborations with them. In this sense, it is therefore useful to involve industry professionals in empirical studies with students, and they should actually play all of the stakeholders' roles. When involved in empirical studies, the authors propose the following roles for professionals, namely: Professionals as students, Professionals as customers, Professionals as researchers, Professionals as teachers (Jaccheri & Morasca, 2006).

Svahnberg *et al.* refer that the students are readily available, often willing to participate, and require no or little compensation. The authors carried out an empirical evaluation using students in requirement engineering area. They investigated what students imagine is important to professionals in requirements selection. The reason for this investigation was to understand whether the students are able to picture what

industry professionals value, and whether the courses allow them to picture the state of industry practice. The results that authors obtained indicate that students have a good understanding of the way industry acts in the context of requirements selection, and they suggest that the students may work well as subjects in empirical studies in this area (Svahnberg, Aurum, & Wohlin, 2008).

Carver *et al.* have developed a checklist that provides guidance for researchers and educators when planning and conducting studies in university courses. According to the authors, empirical studies with students can be valuable to the industrial and research communities if they are conducted in an adequate way, address appropriate goals, do not overstate the generalizability of the results, and take into account threats to internal and external validity (Carver, Jaccheri, Morasca, & Shull, 2010). The empirical study must be carefully planned, executed, and integrated into the course, otherwise, the study will neither provide the students with much educational value nor the researcher with much scientific value.

Based on a literature review of research and pedagogy, Carver *et al.* identified requirements for successful empirical studies with students. Based on these requirements (see Table 4.3) and the authors' experiences as the basis, they create a checklist for planning and conducting an Empirical Study With Students (ESWS). This checklist was designed to help both novice and experienced researchers keep the preparation tasks organized (Carver et al., 2010).

Table 4.3: Empirical Study With Students requirements (Adapted from (Carver *et al.*, 2010))

Req.	Description
R1	External validity issues must be consciously considered.
R2	The ESWS must be properly integrated with the course.
R3	Ethical issues must be adequately addressed by the study design.
R4	The correct goal must be chosen for the study based on its environment.
R5	The study setting must be appropriate relative to its goals, the skills required and the activities under study.
R6	The effect of differences between the subject population and the target population must be discussed.
R7	Students should learn the value of using empirical studies to evaluate products and processes and how to conduct them so they can later perform their own assessments.
R8	Group work or collaborative work should be included in an ESWS.
R9	ESWS should include development projects where possible.

Table 4.4 provides a high-level overview of the checklist items grouped by when they should occur. As we can see, the guidelines of the checklist must be performed over time considering the classes and the empirical study.

Table 4.4: Empirical Study With Students checklist (Adapted from (Carver *et al.*, 2010))

Items to consider when designing and conducting an ESWS	
Before the class begins	
1.	Ensure adequate integration of the study into the course topics.
2.	Integrate the study timeline with the course schedule.
3.	Reuse artefacts and tools as appropriate.
4.	Write up a protocol and have it reviewed.
As Soon as the Class Begins	
5.	Obtain subjects' permission for their participation in the study.
6.	Set subject expectations.
When the Study Begins	
7.	Document information about the experimental context in detail.
8.	Implement policies for controlling/monitoring the experimental variables.
When the Study is Completed	
9.	Plan follow-up activities.
10.	Build or update a lab package

In our experimental environment, we have been used some versions of this checklist to support the design and execution of empirical studies with students. Our empirical studies always related with software process improvement and project management issues.

Based on the literature review we built a table that summarize the differences between students and professionals when used in ESE studies. Table 4.5 provides a classification of the main features that distinguish students from professionals.

Table 4.5: Students versus professionals as subjects in ESE studies

Feature	**Student**	**Professional**
Access	easy	difficult
Participation	easy	difficult
Experience	little or no	a lot of
Cost	inexpensive	expensive

4.4. Experimentation Framework to Design Empirical Studies

In this section, we present a framework for classifying empirical studies in educational context. Firstly, we introduce the original Basili *et al.* framework of experimentation. We present also, some papers, which are rare in the literature, that explore that framework. Follow, we describe the relevance and objectives of the framework. We then present the framework architecture, and we explain in some detail their different components. Follow, we present the results of the framework application obtained in the empirical studies classification of our student's works. This empirical studies was performed in educational context and in Software Engineering area.

In *Experimentation in Software Engineering* Basili *et al.* paper, the authors presented the first framework for analyzing most of the experimental work performed in software engineering over the past several years (Basili et al., 1986). With the work published in that paper, Basili *et al.* had three overall goals: (1) describe a framework for experimentation in software engineering. This framework is intended to help structure the experimental process and provide a classification scheme for understanding and evaluating experimental studies; (2) classify and discuss a variety of experiments from literature according to the framework; (3) identify problems areas and lessons learned in experimentation in software engineering.

The Basili *et al.* framework of experimentation consists of four categories corresponding to phases of the experimentation process: (1) definition, (2) planning, (3) operation, and (4) interpretation (Basili et al., 1986).

The first phase of the experimental process is the study definition phase. The study definition phase contains six parts: 1) motivation, 2) object, 3) purpose, 4) perspective, 5) domain, and 6) scope. The authors argue that most study definitions contain each of the six parts.

The second phase of the experimental process is the study planning phase. The experiment planning phase contain three parts: 1) design, 2) criteria, and 3) measurement. The design of an experiment couples the study scope with analytical methods and indicates the domain samples to be examined. The criteria and measurement parts in Basili *et al.* framework always appear related with cost/quality issues.

The third phase of the experimental process is the study operation phase. The operation of the experiment consists of 1) preparation, 2) execution, and 3) analysis. According to authors, preparation may include a pilot study to confirm the experimental scenario, help organize experimental factors, or inoculate the subjects. During the execution of the study, the experimenters collect and validate the defined data. In

the analysis of the data, the experimenters may include a combination of quantitative and qualitative methods. Before the formal data analysis, a preliminary screening of the data is performed, probably using plots and histograms. Before the application of the statistical models and tests, the process of analyzing the data requires the investigation of any underlying assumptions.

The fourth phase of the experimental process is the study interpretation phase. The interpretation of the experiment consists of 1) interpretation context, 2) extrapolation, and 3) impact. The results of the data analysis from a study are interpreted in a broadening series of contexts. These contexts of interpretation are the statistical framework in which the result is derived, the purpose of the particular study, and the knowledge in the field of research. The representativeness of the sampling analyzed in a study qualifies the extrapolation of the results to other environments. According to authors, several follow-up activities contribute to the impact of a study: presenting/publishing the results for feedback, replicating the experiment, and actually applying the results by modifying methods for software development, maintenance, management, and research.

Figure 4.1 presents a summary of the original experimentation framework. With this framework, Basili *et al.* classified a set of empirical studies involving students, novice, and experienced professionals. These studies were performed in several different software engineering areas such as software testing process, programming debugging, programming language features, software development approaches, among others.

Selby focuses his research on a subset of the measurement and experimentation issues related to frameworks, mechanisms, and infrastructure. In particular, his study highlights research issues or results in these areas: frameworks for measurement and experimentation, existing measures, determining appropriate measures, data collection, experimental designs, and infrastructure for measurement (Selby, 1993). His main concern was about the experimental process and measurement.

Bourque *et al.* tested the experimentation Basili's framework to verify its applicability in software engineering experiments conducted in industrial settings (Bourque & Abran, 1996; Bourque & Côté, 1991). These experiments are set in the areas of software size estimation, software sizing for small adaptive maintenance requests, and a third experiment which attempts to identify some of the fundamental characteristics of business software, using multidimensional statistical techniques. The authors found Basili framework very beneficial and, therefore they believe that it could form the basis for a standard on software engineering experimentation.

Hayes developed a framework (Hayes, 2002) based in Basili *et al.* framework of experimentation for describing and evaluating a real-world projects. These projects were performed by software engineering

students at the University of Kentucky (UK). With UK Medical School support, the students undertook a project to develop a phenylalanine milligram tracker.

I. Definition					
Motivation	**Object**	**Purpose**	**Perspective**	**Domain**	**Scope**
Understand Assess Manage Engineer Learn Improve Validate Assure	Product Process Model Metric Theory	Characterize Evaluate Predict Motivate	Developer Modifier Maintainer Project manager Corporate manager Customer User Researcher	Programmer Program/project	Single project Multi-project Replicated project Blocked subject-project
II. Planning					
Design			**Criteria**		**Measurement**
Experimental designs Incomplete block Completely randomized Randomized block Fractional factorial Multivariate analysis Correlation Factor analysis Regression Statistical models Non-parametric Sampling			Direct reflections of cost/quality Cost Errors Changes Reliability Correctness Indirect reflections of cost/quality Data coupling Information visibility Programmer comprehension Execution coverage Size Complexity		Metric definition Goal-question-metric Factor-criteria-metric Metric validation Data collection Automatability Form design and test Objective vs. subjective Level of measurement Nominal/classificatory Ordinal/ranking Interval Ratio
III. Operation					
Preparation			**Execution**		**Analysis**
Pilot study			Data collection Data validation		Quantitative vs. qualitative Preliminary data analysis Plots and histograms Model assumptions Primary data analysis Model application
IV. Interpretation					
Interpretation context			**Extrapolation**		**Impact**
Statistical framework Study purpose Field of research			Sample representativeness		Visibility Replication Application

Figure 4.1: Summary of the experimentation framework (Adapted from (Basili *et al.*, 1986))

Hayes considered the use of the course project as an experimental study. According to author, these kinds of projects have certain aspects. These aspects or characteristics can serve as a framework for structuring, describing, and evaluating any project. The aspects provide a scheme that can be used to understand projects, compare projects, or evaluate projects and look for areas of improvement. Also, according to the author, the Basili *et al* framework of experimentation addresses many of the aspects of a course project, and then can be used as a starting point. Hayes refers that there are advantages to doing so.

It ensures that his framework is in keeping with published, well-grounded work. It may encourage instructors to use course projects for more than just student grades, to apply experimental software engineering principles and use projects as part of their research (Hayes, 2002; Hayes & Dekhtyar, 2005).

Hayes enhanced the original Basili *et al* framework by adding parts to phases and by adding levels to many of the parts (see italics in Figure 4.2). The resulting course project framework is summarized in Figure 4.2. As we can see, Hayes designated four phases: (1) definition, (2) planning, (3) realization, and (4) interpretation. In order to distinguish the new contribution proposed by Hayes in relation to the Basili *et al.* original experimentation framework, we put all new items in italic words.

Definition Phase I	**Planning Phase II**	**Realization Phase III**	**Interpretation Phase III**
Motivation Understand Assess Manage Engineer Learn Improve Validate Assure *Confirm* *Enhance*	**Project Design** *Problem Domain* *Problem Class* *Problem Complexity*	**Preparation** Pilot study *Artefact development* *Object development*	**Interpretation context** Statistical framework Study purpose Field of research
Purpose *Implement* *Test* Characterize Evaluate Predict Motivate	**Experimental design** Experimental designs Incomplete block Completely randomized Randomized block Fractional factorial Multivariate analysis Correlation Factor analysis Regression Statistical models Non-parametric Sampling	**Execution** *Project execution* Data collection Data validation	**Extrapolation** Sample representativeness
Object Product Process Model Metric Theory **Perspective** Developer Modifier Maintainer Project manager Corporate manager Customer User *Reliability Engineer*	**Criteria** Direct reflections of cost/quality Cost Errors Changes Reliability Correctness Indirect reflections of cost/quality Data coupling Information visibility Programmer comprehension Execution coverage Size Complexity	***Evaluation*** *Quantitative* *Qualitative* *Gold Standard* *Comparison* *Peer-Project comparison*	**Impact** Visibility Replication Application

	Academic *Institution* *Tester* Researche*r* *User Advocate* *Instructor*						
	Domain		**Measurement**		**Analysis**		
	Software Engineers *Reliability Engineers* Program/project		Metric definition Goal-question-metric Factor-criteria-metric Metric validation Data collection Automatability Form design and test Objective vs. subjective Level of measurement Nominal/classificatory Ordinal/ranking Interval Ratio		Quantitative vs. qualitative Preliminary data analysis Plots and histograms Model assumptions Primary data analysis Model application		
	Scope		***Process***				
	Single project Multi-project Replicated project Blocked subject-project		*Teams* *Individuals* *Lifecycle* *Methodology*				
	Importance		***Product***				
	Safety-critical *Mission-critical* *Quality of life* *Convenience*		*Documentation* *Code* *Executable* *Database* *Presentation* *Demonstrations*				
	End User						
	None *Instructor* *Real-world-like* *Real-world*						

Figure 4.2: Summary of the framework for course projects (adapted from (Hayes, 2002))

Hayes conclude that real-world problems are the best ones to assign as course projects and that these projects can be used as experimental studies also, with advanced planning and careful attention to the framework of the study/project.

Goulão and Abreu created a model to experimental software engineering process (Goulao & Brito e Abreu, 2007). According to the authors, this model is aligned with recent proposals for best practices in experimental data dissemination. They further argue that the model can be used in the definition of software engineering experiments and in comparisons among experimental results. Their model relates the process activities with their deliverables. The deliverables are mapped into an underlying logical model of experiment-related concepts that covers the information needs of the experiments' reporting guidelines proposed in (Jedlitschka & Pfahl, 2005), and are represented in their model description with UML (Unified Modeling

Language) class diagrams. The activities carried out during the process are described using UML activity diagrams.

The authors conclude that their model can be used as a guideline for practitioners involved in leveraging data collection activities to improve the software process in their organizations, both in industrial and educational contexts. They concluded also, that their model can may also be used as a framework for supporting experiments comparison, which was identified as one major need for future software engineering research (Goulao & Brito e Abreu, 2007).

4.4.1. Relevance and Objectives of the Framework

Fundamental principles of software engineering must be supported by theory and sound empirical research (Bourque & Abran, 1996). Thus, empirical studies in software engineering play an important and significant role in the evaluation of tools, techniques, methods, and technologies before they are dynamically validated in industrial setup.

As in all areas of science and engineering, empirical research can only be considered rigorous when it is conducted using a valid experimental approach or protocol. In this sense, we made an adapted framework to classify empirical studies performed in an educational context. The framework that we proposed for the first time in this paper (Alves, Ribeiro, & Machado, 2020), was based on the framework of experimentation proposed by Basili. The main goal of the authors was to build a framework for "analyzing most of the experimental work that has been performed in software engineering over the past several years". Their framework is generic, so it does not refer specifically to empirical studies involving students.

The main objective of the framework is to provide a classification scheme for understanding and evaluating empirical studies in software engineering area in an educational context. With this framework we can analyze most of the experimental work that has been done in our computing courses. The framework also has another purpose, it helps structuring experimental processes.

We involved students with BSc, MSc and PhD degrees in computing from our university in developing a software project requested by real clients. The educational approach is mainly based on Project-Based Learning (PBL) principles. Working in a team for a limited period of time and delivering a high-quality product are some of the skills that students should gain during their studies at the university.

After project presentations, students have a meeting with the teaching staff to discuss and analyze the strengths and weaknesses of their work. The analysis should be performed by keeping in mind the

documentation, the project plan and the work experience the team has gathered from the start to the end of the project. To learn from experience is the key to improvement.

Performing empirical studies in real contexts is very difficult due to various obstacles, thus, we created a stable environment that allows the completion of reliable empirical studies with students. In many studies, software engineering students are used instead of professional software developers, although the objective is to draw conclusions valid for professional software developers. The differences may be considered only minor, and it is concluded that software engineering students may be used instead of professional software developers under certain conditions.

Over the past eight years we had more than one thousand students involved in all computing unit courses. All these students, spread over several course units, develop software projects of medium to high complexity and constitute our main source of experimentation. In this thesis we present the results obtained from the classification using the framework of only one of our course units. From the academic year 2010/2011 to 2017/2018, the 105 students enrolled in the Project Management of Information Systems (PMIS) course developed 79 work projects (empirical studies) in different SWEBOK (Software Engineering Body of Knowledge) (Abran et al., 2004) KAs (Knowledge Areas). We classify these works projects to explain the usefulness and applicability of using the proposed framework.

4.4.2. The Framework Architecture

In this section, we present the architecture of our adapted framework along with an explanation of its main components. It is very important to have an overview of the framework to understand the scope of our research.

The main goal of our framework is to provide a classification scheme for understanding and evaluating empirical studies in PMIS area in an educational context. Firstly, we will use the framework to classify empirical studies performed by students of our courses and after a few iterations of refinement of our framework we will classify some experiments collected from the literature. The framework also serves as an instrument to guide us through the implementation of an experimental process with accuracy.

The framework summarized in Figure 4.3 consists of four categories corresponding to phases of the Empirical Study (ES) performed in an educational context: 1) definition, 2) planning, 3) operation, and 4) interpretation. The downward arrows show the order that we must perform in an ES. The arrows “iteration” and “new execution” reveal the dynamic aspect of the experimentation process. This can help the student to

correctly follow the process. Same as the arrows, the parts written in blue are included by us in this new proposed framework.

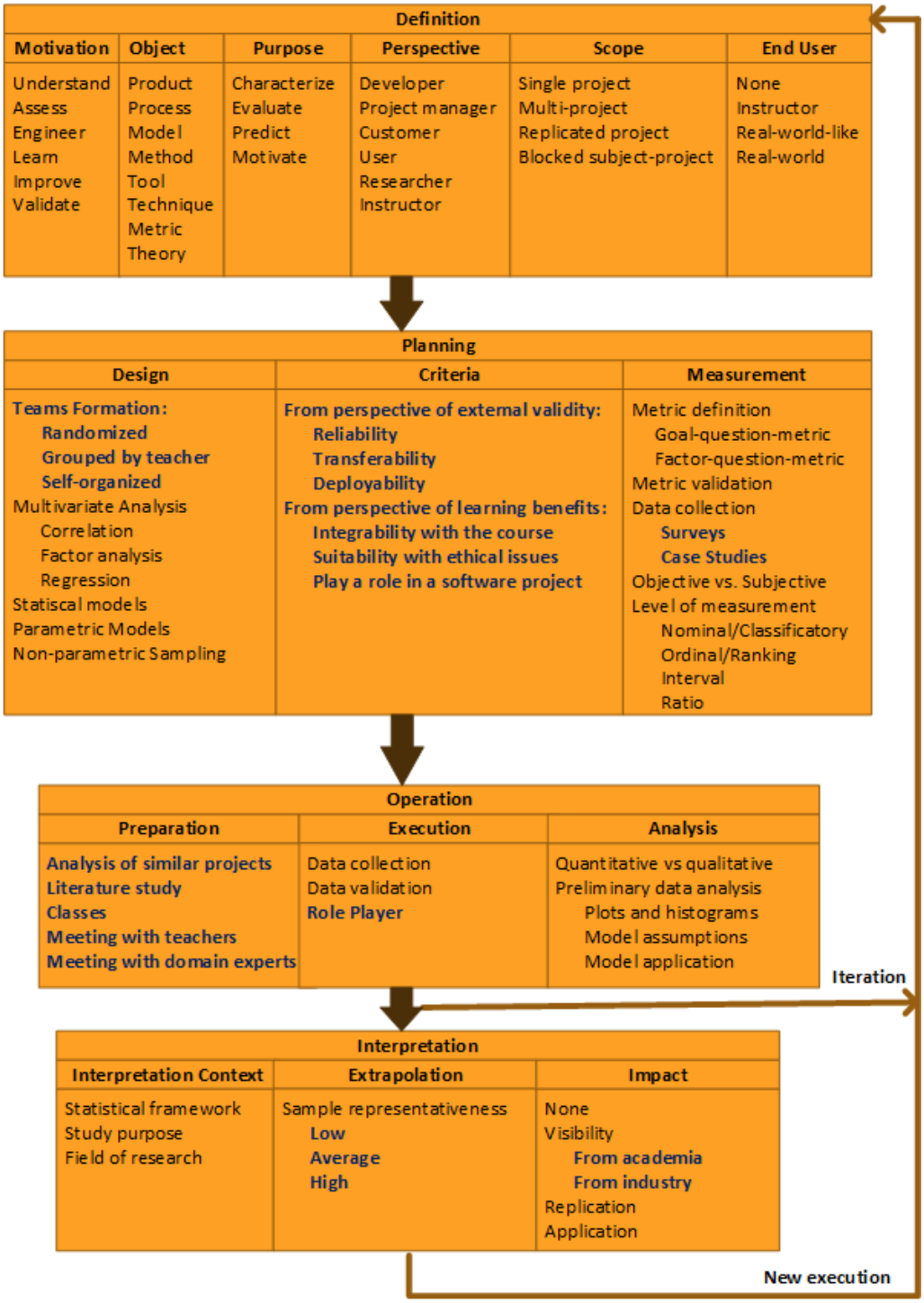


Figure 4.3: Framework for Cataloging Empirical Software Engineering Studies (Adapted from (Basili *et al.*, 1986))

The first phase of the ES is the definition phase. In our framework the definition phase contains six parts: 1) motivation, 2) object, 3) purpose, 4) perspective, 5) scope, and 6) end user. In this phase we excluded one part, the “domain” from the original framework. We think that this part is not important to classify empirical studies in an educational context. However, we added a new part called “end user” adapted from (Hayes, 2002). In this part we will define who uses the project work. The end user of the ES can be categorized as none, instructor, real-world-like, or real-world. The motivation of an ES can be to understand, learn, or assess the performance of a certain tool. The object of study is the primary entity examined in an ES. We are interested in characterizing the final product (software application) or understand and improve the software development process. In an educational context the teacher must provide a list of valid empirical studies, highlighting the object of study or validate a proposal made by the student. The purpose of an ES can be to characterize the change in development activities when using software management tools, to evaluate the effectiveness of these tools, to predict system development effort by using an estimation method, or to validate a theory by analyzing empirical evidence. Normally, empirical studies in the Information Technology (IT) area involve the following subjects: software engineering, computer science, information systems and project management. However, software engineering is a multi-disciplinary field that involves concepts from several disciplines. We think that these disciplines cover the domain of all empirical studies. In empirical studies that examine "software quality”, the interpretation usually includes correctness if it is from the perspective of a developer or reliability if it is from the perspective of a customer. Studies that examine metrics for a given project type from the perspective of the project manager may interest certain project managers, while corporate managers may only be interested if the metrics apply across several project types. The ES also can be performed from the researcher or instructor perspective. The scope of the ES can be categorized as single-project, multi-project, replicated project and blocked subject-project. This classification results from the number of teams and the number of projects involved in the ES. Thus, we have a single project when we have one team and one single project, multi-project variation one we have one team and more than one project, replicated project when we have more than one team and one single project and a blocked subject-project when we have more than team and more than one project. More details about this classification can be found in (Basili et al., 1986).

The second phase of the ES is the planning phase. In our framework the planning phase contains three parts: 1) design, 2) criteria, and 3) measurement. In this phase we maintain the original parts, but we change the features inside each part to be in accordance with an ES in an educational context. In the design of an

ES, we must set the team's formation and the analytical methods that we will use to analyze the data collected. We consider three types of team's formation: randomized, grouped by teacher and self-organized. Multivariate analysis methods, including correlation, factor analysis, and regression generally can be used across all empirical studies scopes. Statistical models may be formulated and customized as necessary. Nonparametric methods should be planned only when limited data is available or distributional assumptions are not met. Sampling techniques can be used to select representative students and projects to examine (Basili et al., 1986). We divide the criteria of the ES planning in two dimensions, one from the perspective of external validity and another from the perspective of learning benefits. The external validity is a major concern in any domain of the ES. It defines the conditions that limit the ability to generalize the results of an experiment to an industrial context. We can classify an ES according to reliability, transferability, and deployability. From the perspective of learning benefits, we must evaluate if the ES was properly integrated with the course, if ethical issues were adequately addressed by the study design and if students played a valid role in the software project environment (Carver et al., 2010). In the ES planning we must define measurements and validate them to show if they capture what is intended. The data collection process includes surveys, normally, interviews or questionnaires and case studies. The required data may include both objective and subjective data and different levels of measurement: nominal (or classificatory), ordinal (or ranking), interval, or ratio. As metric definition the ES can follow a gool-question-metric (Basili & Selby, 1984; Basili & Weiss, 1984) or factor-question-metric paradigm (Cavano & McCall, 1978).

The third phase of the ES is the operation phase. The operation of the ES consists of 1) preparation, 2) execution, and 3) analysis. Before conducting the actual ES, the student should analyze similar projects, review the literature about the object of study, participate actively in the classes, promote meetings with teachers and, if needed and possible, promote meetings with domain experts. Execution covers the actual project accomplishment by students as well as data collection, data validation (if it is also an experimental study), and verification of the student's role and its validity. The analysis of the data may include a combination of quantitative and qualitative methods. The preliminary screening of the data, probably using plots and histograms, usually precedes the formal data analysis. The process of analyzing the data requires the investigation of any underlying assumptions before the application of the statistical models and tests (Basili et al., 1986).

The fourth phase of the ES is the interpretation phase. The interpretation of the ES consists of 1) interpretation context, 2) extrapolation, and 3) impact. The contexts of interpretation are the statistical

framework of which the result is derived, the purpose of the particular study, and the knowledge in the field of research (Basili et al., 1986). The representativeness of the sampling analyzed in a study qualifies the extrapolation of the results to other environments. To classify the sample representativeness, we used a low, average, and high scale. This qualitative scale can easily be converted into a quantitative scale by scaling the sample size for each case. We classify the ES visibility from the academic perspective and from the industrial perspective. We must evaluate if an ES performed by a student can be publish in an international conference or if it should just be used inside university. We must evaluate if the results obtained from ES can be replicated by colleagues, or by professionals from the industry. We must also evaluate if the results from ES can be applied in real context.

4.4.3. Demonstration Case with the Framework

In order to validate our adapted framework, a demonstration case was performed with project works elaborated by students from the University of Minho. This study can be full consulted in the paper "Classifying Empirical Studies in an Educational Context Through an Experimentation Framework" published in the 12th annual International Conference on Education and New Learning Technologies (EDULEARN20) (Alves et al., 2020).

For the validation of our framework, we collected all projects carried out in the PMIS unit course in the period of 2010/2011 up to 2017/2018. In this period, 105 students enrolled in the PMIS course developed 79 projects. The last task of each project is to deliver a final document. This document in some cases has a report format and in other cases has a journal paper format. Students would have to follow the formatting as if they were submitting a paper to a journal, for example using a Springer template. Each one of these documents has between 20 to 50 pages. The total number of pages of all these documents, analyzed by the researcher (author of this thesis), are more than 1600. The work teams enrolled in these projects had one to four students.

In order to follow our approach, to accomplish the treatment and analysis of the results, three steps were executed, namely:

1. Perform a literature review about frameworks that define the experimentation process;
2. Adapt the best framework found in the previous step;
3. Classify the PMIS projects with the adapted framework.

In the step one we found the Basili *et al.* experimentation framework (Basili et al., 1986). Although it is a framework with many years of existence its core concepts are still actual. To create our adapted framework in step two we collect from literature some other adapted experimentation frameworks, although rare they still exist. The results of steps 1 and 2 can be found in previous section. In the third step we used the adapted framework to classify the projects performed by the PMIS unit course students.

Our main goal in using the framework is to achieve a critical improvement of the PMIS unit course. In fact, we intend to get answer to the following research questions:

1. Is the framework applicable to unit courses in an educational context?
2. Does the framework bring us relevant information about the PMIS unit course?

Hereafter we explore the data collected from the classification of all projects using our framework in order to answer those questions. The classification was made based on the reading of documents delivered by students. For this, all components of the framework were reproduced in the Microsoft Excel. The classification was carried out manually, probably involving some subjectivity on the part of the researcher.

Figure 4.4 presents the results of each part of the definition phase of the framework application in all projects. As a preliminary note we would like to clarify that some projects do not have only one value in each part. For example, work project *x* can have as motivation to understand and to learn some metric. This can happen in all different parts of all experimental phases.

Analyzing Figure 4.4, we can conclude that 65.6% of students have as their main motivation to understand and to learn something, a metric, a model, a technique, etc. This is expected because we are in an educational context.

The "object of study" is the primary entity examined in a study. In our classification we found that 39.6% of the projects handled any kind of process, for example the software process development or project management process. We also have a considerable percentage of projects focusing on the study of models, concretely 22.0%, for example, a software reliability model or a software quality model. Tools are the third object of study most chosen by our teams, concretely 15.4%. Some of these tools were applications to automatically calculate a measure to estimate the effort needed to develop a software product. Interestingly, we did not find any work that investigated a final software product or a theory.

The purpose of a project can be to evaluate effectiveness of a design process, characterize a system over time, predict the system development cost by a cost model, motivate the validity of a theory by analyzing

empirical evidence, etc. We find in our classification that students in their projects focus their attention in characterize (46.9%) some model or system or evaluate (40.6%) some metric proposed.

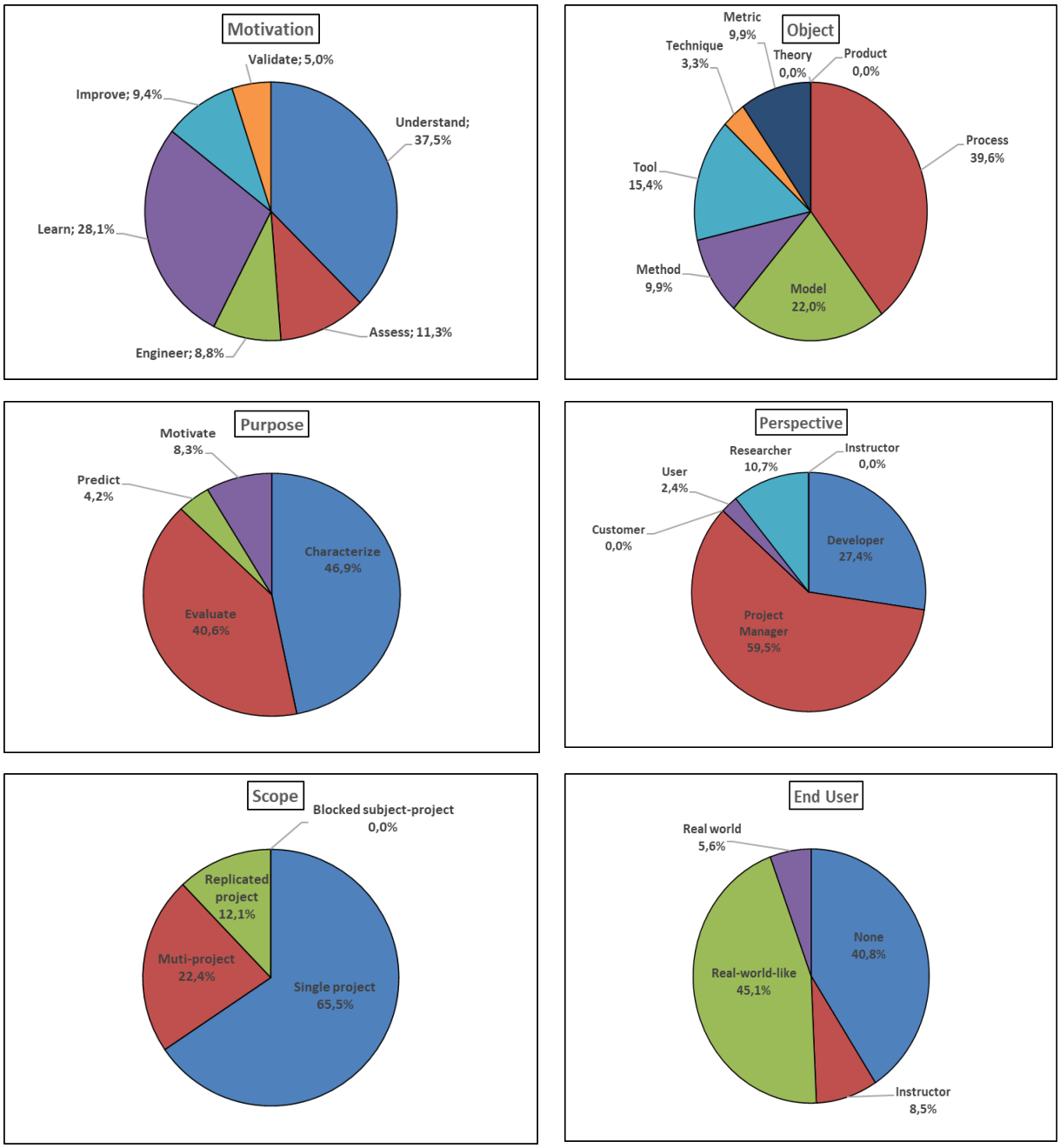


Figure 4.4: Definition phase: motivation, object, purpose, perspective, scope, and end user

Given that all projects were carried out within the scope of PMIS unit course then it is expectable that most of the work was performed from the perspective of the Project Manager (59.5%). Since all students were enrolled in computer science courses then it is also expectable that a considerable number of projects were

performed from perspective of the Developer (27.4%). It is also no wonder that none of the work has been carried out from the customer's perspective, as we are in an educational environment. Interestingly, we did not find any work performed from the perspective of the instructor.

In the scope part of the definition phase, we analyzed four different cases: (1) single project, when a single team (or a single-person) performs a single project; (2) multi-project, when a single team performs a set of projects; (3) replicated project, when more than one team performs one project and (4) blocked subject-project, when more than one team performs a set of projects. Thus, we found in our research that 65.5% of students performed a single project. In fact, in most cases the work project was chosen by students from a list of projects provided by the teacher staff. Some teams (22.4%) were enrolled in a set of projects of other unit courses in our university. As expectable we did not find any project within the blocked subject-project scope.

The end user of a project is the entity (person, company, organization) that will use the work developed in the project. Thus, we found that 45.1% of the projects had real-world-like as end user. In fact, the goal of most projects was to improve a model, a metric, a technique, a process existing in the industrial world. However, in most cases all research was performed in the educational environment. We classified 5.6% of projects as real word end users. Those projects had some data collect from industrial companies. Finally, we classified as none (40.8%) all projects that we cannot clearly find the end user of the project.

Figure 4.5 presents the results of each part of the planning phase of the application of the framework in all projects. This phase refers to the project planning phase, the time when an instructor designs the work project.

The design of an experiment couples the study scope with analytical methods and indicates the domain samples to be examined. This part is relevant if an instructor is engaged in the research, and he/she wants to experimentally study an aspect of software or quality engineering as part of the project. In our framework we added a new item that was how the teams were formed. We conclude that 56.3% of the teams were self-organized and 42.5% were randomized by an algorithm proposed by the teacher staff. Interestingly, we did not find teams grouped by the teacher staff. We can also observe that the statistical issues (multivariate analysis, statistical models, parametric models, and non-parametric sampling) were not much used by our students in their projects.

Different motivations, objects, purposes, perspectives, scopes, and end users require the examination of different criteria. From the external validity perspective, we collected 32.2%, 17.8% and 3.9% of the work

projects within reliability, transferability, and deployability classification, respectively. The small value of deployability classification resulted from our context, in fact, the projects were performed in educational environment. From the perspective of learning benefits, we obtain 41.4% of projects classified as integrated with course. In a few cases, some students had played a role (4.6%) in other projects from other unit courses of our university. We did not find any project with concerns about ethical issues.

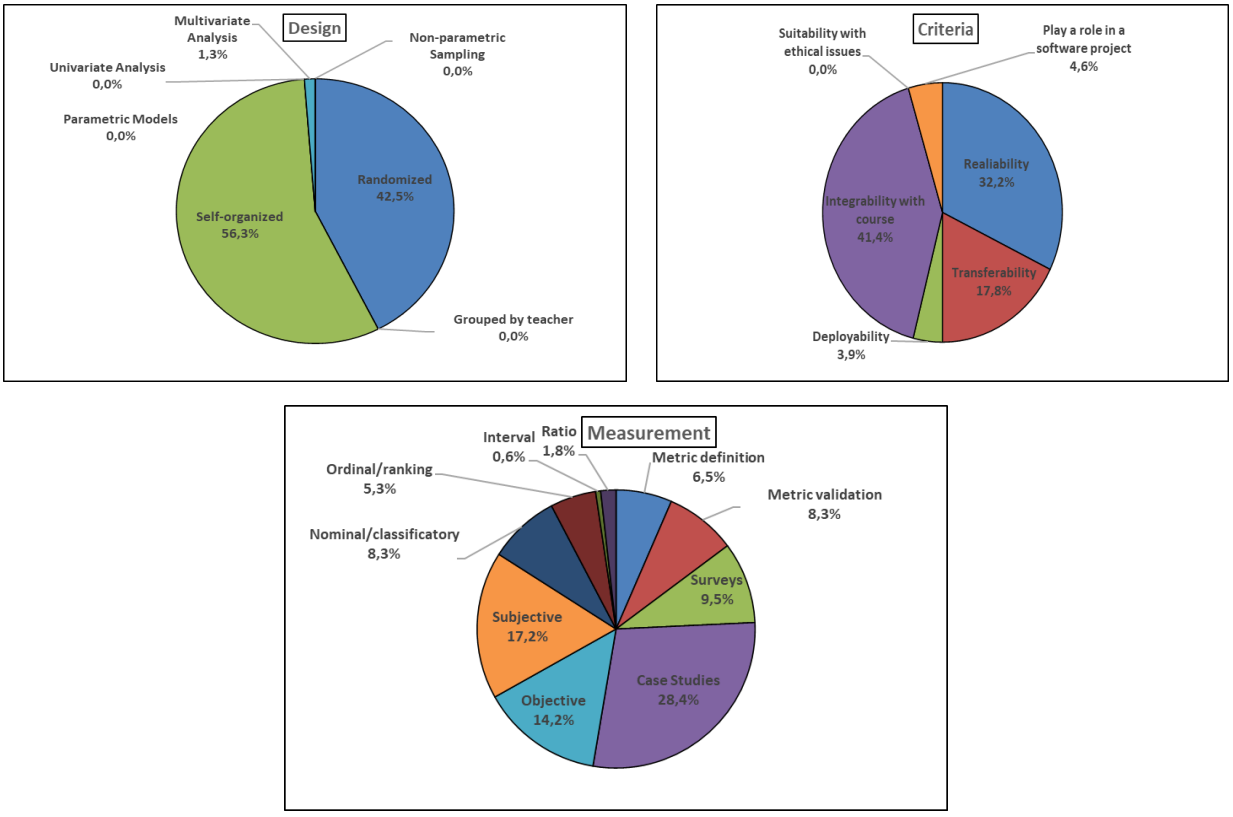


Figure 4.5: Planning phase: design, criteria, and measurement

In the measurement part of the planning phase, we intend to capture some aspects about metrics, data collection, type of research and level of measurement that our students used in their projects. We found that 28.4% of the students used a case study as the research method. In fact, some of PMIS students worked as project facilitators in other unit courses in other computer science courses of our university. Some students used surveys (9.5%) to collect data for their research. Normally, these surveys were questionnaires, but in few cases the students conducted some interviews to domain experts in real companies. In terms of metrics application, we found 6.5% of the work projects with a definition of a metric and 8.3% with a validation of a metric. The data collected by the students in their projects may include both objective and subjective data and different levels of measurement: nominal (or classificatory), ordinal (or ranking), interval, or ratio. We

found 17.2% with subjective data and 14.2% with objective data, which shows a certain balance in the different types of data collected.

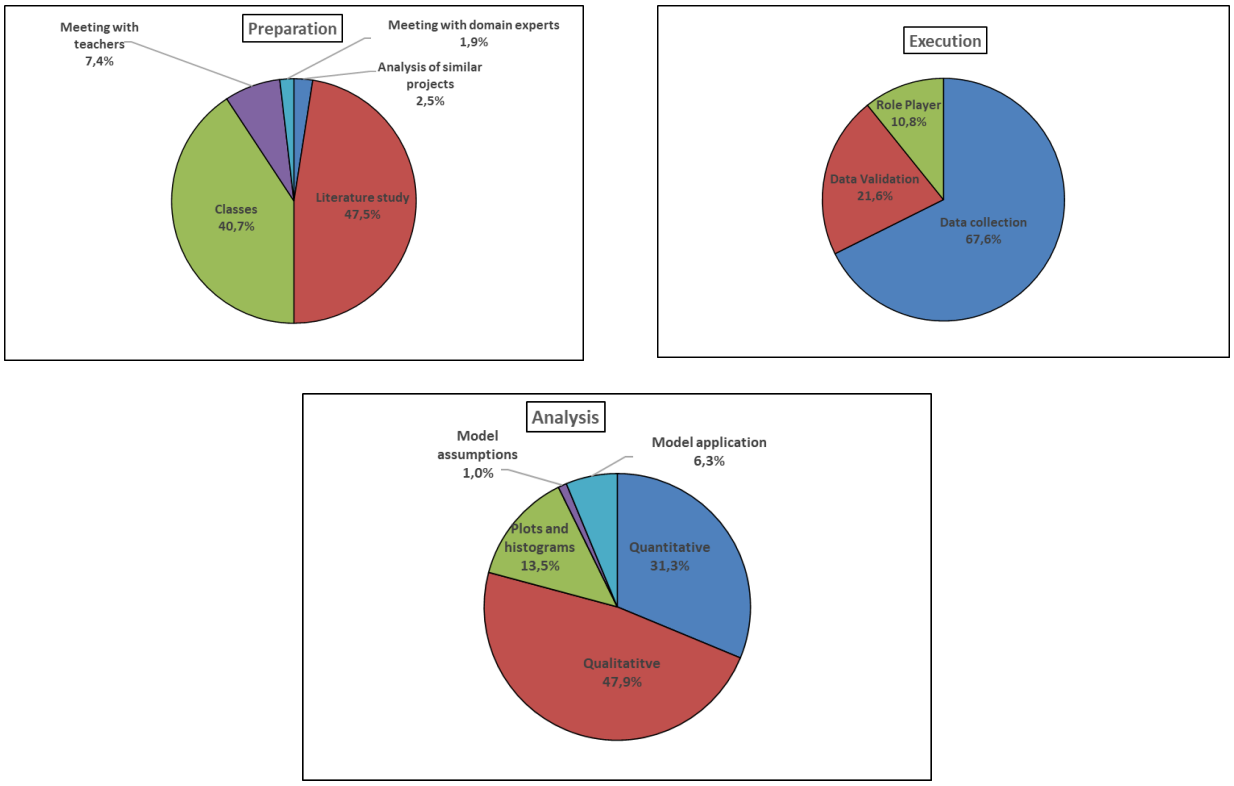


Figure 4.6: Operation phase: preparation, execution, and analysis

Figure 4.6 presents the results of each part of the operation phase of the application of the framework in all projects. This phase is the time when PMIS students accomplish the project.

As expectable, 88.2% of the students get their preparation through literature study (47.5%) and classes (40.7%). The PMIS unit course has contact classes and it has two moments of assessment. In the first moment, in the middle of the academic semester, the students must deliver an intermediate report. In the second moment, at the end of the academic semester, the students must deliver a final document (in report or scientific paper format). The meetings with the teaching staff took place in order to solve some issues related to the projects. We found a relatively low percentage of meetings with domain experts (1.9%). This is explained in large part by the difficulty that students encounter in obtaining some availability from experts. Normally, they are very busy in their jobs. The analysis of similar projects is not relevant in this context, just 2.5% of projects revealed somehow this kind of support.

Execution covers the actual project accomplishment by students as well as data collection and validation. Students collect and validate data during the execution of the study. We classified as data collection

(67.6%) when the main concern of the students was to collect the data for their work project. In the other hand, we classified data validation (21.6%) if the students intended to validate a proposed issue: metric, method, tool, etc. If this distinction was not clear, we would choose both. When a role played was evident we also added this item (10.8%).

The data analysis may include a combination of quantitative and qualitative methods. The preliminary screening of the data, probably using plots and histograms, usually precedes the formal data analysis. The process of analyzing the data requires the investigation of any underlying assumptions applying the statistical models and tests (Basili et al., 1986). We classified quantitative (31.3%) if the students just used quantitative methods to perform the analysis. In the other hand, we classified qualitative (47.9%) if the students just used qualitative methods to perform the analysis. If both methods were applied in their projects, we assigned both. Since students did not use statistical components in the planning design it is uncommon the use of statistical models to perform the analysis (model assumptions, 1.0%, model application, 6.3%). However, 13.5% of the students created some plots and histograms to perform the results analysis.

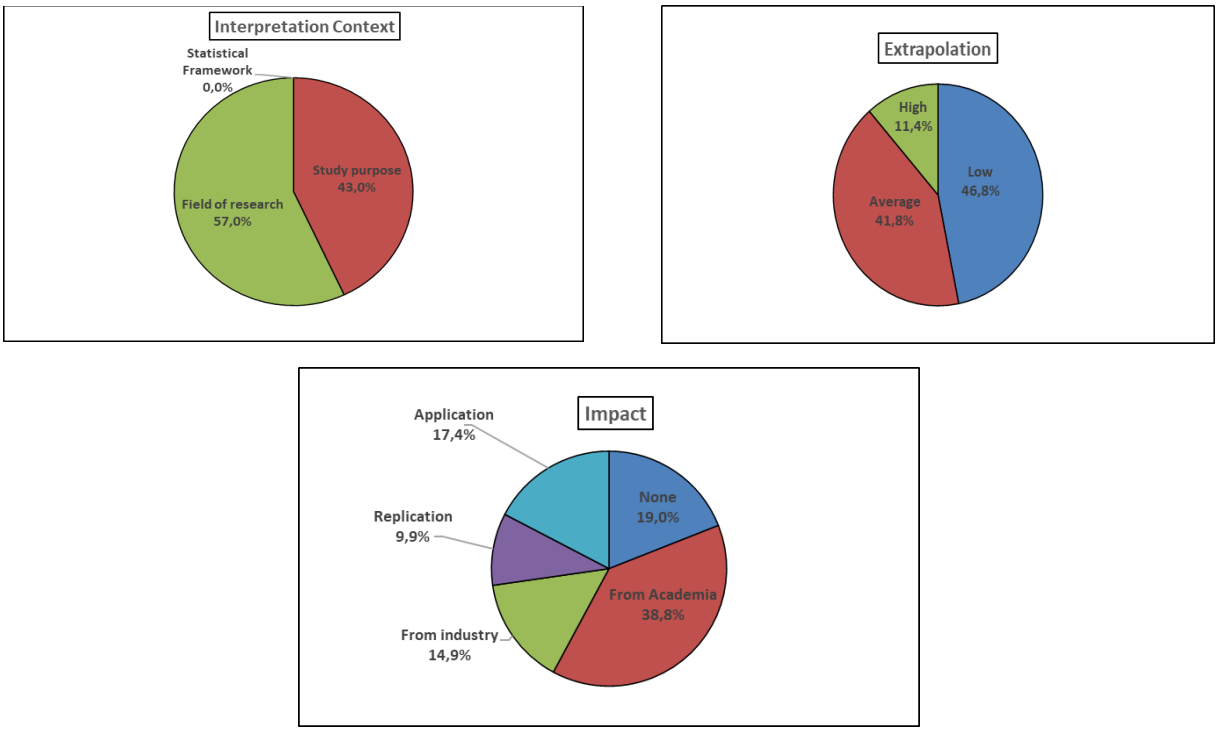


Figure 4.7: Interpretation phase: interpretation context, extrapolation, and impact

Figure 4.7 presents the results of each part of the interpretation phase of the application of the framework in all work projects. At this phase the teacher and/or researcher derives a result from the work project.

The results of the data analysis from a study are interpreted in a broadening series of contexts. These contexts of interpretation are the statistical framework in which the result is derived, the purpose of the particular study, and the knowledge in the field of research (Basili et al., 1986). We classified 57.0% of the projects in the field of research and 43.0% in the study propose. We did not find any project that use a statistical framework in which the result is derived.

The representativeness of the sampling analyzed in a study qualifies the extrapolation of the results to other environments. We defined a scale of low, average, and high to classify the representativeness of the sampling. Thus, we classified 88.6% of the projects as having low and average representativeness of the sampling. We classify just 11.4% of the projects as having a high representativeness of the sampling. These results can be explained by the context where the projects are performed. All projects are performed by students in a unit course of our university.

Several follow-up activities contribute to the impact of a study: presenting/publishing the results for feedback, replicating the experiment, and actually applying the results by modifying methods for software development, maintenance, management, and research (Basili et al., 1986). We defined the visibility of the impact as none, from academia and industry. As expectable, the research performed in the projects were classified mostly with impact visibility from academia (38.8%). In our opinion, this is due to the environment itself and to the lack of experience of the students in the research area. However, we have 14.9% of the projects that can potentially have an impact from an industrial point of view. We classify 19.0% of the projects as none impact. This classification was applied to projects that did not show any type of impact.

We can conclude that our adapted framework is applicable and appropriate to classify project works of the PMIS unit course. In fact, except for statistical issues (multivariate analysis, statistical models, parametric models, and non-parametric sampling) all other components of the framework are suitable.

Given that the framework guides the experimental process and that we obtained results in almost all items, we can conclude that the PMIS unit course is well designed for teaching/learning purposes. It will be interesting to apply our adapted framework to another unit courses in our university and in other universities.

4.5. Experimental Environment

The lack of preparation of SE graduates for a professional career is a common complaint raised by industry practitioners (Karunasekera & Bedse, 2007). One approach to solving, or at least mitigating this problem, is the adoption of the Project Based Learning (PBL) (Barrows & Tamblyn, 1980a) training

methodology. The involvement of students in real industrial projects, incorporated as a part of the formal curriculum, is a well-accepted means for preparing students for their professional careers.

In our experimental environment, we involve students from BSc, MSc, and PhD degrees in Computing from our university that they develop a software project required by a real client. This educational approach allows training students for industry, by involving them with real clients within the development of software projects. The educational approach is mainly based on PBL principles. With our approach, the teaching staff is responsible for creating an environment that enhances communications, team working, management and engineering skills in the students involved.

During Software Engineering (SE) training, it is very difficult to provide industry-standard knowledge and skills, especially non-technical knowledge. These skills can be grouped into three main areas: management, engineering and personal. One challenge facing SE education is that the current lecture-based curriculum hardly engages students. Students often view SE principles as mere academic concepts, which are less interesting and less valuable. The reality is that Computer Science (CS) and Information Systems (IS) graduates often have to develop SE knowledge and skills, especially non-technical knowledge and skills, later on, when they start their careers in industry.

In our approach we involve students from different degrees in Computing from our university. At the BSc level (Bologna 1st cycle) we involve students from Software Process and Methodologies (SPM) and Development of Computer Applications (DCA) courses unit. At the MSc level (Bologna 2nd cycle) we involve students from Analysis and Design of Information Systems (ADIS) and Project Management of Information Systems (PMIS) courses unit. The curriculum integration and the pedagogical cooperation, through an integrated project between the four courses units in analysis, are intended to promote students to work in a software development environment that is similar to an organization environment. Parts of the syllabus of these courses were also framed several times in training given by the teachers in business or industrial contexts, under protocols between the university and relevant organizations.

In this group of courses we must highlight the DCA course unit because of the unifying role it plays when compared to the remaining three. DCA has a learning value of 10 ECTS (European Credit Transfer and Accumulation System) and teachers of subsequent courses “expect” from students an effective ability to develop IT (Information Technology) solutions to problems with medium complexity. This main goal drives the teaching team to adopt a set of procedures and pedagogical practices capable of dealing with the complexity inherent in managing a course unit of this kind.

To perform these software projects, students pursuing the same degree constitute the teams. However, they must work in close collaboration with teams from other degrees. The teaching staff is responsible for creating an environment that enhances communications skills, team working skills, management skills and engineering skills of the students involved. It is a well-accepted fact that a competent software engineer requires a wide variety of skills in areas such as management, engineering, team working and communication (Ali, 2006; Nunan, 1999).

Another challenge is to evaluate teams and individuals who develop unique industry projects (Clark, 2005). In our case, we use “assessment milestones” distributed throughout the semester that allow us to track the students' work progress and thus avoid an end-point evaluation only.

Our approach presents an advance in SE education, in order to overcome the aforementioned challenges. The teams have the opportunity to interact with a real client. They can learn and apply SE principles through a real software project. Thus, they can evolve and improve their technical and non-technical skills. In our setting we promote a win-win approach for all stakeholders: clients, students, teachers, and researchers. Clients will have state of the art projects implemented in their companies. Students can acquire technical and non-technical skills and work closely with real-world problems. Teachers will have the opportunity to teach technical knowledge authentically and realize new problems that companies are facing. Researchers can perform experiments on new and/or existing techniques, tools, and methods. This gives us the opportunity to provide guidelines for SE educators in order to improve their curricula and provide CS students with ready-to-apply SE knowledge skills.

4.5.1. Background

Software Engineering (SE) has brought to the Computer Science(CS) field the confluence of the process and development methodologies with the economic surroundings that are found to be indispensable to the professionals working in the industry with roles and responsibilities that go beyond the mere computer programming (Engle, 1989). In SE we find a clear concern with what is beyond purely technical issues, which have always been the ones that truly the CS devoted full attention to. In this sense, it is important for a software engineer to be educated in communication and management skills. These are skills which are vital to the software engineer’s success and they cannot be left to be learned by "osmosis" (Engle, 1989).

While SE (as a discipline) is dedicated to study the software process development, IS (as a discipline) analyzes the impact of software-based systems on individuals, organization, and society. However, in the

scientific context there is a great divide between methods and research questions (Finkelstein, 2011; Gregg, Kulkarni, & Vinzé, 2001). The cross-fertilization between the two disciplines has allowed the cooperative and coordinated development of the engineering and requirement management, modeling and systems architecture, process development and project management approaches (Avison & Wilson, 2001; Birk et al., 2003; McBride, 2003). This is why, in the context of project technology-based solutions in problems of organizational nature, the two disciplines (and corresponding professional performances) merge into a blend of common methodologies, techniques and tools (Fung, Tam, Ip, & Lau, 2002; Hellens, 1997; Jayaratna & Sommerville, 1998) demonstrating the existence of an SE/IS convergence. The understanding of the intersection between the two disciplines allows students to develop projects of better quality, since the whole project is grounded in the knowledge of both disciplines.

When we are to deliver knowledge and skills to IT students, we should be able to teach engineering and software management. The success of a software engineer is related to the ability of engineering and software management methodologies to adapt to the huge demands that professionals are subject to nowadays, which include dealing with all procedural issues of software production, along with technological competence and sensitivity to the needs and expectations of users (Platt, 2011).

The rationalization of all decisions relating to issues not explicitly technological is fundamental for a correct (effective and efficient) operation of a software development team (Dutoit, McCall, Mistrík, & Paech, 2006). Into this set of concerns and attitudes also fit the aforementioned software economics issues (Tockey, 1997). When properly reconciled with the technological dimension, *Software Engineering Management* has adopted the designation of *Software Engineering and Management* (Shere, 1988). This designation, which reinforces the existence of a management dimension within the SE (as in any other Engineering specialty), represents a break with the CS discipline and an assumption of the socio-technical nature of the SE, and its inevitable convergence with the IS discipline (Kurbel, 2008).

In our approach SPM and DCA courses unit intend to provide students an environment of software development projects similar to an environment in organization context. They introduce engineering and management software techniques to collect and specify the requirements of the software development projects. They also deal with the software lifecycle management issues. These two courses unit are mainly responsible for software development methodologies and software project planning teaching.

The ADIS and PMIS courses unit intend to instill in students an engineering approach to information systems development. This attitude derives from the SE/IS convergence mentioned above and is materialized

in the business solutions development study (enterprise business applications). Thus, the engineering and requirements management techniques and modeling and systems architecture are complemented, and the maturity and software processes improvement issues are systematized, as well as the processes management and project development.

The Department of Information Systems (DSI) of the School of Engineering (EEUM) of the University of Minho, where this research is performed, offers an educational portfolio in the Information Systems and Technologies (IST) area that covers all levels of higher education. When this research was performed, before school year 2021/2022, the Master' s degree had a integrate format. The study plan included five years and conferring a Master' s degree. Currently, due to legislative decisions, the study plan has three years for Bachelor' s degree and, separately, more two years for Master' s degree. Our experimental environment was build based in previous study plan of the courses. Thus, all the research presented here should be seen in light of this assumption.

The initial training of IST professionals is achieved through the integrated Master in Engineering and Management of Information Systems (MIEGSI). This is the main master course that has SPM, DCA, ADIS and PMIS courses unit in its curriculum. The challenges of keeping up with the constant evolution of IST professional training and the demands of adjusting higher education programs to the new principles and rules for higher education have been addressed through several changes in the program structure of this Master's degree. The MIEGSI results from the Information Systems and Technologies BSc degree and the Engineering and Management of Information Systems MSc degree combination. These courses were the result of the adaptation (appropriateness) of the Information Management course to the higher education model established in 2006 (the Bologna Process).

Although the current *Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems* is IS 2010 (ACM-Curricula-IS, 2010), MIEGSI was originally structured according to the previous version of the standard, the IS 2002.

Based on knowledge of the areas and their comparison we can infer that the MIEGSI is a course in the IS area, but with less depth in topics of Information Technology curriculum area than suggested by IS 2002. The SPM and DCA curricular units deal with some topics in great depth whose inclusion in ISBOK (*Information Systems Body of Knowledge*) has its origin in the SWEBOK (*Software Engineering Body of Knowledge*) (Abran et al., 2004). These courses unit belong to a curriculum plan of IS and not CS or SE. The main difference is on the software organizational nature focus (business or enterprise software applications), in conjunction with

the strategy and culture of the enterprise, the existence of a great diversity of requirements sources (from traditional business stakeholders to contexts in which the architecture of information systems already appears rudimentarily conceived) and the coexistence with professionals with very different perspectives on ITs (functional consultants, information systems architect, manager of technology infrastructure, ...), all of which require a large capacity of flexibility in the implementation and management of development processes.

The publication of IS 2010 for updating the IS 2002 model resulted from the natural changes of the technological and industrial practices that the domain has been permanently subject to from its beginnings. The IS 2010 model introduced some adjustments in the courses unit recommended as mandatory (core courses) and significantly increased the range of elective courses. The SPM course unit fulfill the stipulated of *IS 2010.4 IS Project Management* course. In the case of DCA course unit, its syllabus covers what is stipulated in *IS 2010.6 Systems Analysis and Design* and *Application Development* courses. This last course unit, despite being classified as elective type, appears to be referenced as mandatory for all profiles of professional training (career track) considered by IS 2010. Notwithstanding the structural and programmatic adjustments, the author of this thesis considers that the syllabus of MIEGSI is aligned with the recommendations of IS 2010.

In the case of ADIS and the PMIS courses unit from MIEGSI integrated master, the study of alignment with the curriculum frameworks requires analyzing the MSIS 2006 (ACM-Curricula-MSIS, 2006). The framework belongs to the *Computing Curricula* program, which in the AIS (Association for Information Systems) leadership focuses on post-graduate education in IS. This framework presents a strict balance between curricular areas of *IT Technology* and *IS Management* treated in mandatory courses which recommends.

The ADIS course unit is aligned with the *MSIS 2006.2 Analysis Modeling and Design* course. Following the recommendations of MSIS 2006, the ADIS course unit complement the training received, particularly in DCA course unit, on modeling and development cycles issues of the technology-based solutions. The PMIS course unit is aligned with the *MSIS 2006.5 Project and Change Management* course. Following the recommendations of MSIS 2006, the PMIS course unit complement the training received, particularly in SPM course unit, on management of the development process issues, introducing the whole socio-technical dimension of the IS and considering the projects as drivers of technological and organizational change.

The curriculum frameworks under the *Computing Curricula* program were developed with the explicit aim of being adopted in university education in the USA (United States of America) and Canada. However,

the use of its recommendations outside this geographical context has proven to be a useful practice, with many more advantages than disadvantages. Given that the IT area (in its various forms) is still in its infancy (and therefore remains subject to rapid evolution, which makes it extremely difficult to keep up with the various curricular offerings) and that the abovementioned two countries are at the forefront of technological development, it is a recommended practice to regularly peruse what is being said about the training of their professionals.

Despite the enormous worldwide spread of the frameworks produced under the *Computing Curricula* program, there are countries that choose to follow different paths to develop their own frameworks as a result of a concerted effort within their universities. One example is Australia, where, in some universities, an approach has been adapted to training in IS with a specific curriculum (Tatnall & Burgess, 2009). The Australian approach has evolved from a perspective called "Information Management" and for decades was designated "Business Computing" (Retzer, Fisher, & Lamp, 2003).

Another example is Germany, which has developed a body of knowledge suitable for framing how the courses in IS should be taught (Kurbel, Krybus, & Nowakowski, 2021). In the German language, this area has historically been designated "Wirtschaftsinformatik" which over the years has been translated into English as "Business Informatics" or "Business Computer Science". Recently the accepted term for the IS area has been “Business & Information Systems Engineering".

The variability of possible approaches in teaching, not only in the IS sub-area, but, generally, in the IT (Computing) area suggests, sometimes, the use of accreditation curriculum and professional certification as a way to introduce some order in the training of professionals and to recognize, unequivocally, the specific skills that a professional must have in a set of sub-domains required by the industry.

4.5.2. Our Project Based Learning Approach

This section describes how the integration of the four courses units mentioned in the previous section has been managed, as well the solutions tried to promote educational cooperation in the teaching practices context based in projects. This integration has been carried out over the last twelve years. More precisely, it began in the academic year 2009/2010. This integration has been managed to allow minor changes over the years and to improve some relevant aspects. It is in this controlled context that we developed our PBL approach.

The *Bologna Declaration* (European Commission, 2021) was a political commitment to achieve in the short term, a clear set of objectives recognized as fundamental to the construction of the "European Higher Education Area" and to promote "European System of Higher Education" throughout the planet.

Integrated in the changes recommended by the *Bologna Process*, the ECTS is part of a set of procedures that support the new paradigm of organizing student-centered learning (and training objectives) and move from a traditional system curriculum based on the "juxtaposition" of knowledge to a system focused on developing broad curricular areas, defined in terms of training objectives to pursue. This argument reinforces the relevance of integrating a group of courses units around global goals so that we can instill in students several skills in a way not comparable with skills obtained with the same courses in isolation mode. In ECTS, the work done by students in the subject area is expressed by a numerical value that takes into account the hours of student work, in their overall activities, including contact hours and hours spent on internships, projects, works in ground study and evaluation. The entire work done by the student in each course unit is expressed in ECTS credits and each ECTS credit corresponds to a total of 28 hours per semester.

The ECTS system is suited to changes in training, mainly in the development and adoption of: (1) new learning methodologies (more active and participatory), (2) horizontal capabilities and skills (learning to think, learning to learn, learning to teach), (3) specific skills of the profession, (4) general skills (communication capabilities, integration team, leadership, innovation, and adaptation to change). All these dimensions were considered and are thought to be adequately incorporated in the joint project that integrates the four courses of our approach.

In many classrooms, learning is a passive activity. Students take notes during lectures and repeat the same information in the exams. When students read a chapter indicated by the teacher and they answer questions about that, the answers can be found in the chapter and are already known. Even in more experimental areas, the teachers rarely allow to students discover principles themselves; instead, the teachers present laws and techniques, and then they build exercises where students simply practice what they have been taught. This teaching is essentially based on the transmission of knowledge. The *Bologna Process* suggests switching to a school based on the students' work and the effective acquisition of skills, however, it should not put into question a proper proportion of more traditional activities also dedicated to the “simple” transmission of knowledge.

Although the *Bologna Process*, apparently "censors" the "teaching based on the transmission of knowledge", what is called contact time is no more than a series of moments in which teachers and students synchronize spatially to exchange knowledge with the perspective of developing, in students, certain skills. The *Bologna Process* should not avoid the transmission of knowledge between teachers and students, but rather aim to go further, catalyzing the emergence of new skills in students during the transfer of knowledge process. In this context, the transmission of knowledge is desirable and beneficial, so the pedagogical action in circumstances where every student has limited opportunities to interact with teachers does not foresee success. It is in the laboratory classes that transcendence of "skills transfer" can occur if each student is given the opportunity to receive knowledge. Consequently, the number of students spatio-temporally synchronized with the teachers in laboratory activities must be carefully determined, taking into account the expected level of depth of the "quasi-tangible" learning outcomes.

It was based on this change promoted by the *Bologna Process* that in the academic year 2006/07, it was decided to adopt teaching and learning practices based on the PBL (Barrows & Tamblyn, 1980b) principles in the DCA course unit (since it is DCA course which catalyzes the main project work that integrates other courses). The teaching and learning context that PBL facilitates has been shown to be appropriate to the group of courses that integrate our environment (approach), where we want students to develop real skills in the production of technological artifacts imbued with a spirit of great rigor and methodological and procedural awareness and not simply to reproduce texts and definitions held in any book or manual timelessly accepted on any shelf or in any drawer. What drives the student to want to persist in school learning has to do mainly with the way we create and organize educational environments and all activities that we develop there.

Solving a problem according to the PBL approach requires the participation of students. The teacher helps and advises but does not drive. Learning becomes an act of discovery as students examine the problem by investigating its base, analyze possible solutions, develop proposals, and produce a final result. This active learning is not only more interesting and committed to the students but also develops a greater understanding of the syllabus since the students themselves seek information and then use it in an active way with the skills they already hold to complete the project. This way of organizing the teaching and learning activities for skills development promoted by the four courses is considered appropriate. Svinicki and McKeachie argue that "problem-based education is based on the assumptions that human being evolved as individuals who are motivated to solve problems, and that problem solvers will seek and learn whatever knowledge is needed for

successful problem solving. Thus, if an appropriate realistic problem is present before study, students will identify needed information and be motivated to learn it. However, as in introducing any other method, you need to explain to students your purposes" (Svinicki & McKeachie, 2014).

PBL requires the realization by students that learning lies in the prosecution of skills, not with the teacher to tell them that they are right but based on experimentation with the artifacts and documents that they produce. In cooperative learning promoted by PBL, students learn from each other, and they work together to develop the project. This aspect is extremely important in the context of our group of courses, in which there is the involvement of students with different academic degrees and maturity. In PBL, students grow more thoughtful and are harder working than in exercises that require rote memorization. In our approach, the emulation of a real project (with a real client) forces the students to learn from a variety of different sources and to make decisions based on their own research. This process allows students to achieve more advanced levels of cognitive skills, research skills and problem-solving skills.

The PBL's real academic goal is not to develop a final response to the project. The students do not find just one true answer to the problem that, instantly, they agree can be the "correct" solution. Instead, the real learning occurs through the process of solving a problem: thinking through various steps, investigating the subjects, and developing one solution. With the continuing explosion of knowledge and the pace of technological change, the universities cannot continue to provide all the information to the students that they need for their lives. Increasingly, the most important skill that the university can teach students is to learn by themselves. Within the group of courses units integrated in our approach, this issue arises repeatedly when we want students to understand by themselves that throughout their careers will have to learn how to use and design new development processes, notation models, standards, paradigms, frameworks, management, process improvement and project standards.

The recognition of the effectiveness of the PBL approach in engineering teaching has resulted in several initiatives, including scientific projects of a pedagogical nature, in order to produce guidelines for the teaching and learning activities organization under PBL principles. In some forums, the use of PBL in engineering teaching has adopted the *Project-Led Engineering Education* (PLEE) designation (Powell, Powell, & Weenk, 2003).

The project that is presented every academic year to the SPM and DCA students requires energetic learners. Nobody will give them all the necessary information, nor will the answers be found in the books. To solve this problem requires that students "discover complaints", "investigate the reasons" and develop the

best way to resolve the situation. Thus, our approach, which is based on this project, presents some crucial features for creating a context of enormous catalyzing of the phenomena of teaching and learning, namely:

1. *Real client*. The existence of a real client, who interacts with, gathers and receives students in his organization, promotes in students the ability to feel, in practice, the difficulty of organizational software development with incomplete information and systematic doubts from the client, but with explicit business support needs (Trendowicz, Heidrich, & Shintani, 2011). It also allows students to understand how the client thinks and evaluates the effort put into the development of the solution (Burge & Troy, 2006).
2. *Project proposal versus project*. In the first semester, in the SPM course unit, the project proposal phase is created, in which feasibility studies are designed and carried out, with some initial incursion into requirement elicitation. This phase ends with a project proposal elaboration, including time and cost estimates and an initial definition of generic features of the solution. This separation between the project proposal phase (emulated in SPM course unit) and the project phase (emulated in DCA course unit) allows the students to understand the different requirements and planning approaches that they are required to adopt in each of those circumstances and perceive the difference between a more commercial nature (required in the project proposal phase) and a more technically oriented approach (essential in the implementation phase of the project) (Brazier, Garcia, & Vaca, 2007).
3. *Large Teams*. The high dimension teams allow the recreation of the complexity of an industrial context in terms of the multiplicity of tasks to be performed (Blake, 2005) and the enormous need for interaction between the different roles, promoting the development of skills of an inter-personal nature (Slaten, Droujkova, Berenson, Williams, & Layman, 2005). The development of the *soft skills* is one of the great gains acquired by students as a result of their involvement in the project.
4. *Rigor in software process development*. The adoption of a configurable software process development permits the instillation of awareness and rigor in how students decide the management and implementation of the project (Suri & Sebern, 2004). During several academic years, this procedural rigor resulted from the RUP framework adoption with the reduced model of roles (Borges, Monteiro, & Machado, 2011), the obligation to follow the CMMI model practices and to adopt the PMBOK (*Project Management Body of Knowledge*) (PMI, 2021) and SWEBOK (Abran et al., 2004) bodies of knowledge. With these standards, the relationship between the project management and the software process development adopted becomes explicit, allowing

sensitization of the students to the dire need to own specific methodological skills in the area in order to be able to manage projects in the field of technology and information systems. Additionally, it allows the creation of a true perception of the relationship between the structures of the solution implementation and the requirements that gave rise to it (Burgstaller & Egyed, 2010), as well as between product quality and the rigor of process practices adopted by the team.

5. *Complexity of the solution*. The project can create conditions in the educational context where students are engaged in designing and building medium-size software solutions based on requirements from the "real world". This makes for the experience of conceiving an architecture which is methodologically plausible (Naveda, 1999), while developing the students' sensitivity to the effectively effort needed to conceptualize artifacts with similar complexity to those developed in industrial contexts (Wohlin, 1997).
6. *Others*. In every academic year a mechanism or new feature in project work is introduced in the second semester that cumulatively add some novelty to the operational mode of the previous year. Over the years, the following features or mechanisms were introduced (not in chronological order): alignment with SWEBOK, alignment with PMBOK, RUP model, internal evaluation and promotion of human resources, CMMI level 2 practices, CMMI level 3 practices, real client, final presentation with a commercial focus, formalization of the product delivery, outsourcing, technological interoperability between partial solutions, consulting of outside services, use of patterns, formal documentation with RUP templates. Recently, in the last academic years, the following mechanisms or features were adopted, namely: agile practices, reinforcement and hiring human resources, competition between enterprises and corporate bankruptcy.

Over the years it has only been possible to cumulatively manage all these features and mechanisms because the teaching staff is stable, cohesive, and aligned in the way it organizes and engages with students who annually attend the four courses (about a hundred students in SPM and DCA and from about three-and four dozen students in ADIS and PMIS). The curriculum integration and the pedagogical cooperation among the four courses units have also been decisive in making possible the management of all these educational processes that converge in the laboratory classes in the DCA course unit, since it consists of the place of convergence by the students from both levels of education that functionally associate in the project (see Figure 4.8): (1) the ADIS students provide external services (regarding to the functional consultant or senior analyst role) to the SPM students and PMIS students provide external services (regarding to the project

facilitator, senior project manager or process engineer role) to the DCA students; (2) the SPM students emulate phenomena studied by ADIS students and DCA students emulate phenomena studied by PMIS students. Often ADIS and PMIS students choose for their dissertations some of the themes that they dealt when they worked with SPM and DCA students.

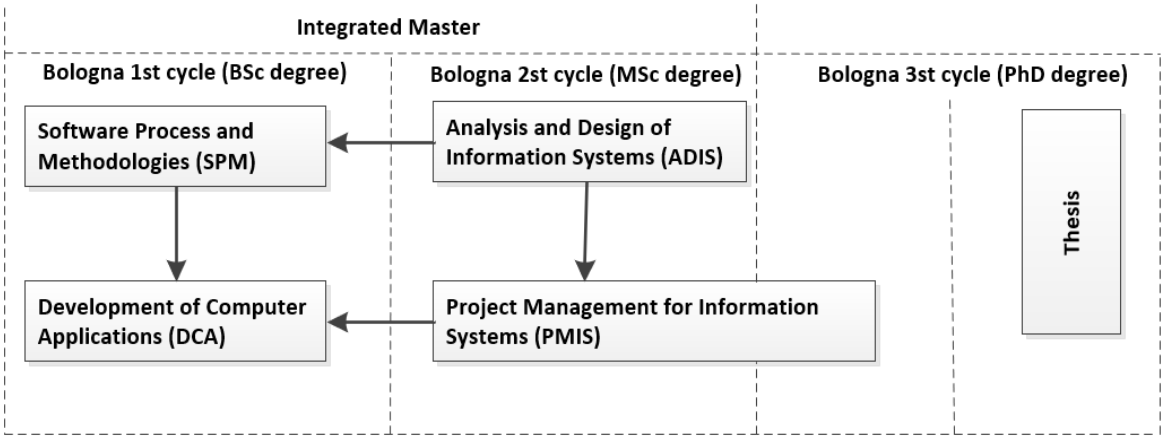


Figure 4.8: Integration among courses

In the last academic years, some of the PhD students in the SEMAG (Software-based Information Systems Engineering and Management Group) (SEMAG, 2001) research group also began to engage with the students of the four courses units, in order to carry out scientific studies supported in educational context experiments and this work has given rise to international publications (some involving ADIS and PMIS students) (Alves, Machado, & Ribeiro, 2012; Alves, Oliveira, Ribeiro, & Machado, 2014; Alves, Ribeiro, & Machado, 2014, 2016, 2018; Alves et al., 2020; Alves, Sousa, Ribeiro, & Machado, 2013; Alves, Souza, Ribeiro, & Machado, 2021; Monteiro, Borges, Machado, & Ribeiro, 2012; Monteiro et al., 2013). Thus, this group of four courses units is simultaneously a space of pedagogical and scientific innovation from which all stakeholders benefit (Siqueira, Barbaran, & Becerra, 2008). It is our perception that almost all stakeholders in this huge process (students, teaching staff and researchers) feel "pleasure" with the involvement in the project (Glass, 2007a; Glass, 2007b), despite the great dedication and tremendous effort that is demanded from all involved.

Although not addressed in this thesis, the pass rate of these curricular units is extremely high, in large part due to our experimental environment (PBL approach).

4.5.3. Results Obtained from our PBL Approach

The real-world project/study approach to teaching software engineering has been successful thus far. It has helped to motivate the teams and to encourage development of higher quality products by the teams. The teams took seriously the importance of the problems that they were helping to solve. The approach teaches inexperienced graduate students many principles of software engineering and software verification and validation that they could apply to newly gained jobs and/or subsequent courses (Hayes, 2002).

Our PBL approach began to be effectively implemented in the academic year 2009/2010, although some well-controlled trials had been tested in previous years. The real client collaboration in the software project just started in that academic year. This client was located in the region of our university. The real client activity area was quite diverse. In our previous software projects, we had collaborations with a factory enterprise, a nonprofit institution of social solidarity, a professional handball team and a professional football team.

In the book chapter "Project-Based Learning: An Environment to Prepare IT Students for an Industry Career", a detailed study of the work projects produced by SPM, DCA, ADIS and PMIS students is presented (Alves, Ribeiro, et al., 2014; Alves et al., 2018). This study covers four academic years, namely, 2009/2010, 2010/2011, 2011/2012 and 2012/2013. In these four academic years we have had, approximately, one thousand students involved in the four educational syllabuses. In each academic year, we had approximately 150 students in SPM and DCA courses and 100 students in ADIS and PMIS courses.

It is not easy to teach the syllabus of the four courses because of their considerable size and a lack of conditions revealed by the students for realization of topics with a more abstract nature. In fact, this group of courses has a considerably high level of requirement, so the methodologies or teaching strategies to be adopted should be in accordance with the level of depth of the topics.

The students should be able to use the study objects to perform the tasks. In the case of the DCA course unit, this level of learning depth requires a deep involvement by the students in practical classes (in the form of exercises proposed and solved in class), complemented by the development of a software project.

This work begins in the first semester under the SPM course unit, in groups of about five students. These groups of students emulate the project proposal context in which they perform an initial analysis of requirements and time and cost planning of the project proposal solution for a real client with whom the teaching staff establishes an annual academic collaboration partnership (Ali, 2006; Kornecki, Khajenoori, Gluch, & Kameli, 2003). In the DCA course unit, in the second semester, the work project is performed by

groups of about 15 students. These groups of students emulate the project implementation context of a proposed solution to the same real client, which they perform a comprehensive analysis and design of the problem, construction, testing and delivery of a software solution coded in an object-oriented language (*Java* or *C#*) with relational databases (*SQL Server* or *MySQL*) and with interfaces for the Web. In both courses, student groups follow the RUP model (Bergandy, 2008) and they use the UML notation extensively. The teams use the laboratory classes to meet in the presence of teachers. In these weekly meetings, the teachers monitor the progress of the project work.

In the case of ADIS and PMIS courses, the learning outcomes with higher depth level are achieved through the involvement of students with groups of SPM and DCA courses, respectively, promoting play roles with responsibility and a high level of maturity, such as the functional consultant, senior analyst, project facilitator, senior project manager or process engineer. These two courses have different learning outcomes with deep levels corresponding to some of the emerging topics addressed and discussed in lectures, which prepare the students with a good understanding of a set of problems that currently emerge in the area.

The learning outcomes with a more advanced level are achieved through the use of a scientific literature searches in order to study in detail some complex themes that arise in the engagement context by the ADIS and PMIS students when they participate in the project work performed by SPM and DCA students. The two semiannual project works instill a hands-on training dynamic that is essential for the two pairs of courses operation as a whole (Broman, 2010; Port & Boehm, 2001): SPM and ADIS courses in the first semester and DCA and PMIS courses in the second semester.

The university regulation concerning the evaluation of courses was changed in 2004. From that year it became possible to evaluate a student that "only" executes a project and thus dispense with the realization of an exam. Thus, the evaluation system adopted allowed the organization of all activities of the four courses around an annual educational project. This project is the integrator of all the topics listed in the syllabus and it follows a student time management approach that is able to facilitate the effective development of skills necessary for evidencing learning outcomes stipulated (Stiller & LeBlanc, 2002). Since the academic year 2006/07 this group of courses has adopted this evaluation system, which provides a much more effective management of students activities inside and outside of the university physical spaces and in a horizon that it extends over the two semesters of the academic year, according to the PBL approach.

Since DCA is the core course in our approach then we will describe in some detail the evaluation system of this curricular unit. This evaluation scheme encourages the incremental demonstration of the partial learning outcomes over the semester.

Between the two semesters in which annual project is developed, the second semester is the one that concentrates the largest effort and diversity of activities and in this sense, DCA is a curricular unit of 10 ECTS. The student evaluation in DCA is carried out based on the activities undertaken by various teams within the semiannual project work in five *assessment milestones*. Each one of the five *assessment milestones* is associated with partial evaluation, four are team evaluations, and one is an individual evaluation. However, it is systematic that several students in a team obtain different classifications in the four team *assessment milestones*, since the individual score calculation results from a plurality of weighting factors from three sources of information (Clark, 2005), such as the teachers traditional evaluation, client evaluation and peer evaluation (other students of the team).

In each academic year, the reports developed in ADIS and PMIS are edited in two separated documents which are assigned with ISSN (International Standard Serial Number). These documents contain all the student project works of both courses. These project works can be a report or a scientific paper that describes all the work performed by the students. These documents serve as consultation reference for future students. These students can find the difficulties, limitations, the positive and negative aspects, and lessons learned and experienced by colleagues of the previous years. Table 4.6 was created based on a detailed analysis of these documents. The table presents a mapping between the SWEBOK Knowledge Areas (KA), and the work projects developed by the ADIS and PMIS students in the academic years 2010/2011 and 2011/2012.

As we can see in Table 4.6, in the 2010/2011 and 2011/2012 academic years, the 140 students enrolled in the ADIS and PMIS courses developed 130 work projects in different SWEBOK KAs. Although the majority of work projects are developed individually, some of them involve teams of 2 or 3 students. We can also observe that there was a large increase in the work projects in the 2011/2012 school year. In fact, in the school year 2010/2011, 34 work projects were analyzed, while in the school year 2011/2012, 96 work projects were analyzed.

Table 4.6: ADIS and PMIS Work Projects by SWEBOK Knowledge Area

SWEBOK Knowledge Area	**Academic Year**				**Sum**
	2010/2011		**2011/2012**		
	ADIS	PMIS	ADIS	PMIS	
Software Requirements	2	2	8	6	18
Software Design	3	0	7	1	11
Software Construction	0	0	1	2	3
Software Testing	0	0	0	0	0
Software Maintenance	0	0	1	1	2
Software Configuration Management	0	0	0	0	0
Software Engineering Management	11	7	11	26	55
Software Engineering Process	6	0	13	8	27
Software Engineering Tools and Methods	0	1	0	0	1
Software Quality	2	0	3	8	13
Sum	24	10	44	52	130

Figure 4.9 shows that most of the project works are developed in the *Software Engineering Management* area, representing 42.3% of the total. The high percentage of the project works in this KA is justified by the involvement of ADIS and PMIS MSc students as a consultant/facilitator role of the DCA development teams (BSc students).

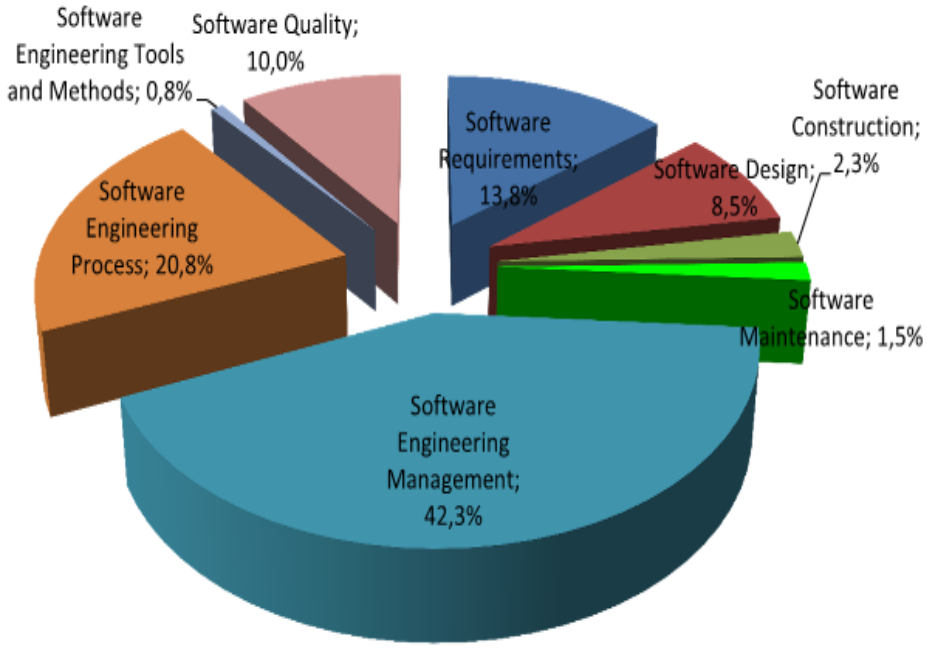


Figure 4.9: ADIS and PMIS Work Projects by SWEBOK KA

Another conclusion obtained by the Table 4.6 analysis is the total absence of project works in the *Software Testing* and *Software and Configuration Management* areas. Normally the tests are performed by the DCA students, those who develop the software applications, and until now, the MSc students have not expressed interest in this KA. Regarding *Software and Configuration Management,* the absence is justified by the fact that we are working in an educational context and some activities of that KA are not performed, such as, software configuration identification, software configuration control, software configuration status accounting, software configuration auditing, and software release management and delivery. It is also worth noting the low number of project works in *Software Construction* and *Software Engineering Tools and Methods* KAs.

Table 4.7 and Table 4.8 show the topics and subtopics of the SWEBOK KA covered by ADIS and PMIS work projects respectively. The numbers preceding the names of the topics and subtopics are those in SWEBOK document.

Table 4.7: Topic and subtopic of the SWEBOK KA covered by ADIS Work Projects

SWEBOK KA	**ADIS Course**	
	SWEBOK Topic	**SWEBOK** Subtopic
Software Requirements	1. Software Requirements Fundamentals	1.1. Definition of a Software Requirement 1.2. Product and Process Requirements 1.3. Functional and Non-functional Requirements
	2. Requirements Process	2.3. Process Support and Management
	4. Requirements Analysis	4.1. Requirements Classification 4.2. Conceptual Modeling
	5. Requirements Specification	5.3. Software Requirements Specification
	7. Practical Considerations	7.2. Change Management 7.4. Requirements Tracing
Software Design	1. Software Design Fundamentals	1.3. Software Design Process
	3. Software Structure and Architecture	3.1. Architectural Structures and Viewpoints 3.2. Design Patterns (microarchitectural patterns)
Software Construction	3. Practical considerations	3.2. Construction Languages
Software Maintenance	4. Techniques for Maintenance	4.3. Reverse engineering
Software Engineering Management	2. Software Project Planning	2.1. Process planning 2.4. Resource allocation 2.5. Risk management
	3. Software Project Enactment	3.2. Supplier contract management 3.4. Monitor process 3.5. Control process
Software Engineering Process	1. Process Implementation and Change	1.1. Process infrastructure 1.2. Software process management cycle
	2. Process Definition	2.1. Software life cycle models 2.2. Software life cycle processes 2.3. Notations for Process Definitions
	4. Process and Product Measurement	4.1. Process measurement 4.2. Software product measurement
Software Quality	1. Software Quality Fundamentals	1.3. Models and quality characteristics

Table 4.8: Topic and subtopic of the SWEBOK KA covered by PMIS Work Projects

SWEBOK KA	**PMIS Course**	
	SWEEBOK Topic	**SWEEBOK** Subtopic
Software Requirements	1. Software Requirements Fundamentals	1.3. Functional and Non-functional Requirements
	2. Requirements Process	2.1. Process Models
		2.3. Process Support and Management
	3. Requirements Elicitation	3.1. Requirements Sources 3.2. Elicitation Techniques
	4. Requirements Analysis	4.4. Requirements Negotiation
	5. Requirements Specification	5.3. Software Requirements Specification
Software Design	5. Software Design Notations	5.1. Structural Descriptions (static view) 5.2. Behavioral Descriptions (dynamic view)
Software Construction	3. Practical considerations	3.1. Construction Design 3.2. Construction Languages 3.3. Coding 3.4 Construction Testing
Software Maintenance	4. Techniques for Maintenance	4.3. Reverse engineering
Software Engineering Management	2. Software Project Planning	2.4. Resource allocation 2.5. Risk management 2.6. Quality management 2.7. Plan management
	3. Software Project Enactment	3.1. Implementation of plans 3.2. Supplier contract management 3.4. Monitor process 3.5. Control process 3.6. Reporting (RUP Roles)
Software Engineering Process	1. Process Implementation and Change	1.1. Process infrastructure
	2. Process Definition	2.2. Software life cycle processes
	4. Process and Product Measurement	4.1. Process measurement 4.2. Software product measurement
Software Engineering Tools	1. Software Engineering Tools	1.7. Software Engineering Management Tools
Software Quality	1. Software Quality Fundamentals	1.3. Models and quality characteristics
	2. Software Quality Management Processes	2.1. Software Quality Assurance 2.2. Verification & Validation
	3. Practical Considerations	3.4. Software Quality Measurement

From Table 4.7 and Table 4.8, we can infer that seven and eight of the ten SWEBOK KAs are covered by the ADIS and PMIS work projects respectively. We just can find one PMIS work project in the *Software Engineering Tools and Methods* area. Specifically, this project work involved the *Microsoft Project Server 2007* key features study, through two visions: one as a project manager and other as system administration. Despite the topics and subtopics diversity covered by the work projects, we cannot infer that there is a great variability of subjects between the two courses.

4.6. Conclusion

In this chapter we present an adapted framework to classify empirical studies performed in an educational context. The goals of the framework were described. The scheme (architecture) of the framework and its main components have been described in detail and its relevance was discussed.

The main goal of our framework is to provide a classification scheme for understanding and evaluating empirical studies in an educational context. We used the framework to classify the projects performed by our students registered in the PMIS course unit. The framework has another utility, it serves as an instrument to guide us (the instructor, the teacher, the researcher) through the implementation of an experimental process with accuracy.

We analyzed 79 projects carried out in PMIS course unit in the period of 2010/2011 to 2017/2018. Each project was developed by teams of one to four students. In total, 105 students were enrolled in these projects.

We can conclude that our adapted framework is applicable and appropriate to classify our course unit. In fact, with the exception of statistical issues (multivariate analysis, statistical models, parametric models, and non-parametric sampling) all other components of the framework are suitable.

Given that the framework guides the experimental process and that we obtained results in almost all items, we can conclude that the PMIS course unit is well designed for teaching/learning purposes.

There is further work to be done though. First, our adapted framework needs to be evaluated and enhanced by other instructors/researchers. Second, it is necessary to apply our framework in other courses inside and outside of computer science area. Finally, based on the lessons learned in this research it is recommendable to do some adjustments in our framework.

In this chapter we present also, our experimental environment. As we have seen, our approach focuses on PBL implementation in an educational environment. Thus, our approach aims to teach topics of the engineering and management process of the software process development in university courses in the Information Systems area. Our main goal is to train professionals for the industry.

The students, professionals in the near future, must be able to apply the most modern methodological approaches in the analysis and design of Information Systems, as well as manage the corresponding projects and development processes, in close cooperation with the most demanding quality and maturity procedural reference models. This dual training of students in engineering approaches and methodologies and in techniques and management methods are complemented with interpersonal skills. The dynamics promoted

by the teaching-learning project promoted by the integration of the four annual courses develops management, engineering, and personal skills. This tripartite training promoted by our approach instills in students an attitude and not only technological learning, in the Software Engineering context. The vision of *Engineering and Management Software* applied to IS problems is what industry seeks to build. So that engineering teams are increasingly able to intervene in organizations.

Using case studies, even with awareness of the limitations of possible inferences and simplifications when compared with empirical software engineering approaches, the students will continue their studies through various courses, capitalizing on experience, awareness, insight, and ability to avoid paths of less desirable decisions. By the end of their studies, trainees will be ready to integrate easily in the industrial world with developed maturity and valuable experience.

An engineer cannot be just a “mere” specialist: it is necessary to combine technical skills with the human and social dimensions. Our students should be prepared to live and work as global citizens, understand how engineers contribute to society. They should be aware that it is not enough to be technically excellent, because there are other dimensions to consider. Our approach allows delivering some important skills: (1) a basic understanding of business processes; (2) a product development with high-quality concerns; (3) know-how to conceive, design, implement and operate medium-size complexity systems and (4) communicative, initiative/leadership, teamwork, analytical and problem solving and personal abilities.

Chapter 5

Demonstration Case 1 – Use Case Points

5.1. Introduction

In this chapter, the first demonstration case developed to assess the contributions of this thesis is analyzed. The demonstration case was developed at an educational environment using graduate and undergraduate students as subjects. The demonstration case was assessed with the *Use Case Points Analysis* method in order to illustrate the main contributions of the previous chapter: empirical software engineering in teaching. It is included, also, an overview of the effort estimation methods.

A demonstration case was developed to assess the suitability of the *Use Case Points Analysis* method in educational context. It involved seven development software teams. The software project developed by the teams was requested by a real client that provided all the information about the organization and interacted directly with the teams (Alves et al., 2013).

In this demonstration case we used graduate and undergraduate students that were randomized grouped in seven teams to develop a software system. We applied the original UCP (Use Case Points) method for estimate the effort needed to develop the software system of each one teams.

The teams were constituted by second year students of the course 8604N5 Software System Development (SSD) from the undergraduate degree in Information Systems and Technology in University of Minho (the first University to offer in Portugal DEng, MSc, and PhD degrees in Computing). The teams had between 10 and 17 people (1 team with 10, 1 team with 14, 1 team with 15, 3 teams with 16 and 1 with 17). Each team receives a sequential identification number (Team 1, Team 2, ..., Team 7) and the description of the customer problem. The teams followed the guidelines established by the RUP (Rational Unified Process) reduced model, executing the phases of inception, elaboration, and construction (Borges et al.,

2011; Borges, 2007; Monteiro et al., 2012). The project lasted 3 months. Teams are encouraged to follow the RUP guidelines for organizing themselves in terms roles/responsibilities/team organization.

The teams following RUP used the eight roles proposed by the reduced model. Due to the complexity of the system, we have decided to instantiate two of the optional sub-roles: System Analyst (that corresponds to a part of the responsibilities of the project manager) and Software Architect (that corresponds to a part of the responsibilities of the integrator).

We promote that at least, any team had 1 project manager, 1 or 2 system analysts, 1 or 2 integrators, 1 software architect, 1 project reviewer, 1 process engineer, 4 to 6 implementers (programmers), 1 system administrator, 1 test manager and 1 system tester.

Each team element was characterized by mean of an online survey to collect information about age, gender, RUP role performed, and the number of working hours. The survey response was 100%. The Table 5.1 shows the teams characterization built based on that survey.

Table 5.1: Teams Characterization

Feature		Team1	Team2	Team3	Team4	Team5	Team6	Team7	Aggregation
Number of elements	Female	2	3	1	3	2	1	1	13
	Male	8	12	15	14	14	15	13	91
	Total	10	15	16	17	16	16	14	104
Average Age (Years)		22.5	21.6	21.9	21.5	20.2	24.4	30.4	23.2
Interpersonal level		Medium-Low	Medium-High	Medium-High	High	Very High	High	Medium	Medium-High
Choose aware of the role		61.5%	79.4%	73.5%	68.7%	73.3%	71.9%	59.4%	69.7%
Number of elements with experience		0	2	1	2	1	4	3	13
Previous role			Programmer		Several		Several		Several
Number of students workers		1	0	0	1	0	7	14	23
External consultant		Yes	Yes	Yes	Yes	No	No	No	Yes
Frequency type		Labor	Labor	Labor	Labor	Labor	Post-Labor	Post-Labor	Labor

Note that the pilot teams have not an improvement software development culture or sensitivity to references such as CMMI (Capability Maturity Model Integration) and RUP, since only 11.7% of the elements revealed experience with software development projects.

As presented and explained in the previous chapter, in our approach we involve students from BSc, MSc and PhD degrees in Computing from University of Minho. The integration among courses with students with different knowledge level and experience allows us to build a development infrastructure similar to a real

company. This educational environment context is centered in Project-Based Learning (PBL) training methodology (Alves, Ribeiro, et al., 2014; Alves et al., 2018).

Our approach presents an advance in Software Engineering area, in order to overcome some existing challenges. The teams have the opportunity to interact with a real client. They can learn and apply Software Engineering principles through a real software project. Thus, they can evolve and improve their technical and non-technical skills. In our setting we promote a win-win approach for all stakeholders: clients, students, teachers, and researchers. Clients will have state of the art projects implemented in their companies. Students can acquire technical and non-technical skills and work closely with real-world problems. Teachers will have the opportunity to teach technical knowledge authentically and realize new problems that companies are facing. Researchers can perform experiments in new and/or existing techniques, tools, and methods.

This first demonstration case study was performed in this educational environment context. We choose to implement this demonstration case in *software engineering process* area, topic *software measurement*, subtopic *software process measurement techniques* (IEEE, 2014). We were interested in apply some metrics to evaluate the complexity and effort and productivity of the teams.

To ensure compliance, in terms of cost, schedule and quality of an IT (Information and Technology) project is important to estimate the total amount of resources early in developing process. Currently, there are several methods to estimate the resources needed to develop a software system. For this propose, in our research we find *Function Points* (FPs) (Albrecht, 1979), *Use Case Points* (UCP) (Karner, 1993) and *LOC* (*Lines of Code*) methods.

As previously mentioned, our final goal of carrying out empirical studies with students is to understand its validity when compared with the corresponding studies in real industrial settings. The case study that involves software metrics seems to us to be suitable for this purpose.

5.2. Goals of the Demonstration Case 1

This demonstration case was performed in our empirical software engineering environment. As previously mentioned, we have a main research question/problem that is *How to Conduct Empirical Software Engineering in educational context?* We perform this demonstration case to validate our environment in terms of adaptability, reliability, and practicability.

In our experimental environment, we have several teams to develop a software product to a real client. Therefore, the research question/problem is:

Is it appropriate to apply the Use Case Point method to estimate the software size and teams productivity in teaching context?

Our final goal of carrying out empirical studies with students is to understand its validity when compared with the corresponding studies in real industrial settings. We intend to answer if it is adequate to use students as subjects in empirical studies? What is the better way to involve students as subjects in order to obtain valid results? Can we use the results with student in real context? The answers to all these questions are relevant to scientific and academic community, and many researchers have dedicated their work in this area.

5.3. Synopsis of Use Case Points Method

In this section, we present an overview of some effort estimation methods, especially the Use Case Points method. We present all steps realized to calculate the use case points of the software product of the teams. Finally, we present the analysis of the results with a comparison of the results collected from software industry.

5.3.1. Effort Estimation Methods

There are great efforts and contributions to measure the size of a software system and estimate the effort needed to develop it. Measuring the size of a software system is different to estimate the effort needed to develop it. The first is an activity that estimates a probability of a software system size in a measurement unit while the second estimates the effort required to developing it. The relationship between the size of a software system and effort required to develop it is given by the productivity of the software development team.

Metrics are measurement methodologies whose main objective is to estimate the size of software system and assist, as an indicator, the project management of software system development. The estimated size is one of the most used metrics for software size, since has direct impact on development effort and project management. It is an indicator of the amount of work to be performed and this kind of knowledge can be used to help us to estimate the cost and the lead time for the project (Karner, 1993). According to Pressman and Maxim, measurement enables managers to plan, monitor, improve and enhance the software process development (Pressman & Maxim, 2020).

Estimating the size of a software system is a critical development process activity. Not only does size impact the technical solution; it also impacts the project management solution (Ross, 2003).

The size of the software system means the amount of work to be performed in a project development. Each project can be estimated according to the physical size (which is measured through the requirements specification, analysis, construction, and testing). Based on the functions that the user gets, in the complexity of the problem that the software system will solve and in the reusability of the project, which measures how much the product will be copied or modified from another existing product (Fenton & Bieman, 2015).

Currently, there are several metrics of size estimation and it is difficult to select the most appropriate for the size of a software project in an organization. The main metrics were developed based on the software functions such as: *Function Points* (Albrecht & Gaffney, 1983), *Feature Points* (Jones, 1988), *Boeing' s 3D Function Points* (Whitmire, 1992), *Mark II* (Symons, 1991), *Use Case Points* (Karner, 1993), COSMIC Full Function Points (COSMIC, 2020; Symons, 2020).

5.3.1.1. LOC (Lines of Code)

The first metrics of software size estimation emerged in mid-1960's, although the first dedicated book on software metrics was not published until 1976 (Gilb, 1977). These metrics were based on the physical size of *Lines of Code* (LOC) and were used as the basis for "measuring programming productivity and effort" (Fenton & Neil, 2000). This metric considers the software from the perspective of the internal structure and it is applied in the final stages of the software project (Misic & Tesic, 1998).

Ross highlights two advantages of using LOC method: 1) the possibility to estimate automatically, and 2) the ease of using historical data because most of the existing data about estimation were measured by LOC method. The disadvantages are related to ambiguity, because the metric becomes ambiguous when dealing with non-textual abstractions and the lack of significance of the measure to the end user (customer) (Ross, 2003). Besides these disadvantages Fenton and Bieman note that a count in LOC depends on the degree of code reusing and the programming language and can be five times higher than another estimate, due to differences in techniques of measurement of blank lines, comment lines, data declaration and statements (Fenton & Bieman, 2015). The authors also emphasize that the LOC method penalizes small and well-designed programs, it is not adequate to nonprocedural programming languages and is difficult to obtain in the early planning stages of development of a software system (Fenton & Bieman, 2015).

LOC method was a metric widely used until mid-1970. From there emerged various programming languages and consequently the need for other ways to estimate the size of software.

5.3.1.2. Use Case Points

The *Use Case Points* (UCP) metric was defined to estimate Object Oriented (OO) projects based on the same philosophy of *Function Points* and in the process "Objectory", where the use case concept was developed. Later, Ivar Jacobson developed "*Object-Oriented Software Engineering (OOSE)*", methodology based in use cases, a technique widely used in industry to describe and collect the functional requirements of the software. Considering that the use cases model was developed to collect the requirements based on use and users vision, it makes sense to base the estimation of size and resources of software projects in use cases (Damodaran & Washington, 2002).

Since the UCP method was selected to perform the empirical study with the students, next we present the relevant characteristics and the counting process of this method.

5.3.2. Use Case Points Method

The first description of the method was published by Gustav Karner (Karner, 1993) with the aim of creating a model that would allow estimating the resources required to develop a software system under Objectory AB (later acquired by Rational Software). Its influences came from the classic *Function Point* method. However, after the original work, the author has not done any further publication or justified aspects of the conception of the method, thus it was being applied and adapted according to the circumstances. The method consists in calculating a metric called *Use Case Points* that give us an estimation of the size and complexity of a software project. If we know the development team productivity (to be obtained based on previous projects), we can derive an estimate of the effort required to develop the software project.

The UCPs are related to functional, technical, and environmental complexity of the software project, namely the following aspects:

- The number and complexity of the actors and use cases in the system;
- Several non-functional requirements with impact on the project (e.g. performance, usability, reliability, etc.);
- The impact of the environment where the project is developed.

When applying the method, we must first calculate the complexity of actors and use cases in the system to quantify the variables *Unadjusted Actor Weight* (UAW) and *Unadjusted Use Case Weight* (UUCW), respectively. When combined with their weight, we obtain an inadequate measure of the size and complexity of the system called *Unadjusted called Use Case Points* (UUCP). The next step is to adjust this measure with several technical factors and environmental factors given by *Technical Complexity Factor* (TCF) and *Environmental* Factor (EF) variables, respectively. These factors combined with the UUCP variable will produce the effective number of UCPs that reflect the size and complexity of the software project. In the following subsections, we detail the steps needed to calculate the UCPs.

***Unadjusted Actor Weight* (UAW)**

An actor is an entity that interacts with the system to achieve its goals: a person, a machine, a software application, etc. Thus, to capture these differences, each actor is classified as simple, average, or complex and each actor type is associated with a weight as shown in Table 5.2.

Table 5.2: Complexity of Actors

Category	Example	Weight
Simple	An external system through an API (Application Programming Interface)	1
Average	A user who interacts with the system through a command line	2
Complex	A user who interacts with the system through a Graphical User Interface (GUI)	3

Although we believe that a GUI (Graphical User Interface) is more complex than a command line or API (Application Programming Interface), there are no published studies that justify this aspect (Abran, 2010). The complexity of the actors in the system is given by the *Unadjusted Actor Weight* (UAW) variable and is calculated by the sum of the products between the actors in each category with their associated weights.

Unadjusted Use Case Weight (UUCW)

A use case is a story about how users interact with the system to achieve their goals. There are several ways to write a use case, however, to avoid that the measurement of the number of UCPs of the software

project is not affected, all use cases of the system must be written with the same level of detail. The user-objectives oriented approach described in (Cockburn, 2000) is suitable for apply the method.

Such as actors are classified according to their complexity, the use cases are also classified as simple, average, or complex through the number of transactions containing the main course of success and alternative courses, as shown in Table 5.3.

Table 5.3: Complexity of Use Cases

Category	Number of transactions	Weight
Simple	3 or less	5
Average	4 to 7	10
Complex	More than 7	15

It is noted that a transaction cannot be a step in the courses of use cases. Ivar Jacobson (the use cases author) defines a transaction as an action triggered by the user to which the system reacts and presents the results of its processing (Jacobson, 1992).

To calculate the *Unadjusted Use Case Weight* (UUCW) variable, each use case must be classified according to its category and the corresponding weight must be multiplied by the number of use cases the systems possess of each category.

Unadjusted Use Case Points (UUCP)

After calculating the complexity of actors and use cases of the system, the respective variables must be combined to determine an estimation of the *Unadjusted Use Case Points* (UUCP). UUCP is considered an unadjusted estimation because it will be adjusted by a set of technical and environmental factors to calculate the effective value of the UCPs of the software project. The calculation of UUCP uses the following equation:

$$\mathrm{UUCP} = \mathrm{UAW} + \mathrm{UUCW} \tag{5.1}$$

Technical Complexity Factor (TCF)

The size of the software system does not depend only on its functions and the users it serves. It also depends on the quality characteristics of the system. Therefore, there is a number of technical or non-functional factors that must be considered to measure the size of the system based on what was agreed

between the customer and the supplier. A list describing all technical factors proposed by Karner is shown in Table 5.4 (Karner, 1993).

The impact of each factor within the project is rated on a scale 0, 1, 2, 3, 4 and 5, where 0 means that it is irrelevant and 5 means it is essential. If the factor is neither important nor irrelevant it must be rated with the value of 3. The rate of the factor must be multiplied by the associated weight as shown in Table 5.4. The sum of all the products calculates the *TFactor* value, which is used to calculate the *Technical Complexity Factor* (TCF):

$$\mathrm{TCF} = 0.6 + (0.01 \times \mathrm{TFactor}) \tag{5.2}$$

Table 5.4: Technical Factors Contributing to Complexity

Factor	Description	Weight
T1	Distributed systems	2
T2	Performance	2
T3	Efficiency	1
T4	Complex internal processing	1
T5	Code reusability	1
T6	Installation ease	0.5
T7	Usability	0.5
T8	Portability	2
T9	Changeability	1
T10	Concurrency	1
T11	Security	1
T12	Accessibility of others	1
T13	Training	1

Karner based the constants 0.6 and 0.01 on the adjustment factors of *Function Points* created by Albrecht (Albrecht, 1979). For example, for a given system with a *TFactor* equal to 42, the TCF is:

$$\mathrm{TCF} = 0.6 + (0.01 \times 42) = 1.02 \tag{5.3}$$

Environmental Factor (EF)

The characterization of the software development teams is also important to obtain a measure of the size and complexity of the software project. Thus, there is a set of aspects related to the development

environment that must be weighed. The levels of the motivation of the team or the familiarity with the software process development adopted are examples of environmental factors that must be considered on the calculation of the UCP value. A list describing all environmental factors proposed by Karner is shown in Table 5.5 (Karner, 1993).

Table 5.5: Environmental Factors Contributing to Efficiency

Factor	Description	Weight
E1	Experience with a software development process	1.5
E2	Development experience with similar projects	0.5
E3	Experience with OOP	1
E4	Maturity in OO analysis	0.5
E5	Motivation	1
E6	Stability of requirements	2
E7	Part time workers	-1
E8	Experience with technologies adopted	-1

The impact of each factor within the project is rated on a scale 0, 1, 2, 3, 4 and 5, where 0 means that it is irrelevant and 5 means it is essential. If the factor is neither important nor irrelevant it must be rated with the value of 3. The rate of the factor must be multiplied by the associated weight as shown in Table 5.5. The sum of all the products calculates the *EFactor* value, which is used to calculate the *Environmental Factor* (EF):

$$\mathrm{EF} = 1.4 + (-0.03 \times \mathrm{EFactor}) \tag{5.4}$$

For example, for a given system with an *EFactor* equal to 17.5, the EF is:

$$\mathrm{EF} = 1.4 + (-0.03 \times 17.5) = 1.4 + (-0.525) = 0.875 \tag{5.5}$$

Use Case Points (UCP)

Finally, after calculating the above variables shown in the previous subsections, the last step is to calculate the effective number of UCPs of the system using the following equation:

$$\mathrm{UCP} = \mathrm{UUCP} \times \mathrm{TCF} \times \mathrm{EF} \tag{5.6}$$

If the team productivity is known, the UCP value can be used to estimate the effort required to develop a software system. According to Karner, each UCP corresponds to 20 man-hours of work (Karner, 1993).

5.3.3. Use Case Points Method Application and Results

Based on the previously described approach for calculating the UCP, a demonstration case was developed to determine the productivity of some student software development teams (Alves et al., 2013).

The teams developed a software project of medium complexity, using the UML notation encompassed in an iterative and incremental software development process, in this case, the RUP. The teams followed the guidelines established by the RUP reduced model (Borges et al., 2011; Borges, Monteiro, & Machado, 2012), executing the phases of inception, elaboration, and construction according to the best practices suggested by CMMI-DEV v1.2 ML2. The project lasted 3 months. This software project was to develop a Web solution using object-oriented technologies and relational databases, to support the information system of one local customer that provided all the information about the organization and interacted directly with the teams.

The main goal of case study was to calculate the productivity of the teams in man-hours per UCP, based on the size of each software project and in effort reported by each team. To determine the team's productivity, we have used the following equation:

$$\text{Productivity} = \text{Effort}[\text{h}] \,/\, \text{Size}[\text{UCP}] \tag{5.7}$$

The measurement of the functional size expressed by UUCP metric was based on the latest version of the use case model that developed by each team. Each team has chosen a different number of components to develop. The quality attributes and restrictions were also different for each team. These differences had considerable impact on the calculation of the TCF. Table 5.6 shows the table with the weights of technical factors assigned to each of the teams. Each one of the technical factors was enumerated from T1 to T13 according to original propose by Karner. Tm1 to Tm7 represents one of each team involved in the case study.

Table 5.7 presents the weight of environmental factors across all teams. Each one of environmental factors was enumerated from E1 to E8 according to original propose by Karner. The weight was assigned by each one of the teams.

Table 5.6: TCF Calculation for each team

F.	Description	W.	Tm1	Res	Tm2	Res	Tm3	Res	Tm4	Res	Tm5	Res	Tm6	Res	Tm7	Res
T1	Distributed systems	2	3	6	3	6	3	6	3	6	3	6	3	6	3	6
T2	Performance	2	5	10	5	10	5	10	5	10	5	10	5	10	5	10
T3	Efficiency	1	5	5	5	5	5	5	5	5	5	5	5	5	5	5
T4	Complex internal processing	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3
T5	Code reusability	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T6	Installation ease	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T7	Usability	0.5	5	2.5	5	2.5	5	2.5	5	2.5	5	2.5	5	2.5	5	2.5
T8	Portability	2	3	6	3	6	3	6	3	6	3	6	3	6	0	0
T9	Changeability	1	3	3	3	3	3	3	3	3	3	3	3	3	0	0
T10	Concurrency	1	3	3	0	0	3	3	3	3	3	3	3	3	0	0
T11	Security	1	3	3	3	3	5	5	3	3	5	5	3	3	3	3
T12	Accessibility of others	1	3	3	3	3	3	3	3	3	5	5	3	3	3	3
T13	Training	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3
			TFactor	47.5		44.5		49.5		47.5		51.5		47.5		35.5
			TCF	**1.075**		**1.045**		**1.095**		**1.075**		**1.115**		**1.075**		**0.955**

Table 5.7: EF Calculation for each team

Factor	Description	Weight	Weight (Teams)	Res.
E1	Experience with RUP	1.5	3	4.5
E2	Development experience with similar projects	0.5	0	0
E3	Experience with OOP	1	3	3
E4	Maturity in OO analysis	0.5	1	0.5
E5	Motivation	1	3	3
E6	Stability of requirements	2	3	6
E7	Part time workers	-1	1	-1
E8	Experience with technologies	-1	3	-3
			EFactor	13
			EF	**1.01**

To calculate the team' s productivity we have used the real effort that each team reported to develop the software system. This effort involved hours spent within the project, as well as attending lectures related with the SSD course.

Table 5.8 presents the metrics collected from each development team. From this table we can conclude that the productivity factors of the teams vary between 5.5 hours to 12.1 hours per UCP. On average, the productivity factor per team is 9.36 hours per UCP. These results are directly related to the total effort accomplished in hours and the size of projects measured in UCPs. In Table 5.8 is possible to observe that number of members of each team vary from 10 to 17. The software application of each team also presents some variation in terms of size and complexity, where the number of requirements they meet have some difference.

Table 5.8: Results of the Software Development Teams

Description	Team1	Team2	Team3	Team4	Team5	Team6	Team7
Number of elements	10	15	16	17	16	16	14
Total effort [hours]	2094	2118	2511	5467.5	3517.5	4287	4548
Unadjusted Actor Weight (UAW)	36	27	45	42	30	21	87
Unadjusted Use Case Weight (UUCW)	170	165	365	400	300	305	570
Unadjusted Use Case Points (UUCP)	206	192	410	442	330	326	657
Technical Complexity Factor (TCF)	1.075	1.045	1.095	1.075	1.115	1.075	0.955
Environmental Factor (EF)	1.01	1.01	1.01	1.01	1.01	1.01	1.01
Software size	221.45	206.4	448.95	475.15	367.95	350.45	627.44
Use Case Points (UCP)	224	203	453	480	372	354	634
Productivity [hours / UCP]	9.36	10.45	5.54	11.39	9.47	12.11	7.18

In general, teams with more elements reported more effort. Normally, we tend to assume that the size of projects in UCPs is proportional to the functional size. However, this did not happen to all teams, as seen in Figure 5.1.

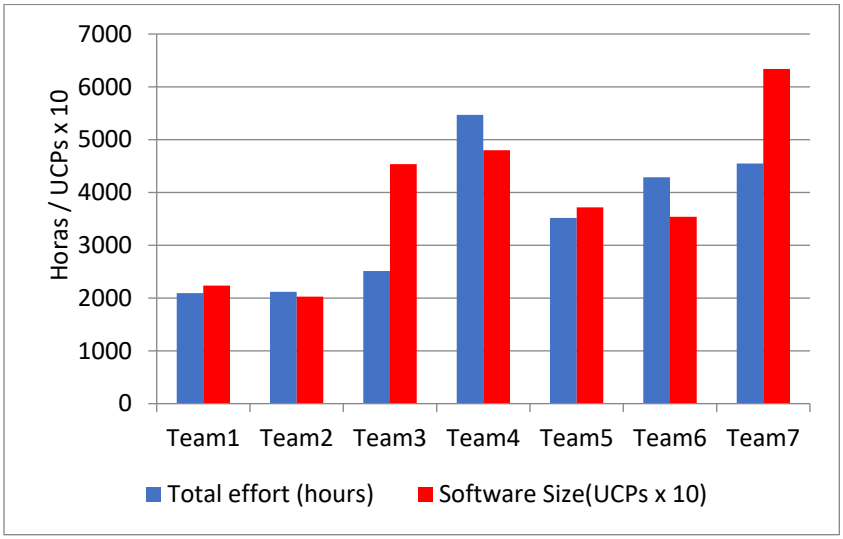


Figure 5.1: Comparison between Total Effort and Software Size per Team

This discrepancy occurs primarily for two reasons. The first indicates that the teams with more members (and reporting more effort) give more emphasis on documentation tasks (mainly the writing of reports). The other reason is due to the quality of artefacts (mainly the use case models) that are limited by the reduced experience of the teams in requirements specification. We could observe in all teams, use cases with poor quality, leading to a subjective interpretation to identify transactions in the main and alternative courses. Additionally, most of the use cases were classified as simple according the UCP method because it had few transactions.

Based on the customer assessment, the teams with the highest software project size in UCPs had the best results, especially teams Team3, Team4 and Team7 (except for the Team6). Finally, it was also found that the use case models of these teams were superior to the others.

5.4. Analysis of the Experimentation Framework

As mentioned in the previous chapter, the main objective of the framework is to provide a classification scheme for understanding and evaluating empirical studies in software engineering area in an educational context. However, the framework also has another purpose, it helps structuring experimental processes. In this chapter, we describe the demonstration case 1. This demonstration case reports an empirical study on the estimation of software development effort with Use Case Points (Alves et al., 2013). This study was entirely performed in our experimental environment. All artifacts, students and teacher staff belong to this environment. Thus, it makes perfect sense to analyze this empirical study using our adapted experimentation framework.

The framework of experimentation, summarized in Figure 4.3, consists of four categories corresponding to phases of the experimentation process: 1) definition, 2) planning, 3) operation, and 4) interpretation. The following sections discuss each of these four phases. The classification of the empirical study for each part of each of the phases is presented below.

In the definition phase of the experimental process, we have six parts: 1) motivation, 2) object, 3) purpose, 4) perspective, 5) scope, and 6) end user. As motivation for this empirical study was to understand, assess and validate the UCP metric. First, it was necessary to understand the UCP metric, and then, it was necessary to assess and validate if this metric is appropriate to estimate the software size and teams' s productivity in educational context. The object of the empirical study was a metric, in this case the UCP metric. The purpose of the study was to evaluate the effectiveness of the UCP metric in educational context.

The study examines the UCP metric for a given project type from the perspective of the project manager. In this empirical study, we can consider also, the perspective of the researcher. In fact, the author of this thesis also followed this study from a research perspective. A general classification of the scopes of experimental studies can be obtained by examining the sizes of the two domains considered, namely, the number of teams per project and the number of projects. In the empirical study, the scope is replicated project because it was analyzed seven teams that performed one project. Finally, the end user of this empirical study is real-world-like because the environment simulates an industrial context, and the object of the study is a real effort estimation metric.

In the planning phase of the experimental process, we have three parts: 1) design, 2) criteria, and 3) measurement. The design of an experiment couples the study scope with analytical methods and indicates the domain samples to be examined (Basili et al., 1986). In our adapted framework of experimentation, we added in design part, how the teams would be formed. In this empirical study the team's formation was self-organized. In fact, the number of members per team varied between 10 and 17 students. The teams were free to choose their elements, although each of them played a well-defined role according to the reduced RUP model. The teams did not resort to statistical models (multivariate analysis, statistical models, parametric models, and non-parametric sampling) in their projects and the empirical study also did not use this type of analysis. Different motivations, objects, purposes, perspectives, scopes, and end users require the examination of different criteria. From perspective of external validity, we classify de empirical study as reliability and from perspective of learning benefits, we classify as integrability with the course. In fact, all subjects in this empirical study belongs to course syllabus. In the measurement part of the planning phase, we intend to capture some aspects about metrics, data collection, type of research and level of measurement that our students used in their projects. In this empirical study, we classify the measurement part as a metric validation. In fact, one of the objectives of the study is to know the applicability of the UCP metric in an educational context. The require data was collected by an objective way.

In the operation phase of the experimental process, we have three parts: 1) preparation, 2) execution, and 3) analysis. In this empirical study, we classify as main sources of preparation, the literature study, and classes. During the execution of this empirical study, data collection is the main action to achieve the project goals. In terms of analysis of the data, we classify the study as using a quantitative method. In this study, no preliminary data analysis was performed.

In the interpretation phase of the experimental process, we have three parts: 1) interpretation context, 2) extrapolation, and 3) impact. The results of the data analysis from a study are interpreted in a broadening series of contexts. These contexts of interpretation are the statistical framework in which the result is derived, the purpose of the particular study, and the knowledge in the field of research (Basili et al., 1986). In this empirical study, we classify the interpretation context as field of research. In our opinion it is the most suitable classification. The representativeness of the sampling analyzed in a study qualifies the extrapolation of the results to other environments. In this empirical study, we classify the sample representativeness as average. In fact, the projects where the data was collected had medium and high complexity. Each one of this seven projects was performed by teams of 10 to 17 students. Several follow-up activities contribute to the impact of a study: presenting/publishing the results for feedback, replicating the experiment, and actually applying the results by modifying methods for software development, maintenance, management, and research (Basili et al., 1986). In terms of the impact, we classify this empirical study as visibility from academia point of view. It would perhaps be too bold to classify it as having visibility from the industry's point of view.

Finally, it is important to mention that the framework of experimentation was used to discipline the experimentation process, both for the students involved in the projects and for the researcher during the development of this empirical study.

5.5. Conclusion

Empirical studies are important in software engineering to evaluate new tools, techniques, methods, and technologies in a structured way before they are introduced in the industrial (real) software process.

In the empirical study, the average productivity of the teams was based on the size of the software projects in UCPs and in total effort made throughout all phases of the software project development. We believe that the average productivity could be used in future situations: (1) in the business planning phase of the project within the Software Process and Methodologies (SPM) course (first semester) to estimate the development effort of the project to implement in the SSD course (second semester); (2) during the implementation phase for control purposes, to calibrate the effort of every moment of the project. The results could also be packaged in database (repositories) with historical data with projects information. This data can include all measurements obtained in all projects of the teams.

After application of the UCP method in three real projects, Karner found that it takes 20 man-hours to complete one UCP (Karner, 1993). This factor has been accepted as an historically collected figure

representing productivity (Anda, Dreiem, Sjoberg, & Jorgensen, 2001; Damodaran & Washington, 2002). The teams involved in the case study presented coefficients of productivity more generous. This does not mean that they are super teams but means that further research is needed to find the reason of this discrepancy. However, we believe that empirical studies involving students on these subjects are important for the scientific community and the industry.

Considering also that productivity is directly related to the effort made and with the functional size per UCP, we suggest for future editions of the SMP and SSD courses that all teams use a development tool, for example Teamwork Project Manager (Teamwork Project Manager, 2021), in order to accurately determine the effective involved effort. Assess the influence of this tool in the team's performance would be an interesting experiment.

Chapter 6

Demonstration Case 2 – Function Points

6.1. Introduction

In this chapter, the second demonstration case developed to assess the contributions of this thesis is analyzed. The demonstration case was developed at an educational environment using graduate and undergraduate students as subjects. The demonstration case was assessed with the *Function Points Analysis* method in order to illustrate the main contributions of the chapter: empirical software engineering in teaching. In order to understand the application of the method the original function point counting procedure it was presented.

As previously mentioned in a previous chapter, to assess our contributions, a set of demonstration cases were implemented. The demonstration cases were developed at an educational environment. In this educational environment, we involve students from BSc, MSc, and PhD degrees in Computing from University of Minho.

To provide technical and non-technical knowledge and soft skills we adopted the Project Based Learning (PBL) training methodology. With this educational approach we intend to train students for industry, by involving them with real clients within the development of software projects.

The characteristics that influence the success of any organization, in particular, IT (Information and Technology) organizations in the competitive and globalized current market are efficiency, effectiveness, delivery the product on time, within budget and with a level of quality desired by the customer. In this sense, we must emphasize the importance of mechanisms for monitoring, control and evaluating the progress of process, project, and product.

Currently, the level of competition among organizations is related to the efficiency of their information systems. The organizations need to adopt more and more new systems. This means that the costs of

development and maintenance for organizations are critical parameters in its management. To control costs with the acquisition of software systems is necessary that organizations do the task of size estimation of the systems that intend to adopt, because you cannot manage what you cannot measure (Humphrey, 1995). The emergence of the estimation method, namely, *Function Points Analysis* (FPA) has allowed to the IT community a significant increase of software measurement practice. However, the count of *Function Points* (FPs) requires a descriptive documentation, such as specifications of the software functionalities.

Based on knowledge of size and complexity of software applications, we can estimate the total amount of resources needed for all developing process. Currently, there are several methods to estimate the resources needed to develop a software system. For this propose, in our research we find *Function Points Analysis* (Albrecht, 1979), *Use Case Points* (Karner, 1993) and *LOC* (Lines of Code) methods.

In this demonstration case we used graduate and undergraduate students that were randomized grouped in two teams to develop a software system. We applied the original FPA method for estimate the size and complexity of the software system developed by each team. Based on knowledge of size and complexity of software applications we can estimate the total amount of resources needed for all developing process.

6.2. Goals of the Demonstration Case 2

Such as the previous demonstration case presented in the previous chapter this demonstration case was performed in our empirical software engineering environment. Also, as previously mentioned, we have a main research question/problem that is *How to Conduct Empirical Software Engineering in educational context?* We perform this demonstration case to validate our environment in terms of adaptability, reliability, and practicability.

In our experimental environment, we have several teams to develop a software product to a real client. Therefore, the research question/problem is:

Is it appropriate to apply the Function Points method to estimate the software size and teams productivity in teaching context?

As mentioned, several times, our final goal of carrying out empirical studies with students is to understand its validity when compared with the corresponding studies in real industrial settings. Are the results obtained in experiments using students comparable with the results obtained in companies? With the results we obtain, are we making a new contribution to scientific knowledge? All these questions are pertinent

and the answers are not easy to get. Throughout this dissertation, not being able to give an exact answer, we tried to do it in an approximate way.

6.3. Synopsis of Function Points Analysis Method

6.3.1. Overview of the FPA

The first description of the method was published in 1979 by Allan Albrecht with the aim of creating a model that would allow measure productivity over all phases of a project independently on the programming language and technologies used. This method arises as an attempt to minimize the difficulties associated with lines of code as a measure of software size, and to assist in generating a mechanism that could predict the effort associated with software development (Albrecht, 1979; Longstreet, 2005). The first version of *Function Points* aimed to be applied early in the software development process, to be related to economic productivity and to be independent of the source code or programming language.

In 1984, Allan Albrecht refined this version and later, with the increased use of the *Function Points* method, it became necessary to define a guide to interpret the original rules for new environments. Due to this need, in 1986 was created the *International Function Point Users Group* (IFPUG) (IFPUG, 2021). From this date, the IFPUG became responsible for defining the counting rules, the training and certification of professionals interested in the application of this metric and dissemination of several historical databases of productivity of software development in industry, provided by various agencies. This information enables the estimation of development time for new software projects and team productivity based on previous estimates made by the *Function Points* method (Garmus & Herron, 2000; Longstreet, 2005). The *Function Points* has become the most widely used and studied metric in the history of software engineering and in late 1993 had groups of users in 18 countries (Jones, 1994).

Actually IFPUG have three types of regular memberships, namely, individual membership, corporate membership and worldwide corporate membership with different benefits (IFPUG, 2021). They publish a "Function Point Counting Practices Manual" (IFPUG, 2010), hold semi-annual conferences, and disseminate information to promote the use of the function point process (DeMarco, 1992).

In 1998 was founded the *International Software Benchmarking Group* (ISBSG) with collaboration of members of IFPUG. The ISBSG has hundreds of members, such as students, programmers, consultants and system managers (ISBSG, 2021).

Another association is the *Netherlands Software Metrics Association* (NESMA). NESMA is a metric Netherlands association that uses a different standard of measurement when compared to IFPUG. However, their goals are very similar to the IFPUG (NESMA, 2021). NESMA has as mission to develop software measurable to allow fact-based decisions on the business value of software, so software can be deployed successfully. NESMA connects organizations and individuals who are involved in making software measurable and is the center of knowledge in the field of software measurements and cost engineering for IT.

It is NESMA' s mission to: (1) Spread knowledge about software measurement and software metrics; (2) Act as a Body of Knowledge for the industry regarding the use of software metrics in all business areas; (3) Remain independent, objective and not-for-profit; (4) Research the applicability of software metrics in all business areas; (5) Connect relevant organizations in the industry that NESMA feels are expert in one of the areas where software measurement and metrics are important; (6) Produce relevant guidelines, reports and other information products that are useful for the software industry; (7) Produce a platform where people can discuss issues they experience with software measurement and metrics or where they can exchange ideas and/or knowledge (NESMA, 2021).

The FPA is one of the first metrics to measure the size of software with some precision. It is the most used in the industry and became an international standard in 2002 through the ISO/IEC 20926 (Dekkers, 2003). Currently, the mapping of the FPA method to estimate object-oriented software projects has been widely discussed in the literature.

The use of FP metric is not a trivial process, requires some practice to apply all the rules presented by *International Function Point Users Group* (IFPUG, 2021). However, FPA helps developers and users quantify the size and complexity of software application functions in a way that is useful to software users (Furey, 1997).

Some researchers have proposed the mapping of the use case driven Object-Oriented Software Engineering (OOSE) method by Jacobson *et al.* into the abstract FPA model. The mapping proposed by the authors has been formulated as a small set of concise rules that support the actual measurement process (Fetcke, Abran, & Tho-Hau, 1998). Other authors based their approach on a class diagram including messages sent between classes. These authors considered each class as an internal logical file and treated messages sent outside the system boundary as transactions (Whitmire, 1993). Uemura *et al.* proposed detailed FPA measurement rules for the design specifications based on the UML (Unified Modeling Language) and develop the function point measurement tool, whose input products are design specifications on Rational

Rose. They used the sequence diagrams and class diagrams from UML notation (Uemura, Kusumoto, & Inoue, 1999).

Some authors have gone beyond mapping the FPA to the context of OO (Object Oriented) proposing new methods, such as: *Fast Count* (Tichenor, 1997), *Object-Oriented Function Point* (OOFP) (Caldiera et al., 1998), and *Object-Oriented Design and Function Points* (OODFP) (Ram & Raju, 2000).

The *Fast Count* method, a variant of the FPA is used since 1993 by the IRS (International Revenue Service), a USA government agency responsible for tax collection and tax law enforcement. This method can determine the software size about four times faster than industry averages can estimate software size and resource requirements for new development projects. In this method the FPs count is based on the central data model and Staffing Estimator, which can accurately size small software work orders and estimate corresponding resources needed without examining the software or meeting with the project team (Tichenor, 1997).

The *Object-Oriented Function Point* (OOFP) created by Caldiera *et al.*, maps the FPA concepts to OO software according of Object Modeling Techniques (OMT) notation. These authors define the counting procedure of the inheritance, aggregation and polymorphism based on the object model developed in the design phase and propose to estimate the software size at different points in the software development as new artifacts are available. The OOFP was applied in eight sub-systems, developed in an industrial environment producing software for telecommunications. These sub-systems were also counted in LOC so that authors could apply regression techniques and find the association between LOC and OOFP methods (Caldiera et al., 1998).

The *Object-Oriented Design Function Points* (OODFP) was adapted by Ram and Raju (2000) from the counting procedures defined by IFPUG. This method estimates the size of OO software in the design phase, based on the functions of the software and the complexity of classes. The complexity of the class can be low, average, or high according to a numeric value defined by the authors based on observations of different projects. For these authors, "a logical file is a collection of data elements which are visible to all methods of a class" and "transactional functions are the methods in a class" (Ram & Raju, 2000).

Some studies on the application of FPA method have shown a decrease in variation of function point count between different counters trained and certificates. Furey cites a study developed in 1994 by the *Quality Assurance Institute* and IFPUG that found a counting variance between trained counters about 11 percent (Furey, 1997). However, Kichenham argues that function point counting involves judgment on the part of the

counter. The author refers a study performed by Chris Kemerer that reports a 12-percent difference for the same product by different people in the same organization. The author also refers a study by Graham Low and Ross Jeffery that report "worse figures": a 30-percent variance within an organization, which rose to more than 30 percent across organizations (Kitchenham, 1997).

6.3.2. Advantages and Flaws of the FPA

The use of FPA metric is not a trivial process, requires some practice to apply all the rules presented by IFPUG. However, FPA helps developers and users quantify the size and complexity of software application functions in a way that is useful to software users. This metric presents some advantages, namely (Furey, 1997):

- Technology independence: FPs measure functional size independent of any development tool or technology used;
- Consistency and repeatability: the existence of a counting manual created by IFPUG helped achieve consistent counts and enable their reuse since it has standardized the counting process;
- Data normalization: FPs let organizations normalize data such as cost, effort, duration, defects, staff, and so on;
- Estimating and comparing: because FPs are based on functional requirements, they can be estimated and counted much earlier and more accurately than LOC. The use of this metric facilitates the comparison of different technologies or software process development with regard to productivity;
- Scope and expectations: customers often express the desire to change or improve their products, these events have a significant impact on software development. The adoption of this metric in these cases allows quantifying the impact of change or improvement action by the number of FPs.

The FPs have some flaws, the main flaw identified relates to construction problems. Albrecht function point construction involves classifying inputs, outputs, logical master files, system interfaces, and queries as simple, average, and complex. This means that the absolute scale counts are reduced to ordinal scale measures, and you can no longer apply other transformations to make them interval or ratio scale (Kitchenham, 1997). Felton and Bieman identified several problems in using this method (Fenton & Bieman, 2015). Follow we describe only some of these problems:

- Problems with early life-cycle use: FPs calculation requires a full software system specification, a user requirements document is not sufficient;
- Problems with application domain: FPs have been used successfully in data-processing applications, but their use for real-time and scientific application is controversial;
- Problems with accuracy: the use of FPs carries the correlation between their constituent elements. This means that under a statistical perspective, the adjustment factor is not significant for improving estimates (Kitchenham & Känsäla, 1993);
- Problems with differentiating specified items: because evaluating the input items and technology factor components involves expert judgment, the calculation of FPs from specification cannot be completely automated. To minimize subjectivity and ensure some consistency across different evaluators, organizations such as IFPUG publish detailed counting rules.

These are the main flaw of the FPA collected from literature. As mentioned earlier, FPA is a metric that has been used for many years and has undergone some improvements.

6.3.3. Objectives

The FPA method aims to establish a measure of "size" of software in FPs (unit of measure for software as well as the hour is unit of measurement for time), by quantifying the functions implemented from the point of view of user in a business focus and not in technical focus (Longstreet, 2005). As the basic principle, this metric focus on requirements specification to obtain estimates of time, cost, effort and resources in early phase of software development process independently of the technology used for their implementation (Furey, 1997). It provides a count indicative early in software development process without knowing details of the data model. Later in the software development process, this count becomes more accurate estimate of the complexity of functions, and, at the end of software development (in the implementation phase), a detailed count is performed, obtained from the degree of complexity of the functions raised in the functional process, data model, description of the screens and reports (IFPUG, 2021; Longstreet, 2005).

In general, organizations apply the FPs as "a method for determining the size of the software package purchased; to support the analysis of productivity and quality; to estimate the costs, resources and efforts of development projects and maintenance of software" (Garmus & Herron, 2000). In addition to these purposes, Longstreet highlights other utilities of the FPs method: "define when and where to make reengineering;

estimate test cases; understand the increase of scope; assist in contract negotiations; develop a standard set of metrics; and make benchmark of software'' (Longstreet, 2005).

6.3.4. Counting Process of Function Points

This section aims to present the counting process that will be adopted to calculate FPs of the applications developed by the teams enrolled in our computing courses. Figure 6.1 shows the function point counting procedure.

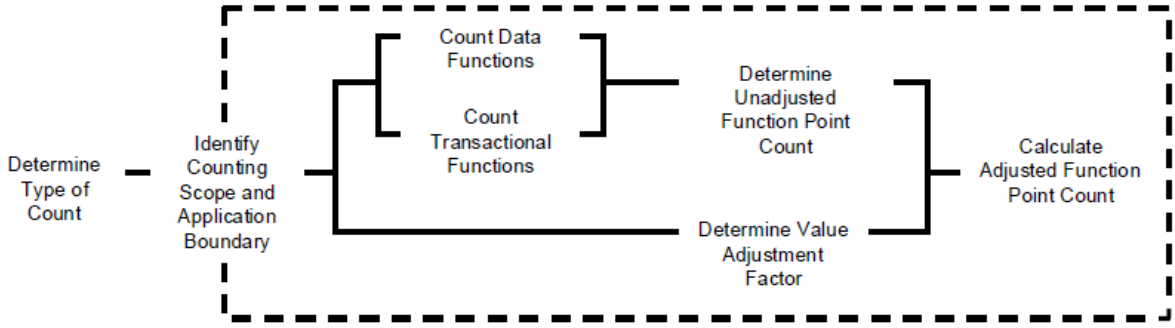


Figure 6.1: High-level procedure for function point counting (extracted from (IFPUG, 2010))

Briefly, the counting process comprises the following steps (IFPUG, 2010):

i. **Determine Type of Count** - In this step, the main goal is to determine the type of function point count. There are three types of function point counts:
   - Development project function point count: measures the functions provided to the users with the first installation of the software delivered when the project is complete;
   - Enhancement project function point count: measures the modifications to the existing application that add, change, or delete user functions delivered when the project is complete;
   - Application function point count: measures the current functions that the application provides to the user. This number is initialized when the development project function point count is completed. It is updated every time completion of an enhancement project alters the application's functions.

ii. **Identify the Counting Scope and Application Boundary** - after determining the type of count is necessary to identify the functionalities that will be included in the counting of FPs. The counting scope defines the functionality that will be included in a particular function point count.

The application boundary indicates the border between the software being measured and the user.

iii. **Count Data Functions** - the data functions represent the functionalities that are related to the internal and external data of the application. These data functions are called *Internal Logical Files* (ILFs) and *External Interfaces Files* (EIFs). The ILFs are a user identifiable group of logically related data that resides entirely within the applications boundary and is maintained through external inputs. The EIFs are a user identifiable group of logically related data that is used for reference purposes only. The data resides entirely outside the application and is maintained by another application. The EIF is an internal logical file for another application (Longstreet, 2005).

iv. **Count Transactional Functions** - represent the functionality provided to the user to process data. Transactional functions are external inputs, external outputs, or external inquiries. *External Inputs* (EIs) is an elementary process in which data crosses the boundary from outside to inside. This data may come from a data input screen or another application. The data may be used to maintain one or more ILF. The data can be either control information or business information. *External Outputs* (EOs) is an elementary process in which derived data passes across the boundary from inside to outside. Additionally, an EO may update an ILF. The data creates reports or output files sent to other applications. These reports and files are created from one or more ILFs and EIF. The processing logic must contain at least one mathematical formula or calculation or create derived data. *External Inquiries* (EQs) is an elementary process with both input and output components that result in data retrieval from one or more ILF and EIF. The input process does not update any ILF, and the output side does not contain derived data. The processing logic contains no mathematical formula or calculation, and creates no derived data (Longstreet, 2005).

v. **Determine Unadjusted Function Point Count** - the *Unadjusted Function Point Count* (UFPC) reflects the specific countable functionality provided to the user by the project or application. The application's specific user functionality is evaluated in terms of what is delivered by the application, not how it is delivered. Only user-requested and defined components are counted. The UFPC count has two function types: data and transactional.

vi. **Determine Value Adjustment Factor** - after completing the previous steps is necessary to calculate the *Value Adjustment Factor* (VAF), because the previous steps reflect only the functionalities that the system will provide to the user and are not considered the system

specifications. The VAF is based on fourteen *General System Characteristics* (GSCs) that rate the general functionality of the application being counted. The fourteen GSCs are:

- Data Communications;
- Distributed Data Processing;
- Performance;
- Heavily Used Configuration;
- Transaction Rate;
- Online Data Entry;
- End-User Efficiency;
- Online Update;
- Complex Processing;
- Reusability;
- Installation Ease;
- Operational Ease;
- Multiple Sites;
- Facilitate Change.

Each characteristic has associated descriptions that help determine the *Degrees of Influence* (DI) of the characteristics. The DI range on a scale of zero to five. The ratings are: 0 means not present, or no influence, 1 incidental influence, 2 moderate influence, 3 average influence, 4 significant influence and 5 strong influence throughout.

vii. **Calculate Adjusted Function Point Count** - after calculating the VAF, the *Adjusted Function Points* (AFP) will be adjusted by multiplying the value of the UFPC calculated in step v by VAF calculated in step vi.

6.3.5. Function Points Analysis Method Application and Results

The demonstration case was developed to determine the productivity of two software development teams using the original procedure for function point counting. The software project developed by the teams was requested by a real client that provided all the information about the organization and interacted directly with the teams (Alves, Oliveira, et al., 2014).

The teams were constituted by second year students of the course 8604N5 Software System Development (SSD) from the undergraduate degree in Information Systems and Technology in University of Minho. The two teams were composed of 14 students each one. Each team receives a sequential identification letter (Team A and Team B) and the description of the customer problem.

The teams developed a software project of medium complexity, using UML notation encompassed in an iterative and incremental software development process, in this case, the Rational Unified Process (RUP).

The teams followed the guidelines established by the RUP reduced model (Borges et al., 2011), executing the phases of inception, elaboration and construction according to the best practices suggested by CMMI-DEV v1.2 ML2. The project lasted 3 months. The teams use the RUP reduced model in order to make the application development more expeditious.

The software project was to develop a Web solution using object-oriented technologies and relational databases, to support the information system of one local client that provided all the information about the organization and interacted directly with the teams. Specifically, the technologies used were MySQL, PHP and Java. The teams had regular meetings with the client, especially in the phase of gathering the requirements of the software application to be developed.

The teams following RUP used the eight roles proposed by the reduced model. Due to the complexity of the system, we have decided to instantiate two of the optional sub-roles: system analyst (that corresponds to a part of the responsibilities of the project manager) and software architect (that corresponds to a part of the responsibilities of the integrator). Team organization was as follows:

- 1 project manager;
- 1 system analyst;
- 1 integrator;
- 1 software architect;
- 1 project reviewer;
- 1 process engineer;
- 5 implementers (programmers);
- 1 system administrator;
- 1 test manager and,
- 1 system tester.

Each team element was characterized by mean of an online survey to collect information about age, gender, RUP role performed, and the number of working hours. The survey response was 100%.

The process of identifying the various components for the calculation of FPs was a manual process. This count was obtained from an intensive study of the code of the applications developed by the two teams. The FPs counting process is not trivial, but its adoption is of great importance when we want to measure the software development. The FPs can support the project management activities, since we can only manage what we can measure.

The following are the main decisions made at each step of the FP calculation for each team. Thus,

i. **Determine Type of Count** - the type of count adapted to calculate FPs of the applications developed by Teams A and B was "application function point count" because the development software process is already completed.

ii. **Identify the Counting Scope and Application Boundary** - the applications developed by teams does not have any interaction with other systems.

iii. **Count Data Functions** - In this step were identified and counted the ILFs and EIFs of the applications developed by Teams A and B. When the FPA method was created the applications did not use relational database, so we used the Entity Relationship Diagrams (ERD) to identify the dependencies of the tables and identify the ILFs and EIFs. From the data model developed by Team A were identified 27 ILFs, but as the application developed did not use all these files, we recorded only the used files and that number decreased to 17. The application developed by Team A has no EIF because there is no communication with external databases. From the data model developed by Team B were identified firstly 16 ILFs, but for the same reason that Team A this number decreased to 12. For the same reason mentioned for the application of Team A, the application of Team B also has no EIFs. After being identified all ILFs and EIFs was necessary to identify the complexity of the applications, for this purpose we used as support the ISO/IEC 20926. Table 6.1 shows the ILF and EIF complexity.

Table 6.1: ILF and EIF Complexity

RETs	**Data Elements**		
	1-19	20-50	>=51
1	Low	Low	Average
2-5	Low	Average	High
>=6	Average	High	High

The "Data Elements" correspond to tables and *Record Element Type* (RET) correspond to attributes.

iv. **Count Transactional Functions** - In this step was counted the functions identified in the applications developed by Team A and B. After an analysis of the software developed by Team A were identified 29 EIs, in these functions were found functionalities for insertion, updating and delete data. As the EQs were identified 12 queries. The application of Team A has no EOs since there is no use of mathematical calculations (for example the sum of all medicines from a certain laboratory). From the analysis of the software developed by Team B were identified 19 EIs, as for the Team A in these functions were found functionalities for insertion, updating and delete data. As the EQs were identified 17 queries. Unlike Team A, the application of the Team B has EOs, two in total, for example, its application has implemented a counting function of all medicines available in a lot of drugs. This is a kind of function that requires a mathematical calculation making it an EO. After being identified all EIs, EQs and EOs was necessary to identify the complexity of the applications, for this purpose we used as support the ISO/IEC 20926. Table 6.2 shows the various intervals and the respective complexity associated to the EIs. On the other hand, Table 6.3 shows the number of intervals and the complexity associated to the respective EQs and EOs.

Table 6.2: EI Complexity

FTRs	**Data Elements**		
	1-4	5-15	>=16
0-1	Low	Low	Average
2	Low	Average	High
>=3	Average	High	High

Table 6.3: EQ and EO Complexity

FTRs	**Data Elements**		
	1-5	6-19	>=20
0-1	Low	Low	Average
2-3	Low	Average	High
>=4	Average	High	High

Where *File Types Referenced* (FTRs) are the ILFs combined number referenced or updated and EIFs referenced.

v. **Determine Unadjusted Function Point Count** - In this step were calculated the unadjusted function points. This step depends on the previous steps because the values identified above will be used at this step. Table 6.4 shows the count of all ILFs, EIFs, EOs, EQs and EIs and their respective complexity. The last row of Table 6.4 shows the sum of all UFPs.

Table 6.4: Total of Unadjusted Function Points

		Team A			Team B		
		Nr.	**Weight**	**UFPs**	**Nr.**	**Weight**	**UFPs**
ILF	Low	15	7	105	7	7	49
	Average	2	10	20	5	10	50
	High	0	15	0	0	15	0
EIF	Low	0	5	0	0	5	0
	Average	0	7	0	0	7	0
	High	0	10	0	0	10	0
EI	Low	23	3	69	17	3	51
	Average	6	4	24	2	4	8
	High	0	6	0	0	6	0
EQ	Low	9	3	27	13	3	39
	Average	3	4	12	3	4	12
	High	0	6	0	1	6	6
EO	Low	0	4	0	2	4	8
	Average	0	5	0	0	5	0
	High	0	7	0	0	7	0
		Total of UFPs		**257**			**223**

vi. **Determine Value Adjustment Factor** - After completing the previous step we must calculate *Value Adjustment Factor* (VAF).The VAF is based on 14 *General System Characteristics* (GSCs) that rate the general functionality of the application being counted. Associated with each of these characteristics should be attributed the *Degree of Influence* (DI). Table 6.5 shows the values of the DI attributed and a brief justification for the allocation of these values.

Substituting the value of DI in equation 6.1 we get the VAF. This value will be used in the next step.

$$\mathrm{VAF} = 0.65 + 0.01 * \mathrm{DI} \tag{6.1}$$

Table 6.5: GSC and Degree of Influence

	Team A		Team B	
Data Communications	0	The system was only used in stand-alone mode	0	The system was only used in stand-alone mode
Distributed Data Processing	0	Not applied	0	Not applied
Performance	0	Not tested	0	Not tested
Heavily Used Configuration	0	Without configurations	1	It was necessary to make Apache configurations
Transaction Rate	0	The system was not implemented	0	The system was not implemented
Online Data Entry	5	The system is a web site	5	The system is a web site
End-User Efficiency	2	Allows interaction with the user through an interface	1	Allows interaction with the user through an interface
Online Update	0	The system does not perform	0	The system does not perform
Complex Processing	0	Does not perform mathematical calculations	1	Perform simple mathematical calculations
Reusability	1	Reusability of some Java and HTML code	3	Reusability of some Java and HTML code
Installation Ease	0	Not tested	0	Not tested
Operational Ease	0	Not tested	0	Not tested
Multiple Sites	0	The system was not implemented on the client	0	The system was not implemented on the client
Facilitate Change	0	The system will not be changed	0	The system will not be changed
Total DI	8		11	
VAF	0.73		0.76	

vii. **Calculate Adjusted Function Point Count** - based on VAF calculated in the previous step we calculate the adjusted function points, usually just called Function Points (FPs), substituting that value in $\mathrm{FPs} = \mathrm{UFPs}{*}\mathrm{VAF}$ (6.2. Table 6.6 shows the calculation of the values of the FPs for Team A and Team B.

$$\mathrm{FPs} = \mathrm{UFPs}{*}\mathrm{VAF} \tag{6.2}$$

Table 6.6: Adjusted Function Points

Team A			Team B		
UFPs	VAT	FPs	UFPs	VAT	FPs
257	0.73	187.61	223	0.76	169.48

After knowing the FPs of the applications of the teams A and B, it is possible to calculate the productivity of each one. To calculate the productivity value was necessary to identify the hours of work needed to develop the applications studied. These data were obtained in the Master's thesis of Mandjam (Mandjam, 2011). The Team A developed the software application in about 3942 hours and Team B in 2435 hours. These number of hours include all work performed by all students in different roles. The productivity calculation is done using the following formula:

$$\text{Productivity} = \text{Effort[h]} \,/\, \text{Size[FPs]} \tag{6.3}$$

Applying the equation $\text{Productivity} = \text{Effort[h]} \,/\, \text{Size[FPs]}$ (6.3) we obtain a productivity of 21.01 Hours of Work/Function Points for Team A and 14.37 Hours of Work/Function Points for Team B. Based on these results we conclude that Team B was more productive than the Team A. The Team B required fewer work hours to perform one FP. The productivity of the Team B was higher by 40.6%. Table 6.7 shows the working hours, FPs and productivity for Team A and Team B.

Table 6.7: Productivity of Team A and Team B

Team A			**Team B**		
Working hours	FPs	Productivity	Working hours	FPs	Productivity
3942	187.61	21.01	2435	169.48	14.37

However, the productivity of the Team B it is higher than productivity of the team A, the grade is lower. This discrepancy is justified by the fact that final grade calculation results from a plurality of weighting factors from three sources of information (Clark, 2005), such as the teachers traditional evaluation, client evaluation and the quality of the final product (business functionality).

Based on FPs values calculated from the software applications of the teams and the grade assigned to one team, we can estimate the grade of the other teams. Table 6.8 shows that based on the classification assigned to the Team A (15.8) and the FPs values previously calculated for both teams it was possible to calculate the classification of the Team B (14.3). We used a proportionality rule to calculate the classification of the Team B.

Table 6.8: Classification from FPs

	Team A	**Team B**
FPs	187.61	169.48
Classification Grade in scale [0;20]	15.8	(169.48*15.8)/187.61 = 14.3

6.4. Analysis of the Experimentation Framework

As mentioned before, the main objective of the framework is to provide a classification scheme for understanding and evaluating empirical studies in software engineering area in an educational context. However, the framework also has another purpose, it helps structuring experimental processes. In this chapter, we describe the demonstration case 2. This demonstration case reports an empirical study on the estimation of size and complexity of software applications with Function Points Analysis (Alves, Oliveira, et al., 2014). Like the empirical study described in the previous chapter, this study also, it was entirely performed in our experimental environment. All artifacts, students and teacher staff belong to this environment. Thus, it makes perfect sense to analyze this empirical study using our adapted experimentation framework.

The framework of experimentation, summarized in Figure 4.3, consists of four categories corresponding to phases of the experimentation process: 1) definition, 2) planning, 3) operation, and 4) interpretation. The following sections discuss each of these four phases. The classification of the empirical study for each part of each of the phases is presented below.

In the definition phase of the experimental process, we have six parts: 1) motivation, 2) object, 3) purpose, 4) perspective, 5) scope, and 6) end user. As motivation for this empirical study was to understand, assess and validate the FPA metric. First, it was necessary to understand the FPA metric, and then, it was necessary to assess and validate if this metric is appropriate to estimate the software size and complexity of software applications in educational context. The object of the empirical study was a metric, in this case the FPA metric. The purpose of the study was to evaluate the effectiveness of the FPA metric in educational context. The study examines the FPA metric for a given project type from the perspective of the project manager. In this empirical study, we can consider also, the perspective of the researcher. In fact, the author of this thesis also followed this study from a research perspective. In the empirical study, the scope is replicated project because it was analyzed two teams that performed one project. The project was the same for both teams. It was about a software project of medium complexity, using UML notation encompassed in an iterative and incremental software development process, in this case, the RUP reduced model. The end user of this empirical study is real-world-like because the environment simulates an industrial context, and the object of the study is a real effort estimation metric.

In the planning phase of the experimental process, we have three parts: 1) design, 2) criteria, and 3) measurement. In this empirical study the team's formation was self-organized. This study involved two teams composed of 14 students each one. The teams were free to choose their elements, although each of them played a well-defined role according to the reduced RUP model. The teams did not resort to statistical models (multivariate analysis, statistical models, parametric models, and non-parametric sampling) in their projects and the empirical study also did not use this type of analysis. From perspective of external validity, we classify de empirical study as reliability and from perspective of learning benefits, we classify as integrability with the course. In fact, all subjects in this empirical study belongs to course syllabus. In the measurement part of the planning phase, we intend to capture some aspects about metrics, data collection, type of research and level of measurement that our students used in their projects. In this empirical study, we classify the measurement part as a metric validation. In fact, one of the objectives of the study is to know the applicability of the FPA metric in an educational context. The require data was collected by an objective way.

In the operation phase of the experimental process, we have three parts: 1) preparation, 2) execution, and 3) analysis. In this empirical study, we classify as main sources of preparation, the literature study, and classes. During the execution of this empirical study, data collection is the main action to achieve the project goals. In terms of analysis of the data, we classify the study as using a quantitative method. In this study, no preliminary data analysis was performed.

In the interpretation phase of the experimental process, we have three parts: 1) interpretation context, 2) extrapolation, and 3) impact. In this empirical study, we classify the interpretation context as field of research. In our opinion it is the most suitable classification. The representativeness of the sampling analyzed in a study qualifies the extrapolation of the results to other environments. In this empirical study, we classify the sample representativeness as low. In fact, the projects where the data was collected had medium complexity. Each one of this two projects was performed by teams of 14 students. In terms of the impact, we classify this empirical study as visibility from academia point of view. This empirical study brings a new contribution to the academy because the findings showed that based on the FPs values and the classification assigned to one team, we can estimate the grades of the other teams. In fact, the assessment schema presented in this empirical study after assigning a reference grade we can calculate an estimate of the remaining grades using the FPs values.

Finally, it is important to mention that the framework of experimentation was used to discipline the experimentation process, both for the students involved in the projects and for the researcher during the development of this empirical study.

6.5. Conclusion

Empirical studies in software engineering have had a significant role in the evaluation of tools, techniques, methods, and technologies before they introduced in industrial software.

A large number of empirical studies reported in the literature used students as subjects instead professionals. This great participation is justified because the students are accessible, available, willing to participate, inexpensive and we can combine the learning objectives of the courses with the research objectives of the studies.

The two teams developed separately a software system (Web application) for a real client. From the effort estimation methods, we used the FPA to estimate the productivity of that two teams.

The process of identifying the various components for the calculation of FPs was a manual process. This count was obtained from an intensive study of the code of the applications developed by the two teams. The FPs counting process is not trivial, but its adoption is of great importance when we want to measure the software development. The FPs can support the project management activities, since we can only manage what we can measure. The productivity in software development is a good indicator for management.

In (Jones, 2007) we find that the average effort is 20 man-hours work per Function Point. In our case study, Team A presents a very close value (21.01), whereas Team B present a value most discrepant (14.37). This does not mean that they are super team but means that further research is needed to find the reason of this discrepancy. However, we believe that empirical studies involving students on these subjects are important for the scientific community and the industry.

In our demonstration case, the productivity of the Team B is higher than the productivity of the Team A. However, the grade is lower because the final grade calculation results from a plurality of weighting factors from three different sources, namely, the teachers traditional evaluation, the client evaluation and the quality of the final product. Based on the FPs values and the classification assigned to one team we can estimate the grades of the other teams. In fact, the teachers assigned a grade of 15.8 to Team A and a grade of 14 to Team B. In the assessment schema presented previously, after assigning a reference grade we can calculate

an estimate of the remaining grades using the FPs values. This method can be an asset, especially when many teams are involved and if it is possible to automate the entire process.

Considering also that productivity is directly related to the effort made and with the functional size per FP, we suggest for future editions of the SSD course unit that all teams use a development tool, for example Teamwork Project Manager (Teamwork Project Manager, 2021), in order to accurately determine the effective involved effort. It would be interesting to assess the influence of this tool in the team's performance. The determination of the productivity of each element according the role of the RUP that he/she assumes is also an interesting experiment.

Chapter 7

Demonstration Case 3 – Software Risks

7.1. Introduction

In this chapter, the third demonstration case developed to assess the contributions of this thesis is analyzed. The demonstration case was developed in an educational environment using graduate and undergraduate students as subjects. All subjects involved in this research are enrolled in the Information Systems Master's Degree of the University of Minho. The main objective of this study was to demonstrate the need for the continuity of studies about the risks presented in software development projects.

Over the past two decades, Information Technology (IT) had a tremendous impact on individuals, organization, and society. In fact, it is practically impossible to imagine the current world without the methods, tools, and facilities of IT.

Due to this phenomenon, it has generated a lot of interest in several researchers around the world, trying to identify the causes of failures in projects and what various factors that can lead to success (Papke-Shields, Beise, & Quan, 2010). According to (Galliers & Sutherland, 2003) IT has spread at a strong pace in organizations, although some of these continue to remain without them it.

With this, due to the speed of updating the technologies involved and also the methodologies inserted in the process addressed, more technologies are experienced and we can access more information in a year than our fathers throughout their entire lifetime, in the past (Miguel, 2002).

Subsequently, the work of (Pressman & Maxim, 2020) addresses the term "software crisis", indicating the absence of tools, methods and procedures with the maturity required for successful software development. These problems in software development, over the past few decades, have created difficulties in managing software projects (Pressman & Maxim, 2020). We can conclude that, it is clear the need to obtain a solid foundation of project management with more targeted attention to the risks they may have.

The study of risks in software development projects is crucial to its success, so there is a need to learn more about the errors practiced in the management of these projects, errors that are known to many managers and scholars of the area, but still continue to grow.

The *Standish Group* presents each year the CHAOS report, which shows the percentage of success or failure of IT projects (The Standish Group International, 2015). The Standish Group definition of project success is based on the triple constraint, which has been the standard for the Project Management Institute (PMI) for several years. Using the triple constraint, the *Standish Group* evaluated projects as *successful*, *challenged* or *failed*. Thus, this institution applies the following definitions:

- *Successful* – A successful project is one that met all three of the triple constraints: schedule, cost, and scope;
- *Challenged* – A challenged project meets two out of three constraints, for example, delivered on time and on budget but not with the desired scope;
- *Failed* – A failed project is one that is canceled before it is completed or completed but not used.

In this report (The Standish Group International, 2015), it is also intended to demonstrate the success and failure factors of these same projects. In the last CHAOS report, the top five factors found in successful projects are identified, namely, (i) user involvement; (ii) executive management support; (iii) clear statement of requirements; (iv) proper planning; and (v) realistic expectations. These factors should be put on a checklist for anyone considering an IT project, whether large or small. While risk rises with size and complexity, even simple projects can fail if the participants cannot follow these five principles. Other factors that lead to the success of the projects have also been identified, but research shows that when there is presence of these five factors the probability of success is greater.

By analyzing previous data from CHAOS report, it is possible to create Figure 7.1. It is noted that there is an increase in projects concluded successfully, but there is still a large rate of failed projects. There are studies of this nature that seek to perceive what is still done wrong in the projects. With our study, we intend to understand the level of similarity between the risks occurred in academic projects and the risks occurred and reported in industrial projects in the IT area.

Considering the strong component of Project Management that the course unit of Development of Computer Applications (DCA) provides, this unit has been selected for the comparative analysis of this study. This course unit fits in the 2nd year of the Integrated Masters Course in Engineering and Management of Information Systems of the University of Minho. The members of DCA teams (work groups) perform a software

development project for six months, having as a client, a partner company of the University of Minho. The teaching methodology followed in the DCA course unit is the Project-based Learning (PBL). More detailed information can be consulted in (Alves et al., 2018).

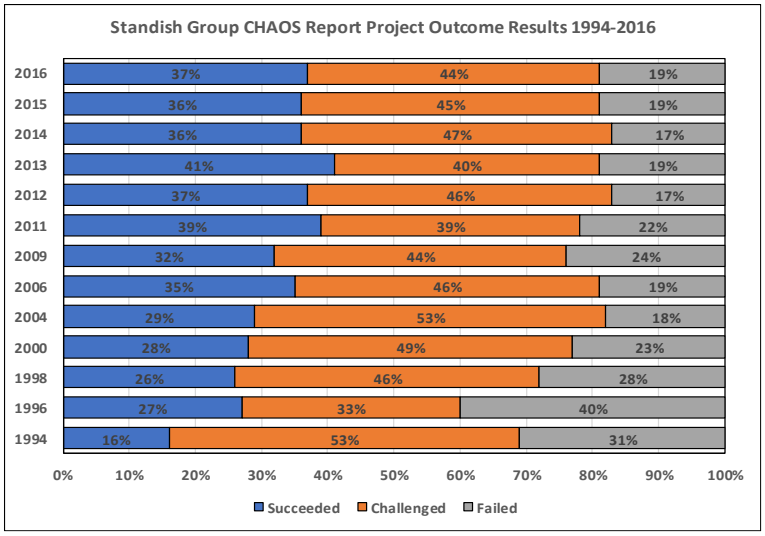


Figure 7.1: Standish Group CHAOS Report Project Outcome Results 1994-2016

The teams developed a software project of medium complexity, using the Unified Modeling Language (UML) notation encompassed in an iterative and incremental software development process, in this case, the Rational Unified Process (RUP). The teams followed the guidelines established by the RUP reduced model, executing the phases of inception, elaboration, and construction according to the best practices suggested by CMMI-DEV (Capability Maturity Model Integration for Development) v1.3 ML2 (Maturity Label 2). This software project was to develop a Web solution using object-oriented technologies (Java or C#) and relational databases (SQL Server or MySQL), to support the information system of one local company that provided all the information about the organization and interacted directly with the teams.

Based on software projects performed by the teams, this demonstration case presents a comparative analysis of the risks find in academic software projects and the risks find in industrial settings. Another goal is to know if exist or not a longevity of risks in software development projects (Alves et al., 2021).

7.2. Goals of the Demonstration Case 3

This demonstration case is not in the software metrics area as the two previous one, but our main purpose is the same. Thus, we have a main research question/problem that is *How to Conduct Empirical*

Software Engineering in educational context? As the two previous demonstrations cases, we perform this demonstration case to validate our environment in terms of adaptability, reliability, and practicability.

In addition to the previous question that is transversal to the entire thesis, in this demonstration case we try to answer more concrete questions within the area of risk management. Thus, in this study we want to know if the risks identified in the beginning of the 90's, when the propagation of software projects occurred, still remain current. Another important question is to know if the risks identified in the three selected studies are similar to the risks identified by the teams in an educational environment. We collected from the literature three sources of risks that seemed to us to be the most appropriate for the comparison.

In addition to the transversal objectives of the thesis, the study carried out in this demonstration case may contribute to the increase of knowledge in the area of risk management. As we will see in the following sections of this chapter, risk management is an area of great concern to companies. Neglecting this area could lead to catastrophic results.

7.3. Synopsis of Risks in Software Development Projects

For Ian Sommerville, the risk is the probability of any adverse situation happening. That author refers that the risk is a measure of the probability that the system will cause an accident. The risk is assessed by considering the hazard probability, the hazard severity, and the probability that the hazard will lead to an accident (Sommerville, 2016). McManus refers that during the development of a software project, there are many instances prone to adversity, no project is risk free (McManus, 2004). This happens because software projects are unpredictable and complex activities.

A risk possesses two characteristics that define it: uncertainty – whether it will happen or not, and loss – if it does happen, it can harm the probability of a project succeeding, partially or as a whole (Alencar & Schmit, 2012; Kendrick, 2015; Miguel, 2019; Pressman & Maxim, 2020). Thus, a risk is considered a negative event, that can happen in a project and can provoke unsatisfactory results (Barki, Rivard, & Talbot, 2001; Cherques, 2004).

According to (PMI, 2021), risks can be considered threats and/or opportunities in a project, given that some threats may come to be considered opportunities during the project. According to Charrete, risk is an event that can cause loss, delay, or damage to a software project (Charette, 2005). As mentioned, there are several approaches to the risks of a project, in our demonstration case we define risks as negative events with the possibility of occurring in projects and with the probability of an unsatisfactory result.

7.2.1. Literature Risks

For this research, we took into consideration risks from three different time periods. Starting with (Boehm, 1991), which presented the ten risks that project managers identified as occurring more often during software projects, those are:

- Personnel shortfalls;
- Unrealistic schedules and budgets;
- Development the wrong functions and properties;
- Development of wrong user interface;
- *Gold-plating* (inclusion of functionalities not solicited by the client);
- Continuing stream of requirements changes;
- Shortfalls in externally furnished components;
- Shortfalls in externally performed tasks;
- Real-time performance shortfalls;
- Straining computer-science capabilities.

Afterward, the research made by (Aloini, Dulmin, & Mininno, 2007) were analyzed, where a risk approach focused on ERP's (Enterprise Resource Planning) implementation projects was made, the most common risks according to these authors are:

- Poor project team skills;
- Low top management involvement;
- Ineffective communication system;
- Low key user involvement;
- Complex architecture and high number of implementation modules;
- Bad managerial conduction;
- Ineffective project management techniques;
- Inadequate change management;
- Ineffective consulting services experiences;
- Poor leadership;
- Ineffective strategic thinking and planning strategic;

In the research made by Júnior and Chaves (Júnior & Chaves, 2014), a synthesis of the main studies in the area is carried out and the risks that are still considered important by software project managers are collected, which are:

- Problems with technical artifacts by third-parties;
- Constant changing of the technical requirements;
- Poor development environment acquaintance;
- Technical issues with development;
- System test failure;
- Bad system development management;
- Delivery failure;
- Poor component conception;
- Lack of documentation;
- Incorrect interaction between organization and system processes;
- Poor system mapping.

These risks listed in the most recent studies refer to the development category included in the software project. It appears that some of the risks already mentioned by Boehm in (Boehm, 1991) still remain as important in more recent studies, demonstrating that with the available studies and advances made in the area of project management, many of the risks faced still remain as points to consider from current projects.

With this unpredictability and complexity that software development work requires, software projects are associated with various types of risks, which must be controlled using risk management (Han & Huang, 2007). The main objective of risk management being to increase the probability and impact of positive events and reduce the probability and impact of negative events on the project. An organization's ability to thrive by taking into account the presence of risk, as well as responding to unexpected events, good or bad, is the first indicator of that organization's ability to compete, and this is undoubtedly the first positive sign (Chapman, 2011).

Risk management is a method for treating risks, with the objective of minimizing or avoiding setbacks and their effects involved, which may affect the software development process. Previously, the area that dealt with risks was only part of a process, but with its evolution it started to act transversally in all software development processes (Dvir & Shenhar, 2010; Nogueira, 2009; Nogueira & Machado, 2012).

7.2.2. The Importance and Benefits of Risk Management

Although risk management is one of the greatest needs in project management, it is recognized that little has been done about it (Ibbs & Kwak, 2000; Raz, Shenhar, & Dvir, 2002; Zwikael & Globerson, 2006; Zwikael & Sadeh, 2007). As a software project progresses, there is an immense chance that something will not work as expected. In 2012, Kutsch *et al.* (Kutsch, Denyer, Hall, & Lee-Kelley, 2013), demonstrated that many project managers still neglect risk management in the course of their project and presented five key beliefs that can justify this attitude, namely:

- *Legitimacy*: managers believe that by following risk management procedures, they generate acceptance and trust among stakeholders, even if the risk management structure is announced without actually being in use;
- *Value*: they believe that risk management should be proven useful, and when there is no obvious value the managers interest in risk management decreases;
- *Competence*: they believe that by demonstrating to the customer that there is a risk that hinders the success of the project, this may jeopardize the competence of managers in front of customers;
- *Fact*: managers dissociate themselves from risk management when risks are considered fictitious or imaginary;
- *Authority*: managers fail to follow risk management when they felt they did not have the autonomy to act in mitigating risks.

With the application of risk management as a striking and disciplined part of the organizational environment it is possible to perceive various benefits. According to ISO/IEC 31010 (IEC 31010:2019, 2019), the main benefits when performing risk management are:

- Provide information to decision makers;
- Communicating risks and uncertainty;
- Assist in the establishment of priorities;
- Contribute to the prevention of incidents based on post-incident investigation;
- Meet regulatory requirements;
- Understand the risk and its potential impact on the objectives of the project.

According to Karolak, uncertainties in estimating project size, quality, and schedule are some of the possibilities to generate difficulties. In this sense, there are some risk management approaches, which are

constantly being improved, in order to provide a holistic view to the project manager about the risks that he may face in his project (Karolak, 1996).

7.2.3. Methodology Followed to Compare Risks

The methodological approach used was the case study. According to Fidel (Fidel, 1984), the case study aims to understand the event under analysis and at the same time develop more general theories about the observed event. Questionnaires, interviews, observation, analysis of artefacts or other methods may be carried out.

For the comparative analysis realized in this study, all the projects performed by the teams of the Development of Computer Applications (DCA) course unit in the period 2011/12 up to 2015/16 were analyzed. In this way, we created a database consisting of the risks identified and the main issues faced by the working groups. Twenty-nine works were considered during these five years. Each team (between 12 and 16 elements) produced a list of risks ordered by their degree of seriousness, from 1 to 25. The lists of risks, collected from the working groups, were classified, and organized according to the probability and impact of their occurrence and according to the consequences that these risks could have on the project.

Due to the strong project management component that DCA provides, it was selected as an object of study due to the numerous works already carried out with the risk management component, where students identify them throughout the course, which integrates a software project for a partner company of the University of Minho. Typically, the organizational structure of teams includes the following roles: project manager (1 member), quality coordinator (1 member), software architect (1 member), development coordinator (1 member), infrastructure coordinator (1 member), analysts (3 members) and programmers (6 members).

The DCA software project is divided into five stages of monitoring and evaluation. The first three-week evaluation period is called “project planning”, with delivery of a report. The second moment corresponds to the presentation and writing of the “functional specification of the solution and prototype 1” carried out in the sixth week of the project. The third moment corresponds to the presentation and writing of the “logical architecture and prototype 2” and is closed in the twelfth week of the project's progress. The fourth moment aims at the “individual practical exam in the informatic laboratory” taking place in the fifteenth week of the project's progress. The last moment aims at the "commercial presentation (client) and laboratory demonstration (professors) of the solution", where a final report is designed with all the development related

to the project, including all the points covered during its execution. This point occurs during the seventeenth and eighteenth week of the project, however the first presentation of the project occurs to the project's clients and only later to the teaching team, and only later the evaluation is assigned to the students.

In order to follow our approach, for the accomplishment of the treatment and analysis of the results, three steps were accomplished, namely:

1. Individualized analysis of risk exposure;
2. Comparative analysis between list of risks and problems faced;
3. Comparative analysis of the list of risks identified with the literature.

In order to identify the risks most frequently indicated by the working groups, an individual analysis of the risk exposure by impact was carried out. Then, the results obtained in the previous step were compared with the list of problems faced by the working groups during the execution of the projects. Finally, a comparative analysis of the list of risks identified with the risk found in the literature was performed. In the following section, we present with some detail the last step, where the studies were considered in three distinct periods for comparative effect. The first two steps were carried out in (Souza, 2016) and it was decided not to include them in this thesis.

7.2.4. Comparative Risk Analysis in Software Projects

After analyzing more than four hundred risks identified and about one hundred problems faced and documented by the working groups, it was possible to highlight the twenty risks mentioned in Table 7.1. This final list of risks shows the main risks identified over the last 5 years in the DCA course unit. We found that the risks are dispersed in different categories within the development of the project, because there were risks linked to the elements of the teams in a more individual way, specific risks in technical areas and risks related to the clients.

Among the risks presented in Table 7.1, there were two risks that drew attention during the study, whether due to the type in which they were framed or the way they were denoted out in the works analyzed. These risks were “Changes in requirements by the customer” and “Quality of project documentation and reports”.

About the first risk, the teams must be very attentive to this point because it is one of the main causes of delays and reformulation of task plans. They should always maintain a margin of safety for the planned

time and pay great attention to the first meetings to collect the requirements. If possible, they should use audio and video recording.

About the second risk, the teams demonstrated that they ended up missing project submission dates due to the lack of revisions and the need to revalidate some points of the project. This fact seems to demonstrate that the working groups had a need to redo reports instead of just correcting them. A rigorous analysis of the templates provided by the RUP (Rational Unified Process) is essential. These templates help the teams to create a project documentation and reports with good quality.

Table 7.1: Final risk list

Risk
Delay or non-fulfillment of dates on delivery of artifacts
Lack of effort and commitment of the team members to the project
Quality of project documentation and reports
Workload/hours for some team members
Communications difficulty between team members
Loss of team members
Shortage of time and resources
Lack of knowledge of the tools being used
Inexperience of team members
Changes in requirements by the customer
Complexity of the system functionalities used in the project
Difficulty in communicating and gathering customer requirements
Difficulty in managing subcontracting
Difficulty in managing the evaluations of other unit courses
Problems with software production
Poor knowledge of the business area
Poor quality of system architecture
Failure in artifact planning
Failure in modeling requested requirements
Lack of adequate space for work and meetings

Next, the reasons that weighed in the choice of studies on risks found in the literature are presented. The Boehm study (Boehm, 1991), considered one of the first carried out in the area and where the ten most common risks in software development projects are presented. The study by Aloini, *et al.* (Aloini et al., 2007) for carrying out a review of the literature on risk management in project planning in the area of ERP. Finally,

a study by Júnior and Chaves (Júnior & Chaves, 2014) was used, which identified new risks for the management of information technology projects through an exploratory survey with project managers. In order to compare the risks identified in previous studies in the industry with the risks identified by DCA teams in an educational environment, we intend to understand the degree of applicability and correctness between the different environments.

In Table 7.2 it is possible to visualize the comparisons made of all the studies considered with the data collected within the academic scope in the last five years. Some risks remained present in all studies, such as non-compliance with deliveries, the various changes in requirements, the difficulty of managing third-parties' tasks and the difficulty of properly planning the schedule. While others did not present direct relationships with risks raised in the academy. Thus, it is worth noting that the fact that all risks are not directly correlated may be due to the fact that some risks have a more specific essence while others end up having a more comprehensive definition nature.

In addition, this research was carried out in order to propose a broader approach, among the studies addressed, some are directed to the ERP area and others focused on development, thus making it possible to perform simultaneous combinations between all studies. It is also important to note that all the risks listed in the academy's working groups were identified by the project manager when he became responsible for identifying the risks.

Thus, two risks call attention, the first named as "Poor leadership" may not have been mentioned in the course unit studies because the project manager is responsible for identifying the risks. Certainly, he/she never point to poor leadership as a risk to consider in the project. The second one is the risk of "gold-plating", identified by Boehm (Boehm, 1991). This risk is defined by some researchers as an artifice that managers use to circumvent crises with customers, but that end up generating unnecessary costs to the projects. Because of this, in (Chowdhury & Arefeen, 2011), "gold-plating" was cited by the Computer Emergency Response Team (CERT) as one of the risks that most generate vulnerabilities in the software applications. The "gold-plating" risk has to do with the inclusion of functionalities that were not requested by the customer in the requirements gathering. This risk may not have arisen in any of the works analyzed because they belong to the educational environment and in addition, we should consider the lack of experience of many elements of the working groups with the resolution of problems of this nature and magnitude.

Table 7.2: Comparison of existing literature with the risks identified in DCA

DCA (2011-2016)	Boehm (1991)	Aloini, Dulmin and Mininno (2007)	Junior and Chaves (2014)
Delay or non-fulfillment of dates on delivery of artifacts	Real-time performance shortfalls	Ineffective strategic thinking and planning strategic	Delivery failure
Lack of effort and commitment of the team members to the project		Low top management involvement	
Quality of project documentation and reports			Lack of documentation
Workload/hours for some team members			
Communications difficulty between team members		Ineffective communication system	
Loss of team members			
Shortage of time and resources			
Lack of knowledge of the tools being used			Technical issues with development (hardware)
Inexperience of team members	Personnel shortfalls	Poor project team skills	
Changes in requirements by the customer	Continuing stream of requirements changes	Inadequate change management	Constant changing technical requisites
Complexity of the system functionalities used in the project		Complex architecture and high number of implementation modules	
Difficulty in communicating and gathering customer requirements		Low key user involvement	Incorrect interaction between organization and system processes
Difficulty in managing subcontracting	Shortfalls in externally performed tasks	Ineffective consulting services experiences	Problems with technical artifacts by third-parties
Difficulty in managing the evaluations of other unit courses			
Problems with software production	Development the wrong functions and properties		Poor component conception
Poor knowledge of the business area			Technical issues with development (business area)
Poor quality of system architecture			System test failure
Failure in artifact planning	Unrealistic schedules and budgets	Ineffective project management techniques	Poor system mapping
Failure in modeling requested requirements	Development of wrong user interface		Bad system development management
Lack of adequate space for work and meetings	Shortfalls in externally furnished components		
	Gold-plating		
	Straining computer-science capabilities		
		Poor leadership	

With the analysis carried out along the comparisons made between the studies, it was possible to obtain the risks that presented greater visibility by the managers of software development projects in projects developed in an industrial setting and in an educational environment.

7.4. Analysis of the Experimentation Framework

As mentioned before, the main objective of the framework is to provide a classification scheme for understanding and evaluating empirical studies in software engineering area in an educational context. However, the framework also has another purpose, it helps structuring experimental processes. In this chapter, we describe the demonstration case 3. This demonstration case reports a study to demonstrate the need for the continuity of studies about the risks presented in software development projects. For this purpose, we analyzed more than four hundred risks and about one hundred problems faced and documented by the working groups. With the collected data we defined a list of twenty risks and conducted a comparative study of these risks with others already formalized in previous industrial studies (Alves et al., 2021). As the previews empirical studies, this study was developed in an academic environment. All subjects involved in this research are enrolled in the Information Systems Master's Degree of the University of Minho.

The framework of experimentation, summarized in Figure 4.3, consists of four categories corresponding to phases of the experimentation process: 1) definition, 2) planning, 3) operation, and 4) interpretation. The following sections discuss each of these four phases. The classification of the empirical study for each part of each of the phases is presented below.

In the definition phase of the experimental process, we have six parts: 1) motivation, 2) object, 3) purpose, 4) perspective, 5) scope, and 6) end user. As motivation for this study was to understand and to assess if the risks founded in the software development projects performed in educational context are similar to risks founded in industrial projects. The object of the study was a process, in this case the process of finding the list of risks. The purpose of the study was to characterize the risks and the main issues faced by the working groups in their projects. This task allowed to create a list with the most common risks and later to compare this list against the list of risks identified in the literature. The study examines the list of risks from the perspective of the project manager. In this study, we can consider also, the perspective of the researcher. In fact, the author of this thesis also followed this study from a research perspective. In this study, the scope is replicated project because it was analyzed, for each school year, several teams that performed the same project. The end user of this empirical study is real-world-like because the environment simulates an industrial context, and there was a concern to compare the list of risks found in academic projects and in industrial projects.

In the planning phase of the experimental process, we have three parts: 1) design, 2) criteria, and 3) measurement. In this empirical study the team's formation was self-organized. This study analyzes the risks founded in projects developed by Development of Computer Applications (DCA) students during five school years. The teams were free to choose their members, although each of them played a well-defined role according to the reduced RUP model. The teams did not resort to statistical models (multivariate analysis, statistical models, parametric models, and non-parametric sampling) in their projects and the empirical study also did not use this type of analysis. From perspective of external validity, we classify the empirical study as transferability and from perspective of learning benefits, we classify as integrability with the course. In fact, all subjects in this empirical study belongs to course syllabus. In the measurement part of the planning phase, we intend to capture some aspects about metrics, data collection, type of research and level of measurement that our students used in their projects. In this study, we classify the measurement part as data collection because it was necessary to analyze more than four hundred risks and about one hundred problems faced and documented by the working groups. The require data was analyzed by a subjective way.

In the operation phase of the experimental process, we have three parts: 1) preparation, 2) execution, and 3) analysis. In this study, we classify as main sources of preparation, the literature study, classes, and analysis of similar projects. During the execution of this study, data collection is the main action to achieve the project goals. In terms of analysis of the data, we classify the study as using a qualitative method. In this study, no preliminary data analysis was performed.

In the interpretation phase of the experimental process, we have three parts: 1) interpretation context, 2) extrapolation, and 3) impact. In this study, we classify the interpretation context as study purpose. In this study, there was a well-defined objective, which was to understand the longevity of risks in software development projects. In our opinion it is the most suitable classification. The representativeness of the sampling analyzed in a study qualifies the extrapolation of the results to other environments. In this empirical study, we classify the sample representativeness as high. In fact, the projects where the data was collected had medium and high complexity. The data was collected from several projects in five school years. In terms of the impact, we classify this empirical study as visibility from academia point of view. A replication of this study to validate the final list of risks generated in other academic projects in other universities and then in other software projects in real companies could be interesting.

Finally, it is important to mention that the framework of experimentation was used to discipline the experimentation process, both for the students involved in the software development projects and for the researcher during the development of this study.

7.5. Conclusion

In this study it was possible to conclude that in general, over the five school years analyzed, the academic teams have conducted a good risk assessment. One of the reasons for this, it is certainly the fact that teams follow current and rigorous software development practices. The use of RUP in conjunction with CMMI-DEV allows teams to follow development processes close to the references used in the industry.

In carrying out this study, the reports of all DCA teams since 2011 were analyzed. Despite the high amount of work analyzed, it was possible to see some alignment in the works presented by the teams at moment five (where the data were extracted for analysis). Thus, it was possible to perceive a likelihood during the individualized analysis of the risk exposure, which allowed the formation of a uniform list of risks with medium and high exposure.

With the correlation of the risks defined in the individualized analysis with the problems reported by the teams, it was possible to conclude that most of the teams carried out a good risk assessment. It should be noted that there will certainly be some transfer of know-how from one year to another, with some adjustments being made based on the reported problems.

Also, according to the data gathered in this study, it was possible to notice that the risks identified in the beginning of the 90's, when the propagation of software projects occurred, still remain current. The risks identified in the three selected studies are similar to the risks identified by the teams in an educational environment, although some risks are more specific in terms of granularity.

It is important to point out that to achieve effective risk management is necessary to implement a methodology from project planning to its closure. This list of risks for academic projects presented, should serve as support for the implementation of a good risk management strategy and for the greater success of the project in relation to all stakeholders.

As a suggestion for future work would be suitable to perform a practical validation of the final list of risks generated in other academic projects in other universities and then in other software projects in real companies. In addition, it would be interesting to study the positive aspect of risks (opportunities) in the academic and non-academic fields, since during the analysis of this study this approach was not considered.

It would also be important to apply this study to projects of a different nature, regardless of whether the product is or not a software application, since the vast majority of risks are transversal to different areas. These recommendations are beyond the work of this thesis.

Chapter 8
Conclusions

8.1. Research Contributions

This chapter is the conclusion of this work. It contextualizes the work in the theme of empirical software engineering in educational context. It then synthesizes the research efforts, the results that express the contributions of the thesis, and suggests possible research tracks for future work.

Large amounts of money are being consumed by software costs, yet problems still exist with software quality and delivering software on schedule and within the development budget. This is partly a result of the absence of measurement programs and partly a result of software developers using software technology which not been evaluated. As consequence, conducting empirical evaluation has started to become a more important part of software engineering research. In the last two decades of the 20th century, the methodology used for empirical software engineering research was immature and underdeveloped. However, in recent decades, several authors and institutions have dedicated themselves to improving this methodology.

Although software engineering is a recent area of knowledge, it has benefited from substantial improvements, due in large part to the work carried out at the experimentation level. The development of specific infrastructures for experimentation and the improvement of the research methodology followed has contributed to this improvement. This has several implications, first, some of the empirical research that has been conducted presents relevant results due to the empirical methodology used. Second, empirical results start to be accepted by software engineering community. Third, the transfer of knowledge between academia and industry has been more fruitful.

Conducting experiments in software engineering area in an industrial context is extremely difficult for several reasons. Companies must deliver results and they are pressured by costs, timing, and quality. Companies cannot release their employees to do experiments. Being aware of this situation, this thesis presents an experimental environment in an educational context. Our main goal was to create a platform that

allows us to perform empirical evaluations of the tools, techniques, methods, models, and technologies used in software engineering.

The work performed and described in this document was achieved through the development of the following major activities stated in the following paragraphs.

In the beginning of the thesis, it was a presentation of an introductory context of the scientific background involved, the methodological context, the goals and the research strategy, and the structure of this document. After this introduction, the several efforts carried out were described.

It was performed a research on the state-of-art in the area of empirical software engineering. This allowed to find the origins, concepts, characteristics, and research in the empirical software engineering area. It was presented the main empirical methods collected from the literature. Different taxonomies to classify experimental studies were presented. All phases of experimentation process in software engineering area were presented with some detail. Also, the main approaches related to the experimentation process were presented. The importance of conducting empirical studies in Software Engineering was revealed. In fact, these studies are needed to develop or improve processes, methods, and tools for software development and maintenance. Some concerns are raised regarding the technology transfer from the software engineering research community to industry. All this background allowed to acquire a deeper knowledge of the research area of this thesis.

An exhaustive literature review was carried out on three important subjects: software development models, project management approaches and software metrics. As is known, there are several links between these areas of knowledge. According to the literature we classified the software processes models in: (1) plan-driven and (2) iterative and change-driven. From a critical perspective, a chronological evolution of software development models was presented. The main characteristics of agile software development were presented, as well as its main methodologies and techniques. Whenever appropriate, a comparison was made between agile software development and traditional process models. The concepts and the project management approaches were presented. PRINCE2 and PMBOK project management references were described in some detail. The main metrics were presented, but special emphasis was given to the UCP and FPs metrics. For these metrics, in a chronological way, the main research produced by the scientific community were presented.

An overview of the Empirical Software Engineering (ESE) in teaching was presented. The main issues about the ESE in educational context were revealed. Based on scientific documents of relevant authors in this

area, the advantages, and disadvantages of using students and professionals in empirical studies were presented. It was presented the main research institutions in the software engineering area that involve students in their empirical studies. The Basili *et al.* framework of experimentation was presented in detail. This framework was adapted by the author of this thesis to classify and guide the empirical studies performed in educational context.

To validate the research efforts of the thesis a set of demonstration cases were performed and analyzed. This thesis provides several contributions. Among these contributions are:

- **framework for classifying empirical studies in educational context**: As in all areas of science and engineering, empirical research can only be considered rigorous when it is conducted using a valid experimental approach or protocol. This thesis presents an adapted framework to classify empirical studies performed in an educational context. This framework was adapted from Basili *et al.* framework of experimentation. The new framework provides a classification scheme for understanding and evaluating empirical studies in software engineering area in an educational context. With this framework we can analyze most of the experimental work that has been done in our computing courses. The framework also has another purpose, it helps structuring experimental processes. In order to validate our adapted framework, a demonstration case was performed with project works elaborated by students from the University of Minho. Thus, we collected all these projects carried out in the PMIS unit course in the period of 2010/2011 up to 2017/2018. In this period, 105 students enrolled in the PMIS course developed 79 projects. All these projects were classified using our framework. Based on this research, we can conclude that our adapted framework is applicable and appropriate to classify project works of the PMIS course unit.
- **Experimental Environment**: Our approach involves students from BSc, MSc, and PhD degrees in Computing from University of Minho that develop a complete software project requested by a real client. The educational approach is mainly based on PBL principles. With our approach, the teaching staff is responsible for creating an environment that enhances communications, team working, management and engineering skills in the students involved. The curriculum integration and the pedagogical cooperation, through an integrated project between the four courses units are intended to promote students to work in a software development environment that is similar to an organization environment. In our setting we promote a win-win approach for all stakeholders: clients, students, teachers, and researchers. This environment allowed to PhD students engage with the

students of the four courses units, in order to carry out scientific studies supported by educational context experiments and this work has given rise to international publications. A mapping between the SWEBOK Knowledge Areas (KA), and the work projects developed by the ADIS and PMIS students in the academic years 2010/2011 and 2011/2012 was performed. In this two academic years, 140 students developed 130 work projects in different SWEBOK KAs.

- **Use Case Points Analysis**: This method was applied to estimate the size and effort of the software projects realized by the teams in educational context. We found that the productivity of the teams varies between 5.5 hours to 12.1 hours per UCP. The demonstration case that apply the UCP metric was analyzed by the experimentation framework.
- **Function Points**: the original FPA method was applied for estimate the size and complexity of the software system developed by two teams. Based on FPs of each team it was possible to calculate their productivity. The demonstration case that apply the FPA metric was analyzed by the experimentation framework.
- **Software Risks**: In terms of risk assessment, the reports of the software projects of the DCA teams during five school years were analyzed. The teams carried out a good risk assessment. The risks identified in the three selected studies are similar to the risks identified by the teams in an educational environment, although some risks are more specific in terms of granularity. The demonstration case to evaluate the software risks was analyzed by the experimentation framework.

During this thesis, some presentations and publications were made. The doctoral proposal was presented in the Symposium for PhD students in Software Engineering, SEDES'2012, IEEE CS Press, and the publications made were:

- Luís M. Alves, Gustavo Souza, Pedro Ribeiro, Ricardo J. Machado. "Longevity of risks in software development projects: a comparative analysis with an academic environment", in Procedia Computer Science, Vol. 181, pp 827-834, [DOI: 10.1016/j.procs.2021.01.236], [ISSN:1877-0509], http://hdl.handle.net/10198/23621, (2021).

- Luís M. Alves, Pedro Ribeiro, Ricardo J. Machado. "Classifying Empirical Studies in an Educational Context Through an Experimentation Framework", in 12th annual International Conference on Education and New Learning Technologies (EDULEARN20), Palma de Maiorca, [ISBN: 978-84-09-17979-4] (2020).

- Luís M. Alves, Pedro Ribeiro, and Ricardo J. Machado, "Project-Based Learning: An Environment to Prepare IT Students for an Industry Career", Information Resources Management Association (IRMA), USA (Ed.), Methodologies, Tools, and Applications, Chap. 80, pp. 1931-1951, IGI Global, Hershey, USA, 2018 [ISBN 978-1-5225-3923-0]. http://hdl.handle.net/10198/20523

- Luís M. Alves, Pedro Ribeiro, and Ricardo J. Machado, "Architectural Element Points: Estimating Software Development Effort by Analysis of Logical Architectures". S. Wrycza (Ed.), Lecture Notes in Business Information Processing, Chap. 5, pp. 72-84, Springer International Publishing, Gdansk, Poland, 2016 [ISBN 978-331946641-5]. http://hdl.handle.net/10198/13765

- Luís M. Alves, Pedro Ribeiro, and Ricardo J. Machado, "Project-Based Learning: An Environment to Prepare IT Students for an Industry Career", Liguo Yu (Ed.), Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills, Chap. 12, pp. 230-249, IGI Global, Hershey, USA, 2014 [ISBN 978-1-4666-5800-4]. http://hdl.handle.net/10198/9861

- Luís M. Alves, Sérgio Oliveira, Pedro Ribeiro, Ricardo J. Machado, "An Empirical Study on the Estimation of Size and Complexityof Software Applications with Function Points Analysis", in 14th International Conference on Computational Science and Its Applications – ICCSA 2014 , pp 27-34, IEEE Computer Society Press, Los Alamitos, California, USA, [ISBN: 978-1-4799-4264-0] (2014).

- Luís M. Alves, André Sousa, Pedro Ribeiro, Ricardo J. Machado, "An Empirical Study on the Estimation of Software Development Effort with Use Case Points", inProceedings of the 43rd annual Frontiers inEducation Conference – FIE 2013, pp 101-107, IEEE Computer Society Press, Los Alamitos, California, USA, [ISBN: 978-1-4673-5261-1] (2013).

- Luís M. Alves, Ricardo J. Machado, Pedro Ribeiro, "Experimental Software Engineering in Educational Context", in Proceedings of the 8th International Conference on the Quality of Information and Communications Technology - QUATIC'2012, Session on SEDES'2012 Workshop, pp 336-341, IEEE Computer Society Press, Los Alamitos, California, USA, [ISBN: 978-0-7695-4777-0] (2012).

Additionally, it is expected further publications from this dissertation related to the results and conclusion of the demonstration cases analysis.

8.2. Future Work

The empirical software engineering area has many research topics and challenges. In this thesis we deal with empirical software engineering in educational context. Additional research tracks and efforts might be considered for those that would like to use this PhD document as a baseline for future work, namely:

- Apply the experimentation framework in other course units of other universities;
- Adapt the experimentation framework to other scientific areas;
- Compare our experimental environment with others from other universities;
- Develop an efficient mechanism to transfer knowledge obtained in experiments from academia to industry;
- Create greater synergies with software development companies to carry out experiments.

References

Abels, S., Ahlemann, F., Hahn, A., Hausmann, K., & Strickmann, J. (2006). PROMONT - a Project Management Ontology as a Reference for Virtual Project Organizations. In R. Meersman, Z. Tari, & P. Herrero (Eds.), *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops* (Vol. 4277, pp. 813-823). Berlin: Springer-Verlag Berlin.

Abrahamsson, P. (2001). Rethinking the Concept of Commitment in Software Process Improvement. *Scandinavian Journal of Information Systems, 13*(1), 69-98.

Abrahamsson, P. (2003). *Extreme Programming: First Results from a Controlled Case Study*. Paper presented at the The Proceedings of 29th Euromicro Conference (EUROMICRO' 03), Belek-Antalya, Turkey.

Abrahamsson, P. (2010, March 2010). Striving for Multi-Disciplinary Research. *Software Factory Magazine, 1,* 5.

Abrahamsson, P., Oza, N., & Siponen, M. T. (2010). Agile Software Development Methods: A Comparative Review. In T. Dingsøyr, T. Dybå, & N. B. Moe (Eds.), *Agile Software Development: Current Research and Future Directions* (pp. 31-59). Berlin, Heidelberg: Springer Berlin Heidelberg.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile Software Development Methods: Review and Analysis*. Oulu, Finland: VTT Publications 478.

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003, 3-10 May 2003). *New Directions on Agile Methods: a Comparative Analysis.* Paper presented at the 25th International Conference on Software Engineering (ICSE'03), Portland, OR, USA.

Abrahão, S., & Poels, G. (2007). Experimental Evaluation of an Object-oriented Function Point Measurement Procedure. *Information and Software Technology, 49*(4), 366-380. doi:http://dx.doi.org/10.1016/j.infsof.2006.06.001

Abran, A. (2010). *Software Metrics and Software Metrology* (1st ed.). New Jersey, USA: Wiley-IEEE Computer Society.

Abran, A., Bourque, P., Dupuis, R., Moore, J. W., & Tripp, L. (2004). *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, California, USA: IEEE Computer Society Press.

Abran, A., & Robillard, P. N. (1996). Function Points Analysis: an Empirical Study of its Measurement Processes. *IEEE Transactions on Software Engineering, 22*(12), 895-910. doi:10.1109/32.553638

ACM-Curricula-IS. (2010). *IS 2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems*. Retrieved from https://www.acm.org/binaries/content/assets/education/curricula-recommendations/is-2010-acm-final.pdf

ACM-Curricula-MSIS. (2006). *Model Curriculum and Guidelines Graduate Degree Programs in Information Systems*. Retrieved from https://www.acm.org/binaries/content/assets/education/curricula-recommendations/msis-2006.pdf

Adrion, W. R. (1993). Research Methodology in Software Engineering: Summary of the Dagstuhl Workshop on Future Directions in Software Engineering. *SIGSOFT Softw. Eng. Notes, ACM Press, New York, 18*(1), 36-37.

Agile Alliance. (2001). Manifesto for Agile Software Development. Retrieved 2022-05-12 from https://agilemanifesto.org/

Agtriadi, H. B., Chandra, N., Warnars, H. L. H. S., & Gaol, F. L. (2017, Nov. 20-22). *Software Size Measurement with Use Case Point for Employee Application Software at STT-PLN.* Paper presented at the 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), Phuket, Thailand.

Albrecht, A. J. (1979, October 15-17). *Measuring Application Development Productivity.* Paper presented at the IBM Application Development Symposium, Monterey, California, USA.

Albrecht, A. J., & Gaffney, J. E., Jr. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering, SE-9*(6), 639-648. doi:10.1109/tse.1983.235271

Alencar, A. J., & Schmit, E. A. (2012). *Análise de Risco em Gerência de Projetos* (3rd Ed.). Rio de Janeiro, Brasil: Brasport Livros e Multimídia Ltda.

Ali, M. R. (2006). Imparting Effective Software Engineering Education. *ACM SIGSOFT Software Engineering Notes, 31*(4), 1-3. doi:10.1145/1142958.1142960

Aloini, D., Dulmin, R., & Mininno, V. (2007). Risk Management in ERP Project Introduction: Review of the Literature. *Information & Management, 44*(6), 547-567. doi:https://doi.org/10.1016/j.im.2007.05.004

Alves, L. M., Machado, R. J., & Ribeiro, P. (2012, September 3-6). *Experimental Software Engineering in Educational Context.* Paper presented at the 2012 Eighth International Conference on the Quality of Information and Communications Technology (QUATIC 2012), Lisboa, Portugal.

Alves, L. M., Oliveira, S., Ribeiro, P., & Machado, R. J. (2014). *An Empirical Study on the Estimation of Size and Complexity of Software Applications with Function Points Analysis*. Paper presented at the 6th International Workshop on "Tools and Techniques in Software Development Process" (TTSDP'14) in conjunction with The 2014 International Conference on Computational Science and Its Applications (ICCSA 2014), Guimarães, Portugal.

Alves, L. M., Ribeiro, P., & Machado, R. J. (2014). Project-Based Learning: An Environment to Prepare IT Students for an Industry Career. In *Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills* (pp. 230-249). Hershey, PA, USA: IGI Global.

Alves, L. M., Ribeiro, P., & Machado, R. J. (2016). Architectural Element Points: Estimating Software Development Effort by Analysis of Logical Architectures. In S. Wrycza (Ed.), *Information Systems: Development, Research, Applications, Education: 9th SIGSAND/PLAIS EuroSymposium 2016, Gdansk, Poland, September 29, 2016, Proceedings* (pp. 72-84). Cham, Switzerland: Springer International Publishing.

Alves, L. M., Ribeiro, P., & Machado, R. J. (2018). Project-Based Learning: An Environment to Prepare IT Students for an Industry Career. In *Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications* (pp. 1931-1951). Hershey, PA, USA: IGI Global.

Alves, L. M., Ribeiro, P., & Machado, R. J. (2020, July 5-6). *Classifying Empirical Studies in an Educational Context Through an Experimentation Framework.* Paper presented at the 12th annual International Conference on Education and New Learning Technologies (EDULEARN20), Palma de Maiorca (Virtual Conference).

Alves, L. M., Sousa, A., Ribeiro, P., & Machado, R. J. (2013, October 23-26). *An Empirical Study on the Estimation of Software Development Effort with Use Case Points.* Paper presented at the 2013 Frontiers in Education Conference, Oklahoma City, Oklahoma, USA.

Alves, L. M., Souza, G., Ribeiro, P., & Machado, R. J. (2021). Longevity of Risks in Software Development Projects: a Comparative Analysis with an Academic Environment. *Procedia Computer Science, 181*, 827-834. doi:https://doi.org/10.1016/j.procs.2021.01.236

Amaral, E. A. G. G. (2006). *Empacotamento de Experimentos em Engenharia de Software.* (MSc). Universidade Federal do Rio de Janeiro, Brasil,

Anda, B. (2002). *Comparing Effort Estimates Based on Use Case Points with Expert Estimates*. Paper presented at the Empirical Assessment in Software Engineering (EASE 2002), Keele, UK.

Anda, B., Benestad, H. C., & Hove, S. E. (2005, Nov. 17-18). *A Multiple-case Study of Software Effort Estimation Based on Use Case Points.* Paper presented at the International Symposium on Empirical Software Engineering, Noosa Heads, QLD, Australia.

Anda, B., Dreiem, H., Sjoberg, D. I. K., & Jorgensen, M. (2001, October 1-5). *Estimating Software Development Effort Based on Use Cases-Experiences from Industry.* Paper presented at the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools (UML' 2001), Toronto, Canada.

Antoniol, G., Fiutem, R., & Lokan, C. (2003). Object-Oriented Function Points: An Empirical Validation. *Empirical Software Engineering, 8*(3), 225-254. doi:10.1023/a:1024472727275

Antoniol, G., Lokan, C., Caldiera, G., & Fiutem, R. (1999). A Function Point-Like Measure for Object-Oriented Software. *Empirical Software Engineering, 4*(3), 263-287. doi:10.1023/a:1009834811663

Arnold, M., & Pedross, P. (1998). *Software Size Measurement and Productivity Rating in a Large-scale Software Development Department*. Paper presented at the Proceedings of the 20th international conference on Software engineering, Kyoto, Japan.

Avison, D., & Wilson, D. (2001). A Viewpoint on Software Engineering and Information Systems: What We Can Learn from the Construction Industry? *Information and Software Technology, 43*(13), 795-799. doi:http://dx.doi.org/10.1016/S0950-5849(01)00186-0

AXELOS. (2017). *Managing Successful Projects with PRINCE2*. United Kingdom: TSO, The Stationery Office.

Azzeh, M., Nassif, A., & Attili, I. B. (2021). Predicting Software Effort from Use Case Points: A Systematic Review. *Science of Computer Programming, 204*, 1-26. doi:10.1016/j.scico.2020.102596

Azzeh, M., & Nassif, A. B. (2017). Analyzing the Relationship Between Project Productivity and Environment Factors in the Use Case Points Method. *Journal of Software: Evolution and Process, 29*(9), 1-19. doi:doi:10.1002/smr.1882

Azzeh, M., Nassif, A. B., Banitaan, S., & López-Martín, C. (2018, Dec. 17-20). *Ensemble of Learning Project Productivity in Software Effort Based on Use Case Points.* Paper presented at the 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, USA.

Azzeh, M., Nassif, A. B., & Martín, C. L. (2021). **Empirical Analysis on Productivity Prediction and Locality for Use Case Points Method**. *Software Quality Journal, 29*(2), 309-336. doi:10.1007/s11219-021-09547-0

Bagheri, S., & Shameli-Sendi, A. (2018, June 13-15). *Software Project Estimation Using Improved Use Case Point.* Paper presented at the 16th International Conference on Software Engineering Research, Management and Application (SERA 2018), Kunming, China.

Balzer, Cheatham, & Green. (1983). Software Technology in the 1990's: Using a New Paradigm. *Computer, 16*(11), 39-45. doi:10.1109/MC.1983.1654237

Barabino, G., Concas, G., Corona, E., Grechi, D., Marchesi, M., & Tigano, D. (2015). Web Framework Points: An Effort Estimation Methodology for Web Application Development Using a Content Management Framework. *Journal of Software: Evolution and Process, 27*(9), 603-624. doi:10.1002/smr.1715

Barki, H., Rivard, S., & Talbot, J. (2001). An Integrative Contingency Model of Software Project Risk Management. *Journal of Management Information Systems, 17*(4), 37-69. doi:https://doi.org/10.1080/07421222.2001.11045666

Barrows, H. S., & Tamblyn, R. H. (1980a). *Problem-Based Learning: An Approach to Medical Education* (1st ed.). Broadway, New York, USA: Springer.

Barrows, H. S., & Tamblyn, R. H. (1980b). *Problem-Based Learning: An Approach to Medical Education*. Broadway, New York, USA: Springer.

Basili, V. (1985, September 10-13). *Quantitative Evaluation of Software Methodology.* Paper presented at the The First Pan Pacific Computer Conference, Melbourne, Australia.

Basili, V. (1989, September 20-22). *Software Development: a Paradigm for the Future.* Paper presented at the Proceedings of the 13th Annual International Computer Software and Applications Conference (COMPSAC 89) Orlando, FL, USA.

Basili, V., Caldiera, G., McGarry, F., Pajerski, R., Page, G., & Waligora, S. (1992, May 11-15). *The Software Engineering Laboratory-an Operational Software Experience Factory.* Paper presented at the 14th International Conference on Software Engineering (ICSE 92), Melbourne, Australia.

Basili, V., Caldiera, G., & Rombach, H. D. (2002a). Experience Factory. In *Encyclopedia of Software Engineering*: John Wiley & Sons, Inc.

Basili, V., Caldiera, G., & Rombach, H. D. (2002b). Goal Question Metric (GQM) Approach. In *Encyclopedia of Software Engineering* (Vol. 2, pp. 527–532): John Wiley & Sons, Inc.

Basili, V., Costa, P., Lindvall, M., Mendonca, M., Seaman, C., Tesoriero, R., & Zelkowitz, M. (2001, Nov 27-29). *An Experience Management System for a Software Engineering Research Organization.* Paper presented at the 26th Annual NASA Goddard Software Engineering Workshop, Greenbelt, MD, USA.

Basili, V., & Green, S. (1994). Software Process Evolution at the SEL. *Software, IEEE, 11*(4), 58-66. doi:10.1109/52.300090

Basili, V. R. (1992). *Software Modeling and Measurement: The Goal Question Metric Paradigm* (CS-TR-2956 (UMIACS-TR-92-96)). Retrieved from Computer Science Technical Report Series, University of Maryland, USA:

Basili, V. R. (1996, 25-29 Mar 1996). *The Role of Experimentation in Software Engineering: Past, Current, and Future.* Paper presented at the Software Engineering, 1996., Proceedings of the 18th International Conference on.

Basili, V. R. (2011). Learning through Applications: The Maturing of the QIP in the SEL. In A. O. a. G. Wilson (Ed.), *Making Software What Really Works, and Why We Believe It* (pp. 65-78): O' Reilly.

Basili, V. R., & Reiter, R. W., Jr. (1981). A Controlled Experiment Quantitatively Comparing Software Development Approaches. *Software Engineering, IEEE Transactions on, SE-7*(3), 299-320.

Basili, V. R., & Rombach, H. D. (1988). The TAME Project: Towards Improvement-oriented Software Environments. *IEEE Transactions on Software Engineering, 14*(6), 758-773.

Basili, V. R., & Seaman, C. (2002). The Experience Factory Organization. *Ieee Software, 19(3),* 30-31.

Basili, V. R., & Selby, R. W. (1984, August 13-16). *Data Collection and Analysis in Software Research and Management.* Paper presented at the American Statistical Association and Biomeasure Society Joint Statistical Meetings, Philadelphia, USA.

Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986). Experimentation in Software Engineering. *IEEE Trans. Softw. Eng., 12*(7), 733-743.

Basili, V. R., Shull, F., & Lanubile, F. (1999). Building Knowledge through Families of Experiments. *Software Engineering, IEEE Transactions on, 25*(4), 456-473.

Basili, V. R., Trendowicz, A., Kowalczyk, M., Heidrich, J., Seaman, C. B., Münch, J., & Rombach, H. D. (2014). *Aligning Organizations Through Measurement - The GQM+Strategies Approach*: Springer.

Basili, V. R., & Weiss, D. M. (1984). A Methodology for Collecting Valid Software Engineering Data. *Software Engineering, IEEE Transactions on, SE-10*(6), 728-738.

Baskerville, R. L., Mathiassen, L., & Pries-Heje, J. (2005). *Business Agility and Information Technology Diffusion*. New York, USA: Springer.

Beck, K. (1999). Embracing Change with Extreme Programming. *Computer, 32*(10), 70-77. doi:10.1109/2.796139

Beck, K. (2003). *Test-Driven Development By Example* (1st ed.). Boston, USA: Addison-Wesley.

Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Boston, USA: Addison-Wesley.

Benington, H. D. (1983). Production of Large Computer Programs. *Annals of the History of Computing, 5*(4), 350-361. doi:10.1109/mahc.1983.10102

Berander, P. (2004). *Using Students as Subjects in Requirements Prioritization.* Paper presented at the Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04), Redondo Beach, CA, USA.

Bergandy, J. (2008, October 22-25). *Software Engineering Capstone Project with Rational Unified Process (RUP).* Paper presented at the Proceedings of the 38th ASEE/IEEE Frontiers in Education Conference (FIE), New York, USA.

Biolchini, J., Mian, P. G., Natali, A. C. C., & Travassos, G. H. (2005). *Systematic Review in Software Engineering*. Retrieved from Rio de Janeiro, Brasil:

Birk, A., Heller, G., John, I., Schmid, K., von der Massen, T., & Muller, K. (2003). Product Line Engineering, the State of the Practice. *Software, IEEE, 20*(6), 52-60.

Blake, M. B. (2005). Integrating Large-scale Group Projects and Software Engineering Approaches for Early Computer Science Courses. *IEEE Transactions on Education, 48*(1), 63-72. doi:10.1109/te.2004.832875

Boehm, B., & Basili, V. (2002). *The CeBASE Framework for Strategic Software Development and Evolution*.

Boehm, B., & Turner, R. (2003a). *Balancing Agility and Discipline: A Guide for the Perplexed* (1st ed.). Boston, USA: Addison-Wesley.

Boehm, B., & Turner, R. (2003b). Using Risk to Balance Agile and Plan-Driven Methods. *Computer, 36*(6), 57–66. doi:10.1109/mc.2003.1204376

Boehm, B., & Turner, R. (2005). Management Challenges to Implementing Agile Processes in Traditional Development Organizations. *Ieee Software, 22*(5), 30-39. doi:10.1109/MS.2005.129

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *Computer, 21*(5), 61-72. Retrieved from 10.1109/2.59

Boehm, B. W. (1991). Software Risk Management: Principles and Practices. *Ieee Software, 8*(1), 32-41. doi:10.1109/52.62930

Bonnie, E. (2021). Fundamentals of the Scrum Methodology. *Project Management · 5 Min Read*. Retrieved from https://www.wrike.com/blog/fundamentals-of-the-scrum-methodology/

Borges, P., Monteiro, P., & Machado, R. J. (2011, August 30 - September 2). *Tailoring RUP to Small Software Development Teams.* Paper presented at the 37th Euromicro Conference (SEAA 2011), Oulu, Finland.

Borges, P., Monteiro, P., & Machado, R. J. (2012). *Mapping RUP Roles to Small Software Development Teams*. Paper presented at the 4th International Conference on Software Quality Days (SWQD 2012), Vienna.

Borges, P. M. P. (2007). *Configuração do RUP com Vista à Simplificação dos Elencos Processuais em PMEs de Desenvolvimento de Software.* (MSc). Escola de Engenharia, Universidade do Minho, Guimarães, Portugal.

Bourque, P., & Abran, A. (1996). *An Experimental Framework for Software Engineering Research.* Paper presented at the Forum on Software Engineering Standards Issues (SES' 96), Montréal, Québec, Canada.

Bourque, P., & Côté, V. (1991). An Experiment in Software Sizing with Structured Analysis Metrics. *Journal of Systems and Software, 15*(2), 159-172. doi:http://dx.doi.org/10.1016/0164-1212(91)90053-9

Brazier, P., Garcia, A., & Vaca, A. (2007, October 10-13). *A Software Engineering Senior Design Project Inherited from a Partially Implemented Software Engineering Class Project.* Paper presented at the Proceedings of the 37th ASEE/IEEE Frontiers in Education Conference (FIE 2007), Milwaukee, Wisconsin, USA.

Broman, D. (2010, March 9-12). *Should Software Engineering Projects Be the Backbone or the Tail of Computing Curricula?* Paper presented at the Proceedings of the 23rd IEEE Conference on Software Engineering Education and Training (CSEE&T), Pittsburgh, USA.

Bruegge, B., & Dutoit, A. H. (2010). *Object-oriented Software Engineering: Conquering Complex and Changing Systems*. USA: Prentice Hall.

Burge, J., & Troy, D. (2006, April 19-21). *Rising to the Challenge: Using Business-Oriented Case Studies in Software Engineering Education.* Paper presented at the Proceedings of the 19th Conference on Software Engineering Education and Training (CSEE&T), Turtle Bay, Hawaii.

Burgstaller, B., & Egyed, A. (2010, September 12-18). *Understanding Where Requirements Are Implemented.* Paper presented at the Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM), Timisoara, Romania.

Caldiera, G., Antoniol, G., Fiutem, R., & Lokan, C. (1998, Nov 20-21). *Definition and Experimental Evaluation of Function Points for Object-oriented Systems.* Paper presented at the Fifth International Software Metrics Symposium (ISMS 98), Bethesda, MD, USA.

Campbell, D. T., & Stanley, J. C. (1963). *Experimental and Quasi-experimental Designs for Research*. Boston, USA: Houghton Mifflin Company.

Carver, J., Jaccheri, L., Morasca, S., & Shull, F. (2003, 3-5 Sept. 2003). *Issues in Using Students in Empirical Studies in Software Engineering Education.* Paper presented at the Software Metrics Symposium, 2003. Proceedings. Ninth International.

Carver, J., Shull, F., & Basili, V. (2003). *Observational Studies to Accelerate Process Experience in Classroom Studies: An Evaluation*. Paper presented at the International Symposium on Empirical Software Engineering (ISESE 2003), Rome, Italy.

Carver, J., VanVoorhis, J., & Basili, V. (2004, 19-20 Aug. 2004). *Understanding the Impact of Assumptions on Experimental Validity.* Paper presented at the Proceedings of 2004 International Symposium on Empirical Software Engineering (ISESE '04) Redondo Beach, California, USA.

Carver, J. C., Jaccheri, L., Morasca, S., & Shull, F. (2010). A Checklist for Integrating Student Empirical Studies with Research and Teaching Goals. *Empirical Software Engineering, 15*(1), 35-59. doi:10.1007/s10664-009-9109-9

Cavano, J. P., & McCall, J. A. (1978). *A Framework for the Measurement of Software Quality*. Paper presented at the Software Quality Assurance Workshop on Functional and Performance issues, New York, USA. https://doi.org/10.1145/800283.811113

Ceddia, J., & Dick, M. (2004). *Automating the Estimation of Project Size from Software design Tools Using Modified Function Points*. Paper presented at the Sixth Australasian Conference on Computing Education Conference (ACE2004), Dunedin, New Zealand.

Chamundeswari, A., & Babu, C. (2010, February 25). *An Extended Function Point Approach for Size Estimation of Object-Oriented Software.* Paper presented at the 3rd India Software Engineering Conference (ISEC' 2010), Mysore, India.

Chapman, R. J. (2011). *Simple Tools and Techniques for Enterprise Risk Management* (2nd ed.). West Sussex, United Kingdom: John Wiley & Sons Ltd.

Charette, R. N. (2005). Why Software Fails [Software Failure]. *IEEE Spectrum, 42*(9), 42-49. doi:10.1109/MSPEC.2005.1502528

Cherques, H. R. T. (2004). *Modelagem de Projetos* (2nd ed.). Brasil: Atlas.

Chowdhury, A. A., & Arefeen, S. (2011). Risk Management : Importance and Practices. *International Journal of Computer and Information Technology (IJCIT), 02*(01), 49-54.

Clark, N. (2005). *Evaluating Student Teams Developing Unique Industry Projects*. Paper presented at the 7th Australasian Computing Education Conference (ACE2005), Newcastle, Australia.

Coad, P., Luca, J. d., & Lefebvre, E. (1999). *Java Modeling In Color With UML: Enterprise Components and Process*: Prentice Hall.

Cockburn, A. (1998). *Surviving Object-Oriented Projects* (1st ed.): Addison-Wesley.

Cockburn, A. (2000). *Writing Effective Use Cases* (1st ed.). New York, USA: Addison-Wesley.

Cockburn, A. (2005). *Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams* (1st ed.). Boston, USA: Addison-Wesley.

Coplien, J. O., & Harrison, N. B. (2005). *Organizational Patterns of Agile Software Development* (1st ed.). Upper Saddle River, New Jersey, USA: Pearson.

COSMIC. (2020). *The COSMIC Functional Size Measurement Method Version 4.0*. Retrieved from https://cosmic-sizing.org/wp-content/uploads/2020/08/Measuring-software-size-v1.0-August-2020-1.pdf

Creswell, J. W., & Creswell, J. D. (2017). *Research Design: Qualitative, Quantitative and Mixed Methods Approaches* (5th ed.). Thousand Oaks, California: Sage Publications Inc.

Damodaran, M., & Washington, A. (2002). *Estimation Using Use Case Points*. Computer Science. University of Houston-Victoria. Texas, USA. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=1ED37920C53DC970678BAB05F6296014?doi=10.1.1.130.7226&rep=rep1&type=pdf

Dekkers, C. A. (2003). Measuring the “Logical” or “Functional” Size of Software Projects and Software Application. *ISO Bulletin*, 10-13.

DeMarco, A. A. (1992). *Function Point Software Sizing*. PRICE Systems.

Deming, W. E. (1990). *Out of the Crisis*. Cambridge, Massachussetts, USA: The MIT Press.

Denzin, N. K., & Lincoln, Y. S. (2011). *The SAGE Handbook of Qualitative Research* (4th Ed. ed.). Los Angeles, USA: SAGE Publications, Inc.

DSDMConsortium. (2003). *DSDM: Business Focused Development* (2nd ed.): Addison-Wesley.

Dutoit, A. H., McCall, R., Mistrík, I., & Paech, B. (2006). Rationale Management in Software Engineering: Concepts and Techniques. In A. H. Dutoit, R. McCall, I. Mistrík, & B. Paech (Eds.), *Rationale Management in Software Engineering* (pp. 1-48). Berlin, Heidelberg: Springer Berlin Heidelberg.

Dvir, D., & Shenhar, A. J. (2010). *Reinventando Gerenciamento de Projetos. A Abordagem Diamante ao Crescimento e Inovação Bem-Sucedidos* (1st ed.): Mc Graw-Hill.

Dybå, T., Kampenes, V. B., & Sjøberg, D. I. K. (2005). A Systematic Review of Statistical Power in Software Engineering Experiments. *Information and Software Technology, 48*(8), 745-755. doi:http://dx.doi.org/10.1016/j.infsof.2005.08.009

Dyba, T., Kitchenham, B. A., & Jorgensen, M. (2005). Evidence-based Software Engineering for Practitioners. *Software, IEEE, 22*(1), 58-65. doi:10.1109/ms.2005.6

EASE. (2021). International Conference on Evaluation and Assessment in Software Engineering (EASE). Retrieved 2021-07-12 from https://conf.researchr.org/home/ease-2021

Eckstein, J. (2004). *Agile Software Development in Large - Diving into the Deep*. New York, USA: Dorset House.

EMSE. (2021). Empirical Software Engineering (EMSE). Retrieved 2021-07-12 from https://www.springer.com/journal/10664

Engle, C. B. (1989). Software Engineering is Not Computer Science. In N. Gibbs (Ed.), *Software Engineering Education* (Vol. 376, pp. 257-262). New York, USA: Springer.

ESE Group. (2021a). Experimental and Software Engineering Group, Universidade Federal do Rio de Janeiro (UFRJ/COPPE). Retrieved 2021-07-12 from http://lens-ese.cos.ufrj.br/ese/

ESE Group. (2021b). Experimental Software Engineering - Glossary of Terms. Retrieved 2021-07-13 from http://lens-ese.cos.ufrj.br/wikiese/index.php/Experimental_Software_Engineering_-_Glossary_of_Terms#A

ESE Helsinki. (2021). Empirical Software Engineering Helsinki (ESE). Retrieved 2021-07-13 from https://www2.helsinki.fi/en/researchgroups/empirical-software-engineering

European Commission. (2021). The Bologna Process - Towards the European Higher Education Area. Retrieved 2021-09-08 from http://ec.europa.eu/education/higher-education/bologna_en.htm

Fenton, N., & Bieman, J. (2015). *Software Metrics: A Rigorous and Practical Approach* (3rd ed.). Boca Raton, USA: CRC Press.

Fenton, N. E., & Neil, M. (2000, June 4-11). *Software Metrics: Roadmap.* Paper presented at the Conference on The Future of Software Engineering (CFSE 2000), Limerick, Ireland.

Fetcke, T., Abran, A., & Nguyen, T.-H. (1997). Mapping the OO-Jacobson Approach to Function Point Analysis. In F. Lehner, R. Dumke, & A. Abran (Eds.), *Software Metrics: Research and Practice in Software Measurement* (pp. 59-73). Wiesbaden: Deutscher Universitätsverlag.

Fetcke, T., Abran, A., & Tho-Hau, N. (1998, Jul 28 - Aug 1). *Mapping the OO-Jacobson Approach into Function Point Analysis.* Paper presented at the TOOLS 23' 97.

Fidel, R. (1984). The Case Study Method: A Case Study. *Library and Information Science Research, 6*(3), 273-278.

Finkelstein, A. (2011). Ten Open Challenges at the Boundaries of Software Engineering and Information Systems. In H. Mouratidis & C. Rolland (Eds.), *Advanced Information Systems Engineering* (Vol. 6741, pp. 1-1). New York, USA: Springer Berlin Heidelberg.

Fitzgerald, B. (1997). The Use of Systems Development Methodologies in Practice: a Field Study. *Information Systems Journal, 7*(3), 201-212. doi:https://doi.org/10.1046/j.1365-2575.1997.d01-18.x

Fitzgerald, B. (1998). An Empirical Investigation into the Adoption of Systems Development Methodologies. *Information & Management, 34*(6), 317-328. Retrieved from <Go to ISI>://000077405100002

Fraunhofer. (2021). Fraunhofer-Gesellschaft. Retrieved 2021-07-13 from https://www.fraunhofer.de/en.html

Freedman, R. (2010). Adaptive Project Framework: A New Level of Agile Development. *TechRepublic*. Retrieved from https://www.techrepublic.com/article/adaptive-project-framework-a-new-level-of-agile-development/

Freitas, M., Fantinato, M., Sun, V., Thom, L. H., & Garaj, V. (2020). Function Point Tree-Based Function Point Analysis: Improving Reproducibility Whilst Maintaining Accuracy in Function Point Counting. In J. Filipe, M. Śmiałek, A. Brodsky, & S. Hammoudi (Eds.), *Enterprise Information Systems* (Vol. 378, pp. 182-209): Springer International Publishing.

Fung, R. Y. K., Tam, W. T., Ip, A. W. H., & Lau, H. C. W. (2002). Software Process Improvement Strategy for Enterprise Information Systems Development. *International Journal of Information Technology and Management, 1*(2-3), 225-241. doi:10.1504/ijitm.2002.001198

Furey, S. (1997). Why We Should Use Function Points. *IEEE on Software, 14*(2), 28-30. doi:10.1109/52.582971

Galliers, R. (1991). Choosing Appropriate Information Systems Research Approaches: A Revised Taxonomy. In H. E. Nissen, H. K. Klein, & R. Hirschheim (Eds.), *Information Systems Research: Contemporary Approaches and Emergent Traditions* (pp. 327-345): Elsevier Science Publishers (North Holland).

Galliers, R. D., & Sutherland, A. R. (2003). The Evolving Information Systems Strategy. In R. D. Galliers & D. E. Leidner (Eds.), *Strategic Information Management* (3rd ed., pp. 33-63). Oxford, UK: Butterworth-Heinemann.

Garmus, D., & Herron, D. (2000). *Function Point Analysis: Measurement Practices for Successful Software Projects* (1st ed.). New Jersey, USA: Addison-Wesley.

Gilb, T. (1977). *Software Metrics* (1st ed.). Cambridge, Massachusetts, USA: Winthrop Publishers.

Gilb, T. (1981). Evolutionary Development. *ACM SIGSOFT Software Engineering Notes, 6*(2), 17-17. doi:http://doi.acm.org/10.1145/1010865.1010868

Gilb, T. (1988). *Principles of Software Engineering Management* (1st ed.). Wokingham, UK: Addison-Wesley.

Gilb, T. (2005). *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage* (1 st ed.). Burlington, UK: Butterworth-Heinemann.

Glass, R. L. (2007a). Is Software Engineering Fun? *Ieee Software, 24*(1), 96-95. doi:10.1109/ms.2007.18

Glass, R. L. (2007b). Is Software Engineering Fun? Part 2. *Ieee Software, 24*(2), 104-103. doi:10.1109/ms.2007.46

Goulao, M., & Brito e Abreu, F. (2007, 12-14 Sept. 2007). *Modeling the Experimental Software Engineering Process.* Paper presented at the 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), Lisbon New University, Lisbon, Portugal.

Gould, P. (1997). What is Agility? *Manufacturing Engineer, 76*(1), 28-31. doi:doi.org/10.1049/me:19970113

Gregg, D., Kulkarni, U., & Vinzé, A. (2001). Understanding the Philosophical Underpinnings of Software Engineering Research in Information Systems. *Information Systems Frontiers, 3*(2), 169-183. doi:10.1023/a:1011491322406

Han, W.-M., & Huang, S.-J. (2007). An Empirical Analysis of Risk Components and Performance on Software Projects. *Journal of Systems and Software, 80*(1), 42-50. doi:https://doi.org/10.1016/j.jss.2006.04.030

Harput, V., Kaindl, H., & Kramer, S. (2005, Sept. 19-22). *Extending Function Point Analysis to Object-oriented Requirements Specifications.* Paper presented at the 11th IEEE International Software Metrics Symposium (METRICS 2005), Como, Italy.

Harrison, W. (2000). *N=1 An Alternative for Software Engineering Research?* Paper presented at the 22nd International Conference on Software Engineering (ICSE 2000), Limerick, Ireland.

Hayes, J. H. (2002, February 25-27). *Energizing Software Engineering Education through Real-World Projects as Experimental Studies.* Paper presented at the 15th Conference on Software Engineering Education and Training, Ottawa, Canada.

Hayes, J. H., & Dekhtyar, A. (2005). A Framework for Comparing Requirements Tracing Experiments. *International Journal of Software Engineering & Knowledge Engineering, 15*(5), 751-781.

Hellens, L. A. (1997). Information Systems Quality Versus Software Quality a Discussion from a Managerial, an Organisational and an Engineering Viewpoint. *Information and Software Technology, 39*(12), 801-808. doi:http://dx.doi.org/10.1016/S0950-5849(97)00038-4

Highsmith, J., & Cockburn, A. (2001). Agile Software Development: The Business of Innovation. *Computer, 34*(9), 120–122. doi:10.1109/2.947100

Highsmith, J. A. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York, USA: Dorset House Publishing.

Hillman, M. F., & Subriadi, A. P. (2019). 40 Years Journey of Function Point Analysis: Against Real-time and Multimedia Applications. *Procedia Computer Science, 161*, 266-274. doi:https://doi.org/10.1016/j.procs.2019.11.123

Höst, M. (2002, 25-27 February). *Introducing Empirical Software Engineering Methods in Education.* Paper presented at the CSEE&T 2002, Covington, Kentucky, USA.

Höst, M., Regnell, B., & Wohlin, C. (2000). Using Students as Subjects -A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering, 5*(3), 201-214. doi:10.1023/a:1026586415054

Hove, S. E., & Anda, B. (2005). *Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research.* Paper presented at the 11th IEEE International Symposium Software Metrics (METRICS 2005), Como, Italy.

Hubstaff. (2021). 2021 Remote Project Management Report. Retrieved 2022-05-26 from https://hubstaff.com/tasks/state-of-remote-project-management#contents-category-2

Humphrey, W. S. (1995). *A Discipline for Software Engineering*: Addison Wesley.

Ibbs, C. W., & Kwak, Y. H. (2000). Assessing Project Management Maturity. *Project Management Journal, 31*(1), 32-43. doi:10.1177/875697280003100106

IEC 31010:2019. (2019). *Risk Management -- Risk Assessment Techniques*. Retrieved from https://www.iso.org/standard/72140.html

IEEE. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK)* (P. Bourque & R. E. Fairley Eds.): IEEE Computer Society Press.

IESE, F. (2021). Fraunhofer Institute for Experimental Software Engineering. Retrieved 2021-07-12 from http://www.iese.fraunhofer.de/en.html

IFPUG. (2010). *Function Point Counting Practices Manual: Release 4.3.1* (978-0-9753783-4-2). Retrieved from USA: https://www.ifpug.org/

IFPUG. (2021). International Function Point Users Group. Retrieved 2021-09-24 from http://www.ifpug.org

Irawati, A. R., & Mustofa, K. (2012). Measuring Software Functionality Using Function Point Method Based on Design Documentation. *International Journal of Computer Science Issues, 9*(3), 124-130. Retrieved from https://www.ijcsi.org/articles/Measuring-software-functionality-using-function-point-method-based-on-design-documentation.php

ISBSG. (2021). International Software Benchmarking Standards Group. Retrieved 2021-09-24 from https://www.isbsg.org

ISERN. (2021). International Software Engineering Research Network. Retrieved 2021-07-14 from https://isern.iese.de/

Iskandar, K., Gaol, F. L., Soewito, B., Warnars, H. L. H. S., & Kosala, R. (2016, Oct 3-5). *Software Size Measurement of Knowledge Management Portal with Use Case Point.* Paper presented at the 2016 International Conference on Computer, Control, Informatics and its Applications (IC3INA), Jakarta, Indonesia.

Jaccheri, L., & Morasca, S. (2006, 26-30 June). *Involving industry professionals in empirical studies with students.* Paper presented at the International Conference on ESE Issues: Critical Assessment and Future Directions, Dagstuhl Castle, Germany.

Jaccheri, L., & Osterlie, T. (2005). *Can We Teach Empirical Software Engineering?* Paper presented at the 11th IEEE International Software Metrics Symposium (METRICS 2005), Como, Italy.

Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. New York, USA: Addison-Wesley.

James, T. (2005). Stepping Back from Lean. *Manufacturing Engineer, 84*(1), 16-21. doi:10.1049/me:20050101

Jayaratna, N., & Sommerville, I. (1998). The Role Of Information Systems Methodology In Software Engineering [Editorial]. *IEE Proceedings - Software, 145*(4), 93-94.

Jedlitschka, A., & Pfahl, D. (2005, 17-18 Nov. 2005). *Reporting Guidelines for Controlled Experiments in Software Engineering.* Paper presented at the 2005 International Symposium on Empirical Software Engineering (ISESE 2005), Noosa Heads, Australia.

Jeffries, R. E. (1999). eXtreme Testing: Why Aggressive Software Development Calls for Radical Testing Efforts. *Software Testing & Quality Engineering, March/April*, 23-26.

Jones, C. (1988, October 11-14). *A Short History of Function Point and Feature Points.* Paper presented at the IFPUG Fall 1988 Conference, Montreal, Québec, Canada.

Jones, C. (1994). Function Points. *Computer, 27*(8), 66-67. doi:10.1109/MC.1994.10088

Jones, C. (2004). Software Project Management Practices: Failure Versus Success. *CrossTalk : the journal of defense software engineering*(October, 2004). Retrieved from http://www.pauldee.org/se-must-have/jones-failure-success.pdf

Jones, C. (2007). *Estimating Software Costs: Bringing Realism to Estimating* (2nd ed.). New York: McGraw-Hill

Júnior, I. G., & Chaves, M. S. (2014). Novos Riscos para a Gestão de Projetos de Tecnologia da Informação com Equipes Locais. *Iberoamerican Journal of Project Management, 5*(2), 16-38.

Juristo, N., & Moreno, A. M. (2001). *Basics of Software Engineering Experimentation*. Netherlands: Kluwer Academic Publishers.

Kähkönen, T., & Pekka, A. (2004). Achieving CMMI Level 2 with Enhanced Extreme Programming Approach. In F. Bomarius & H. Iida (Eds.), *Product Focused Software Process Improvement* (pp. 378-392). Berlin, Heidelberg: Springer.

Karahasanovic, A., Anda, B., Arisholm, E., Hove, S., Jørgensen, M., Sjøberg, D., & Welland, R. (2005). Collecting Feedback during Software Engineering Experiments. *Empirical Software Engineering, 10*(2), 113-147. doi:10.1007/s10664-004-6189-4

Karner, G. (1993). Resource Estimation for Objectory Projects. *Objective Systems SF AB.* Retrieved from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.604.7842&rep=rep1&type=pdf

Karolak, D. W. (1996). *Software Engineering Risk Management* (1st ed.). Los Alamitos, Califórnia, USA: IEEE Computer Society Press.

Karunasekera, S., & Bedse, K. (2007). *Preparing Software Engineering Graduates for an Industry Career*. Paper presented at the 20th Conference on Software Engineering Education & Training (CSEET '07), Dublin, Ireland.

KBSt. (2004). *Federal Government Co-Ordination and Advisory Agency The New V-Modell XT: Development Standard for IT Systems of the Federal Republic of Germany*. Retrieved from Germany: http://ftp.uni-kl.de/pub/v-modell-xt/Release-1.1-eng/Dokumentation/pdf/V-Modell-XT-eng-Teil1.pdf

Kendrick, T. (2015). *Identifying and Managing Project Risk: Essential Tools for Failure-Proofing Your Project* (3rd Ed.). New York. USA: AMACOM.

Kerzner, H. (2017). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling* (12th ed.). Hoboken, New Jersey, USA: John Wiley & Sons, Inc.

Kettunen, P. (2010, March 2010). Software Factory Fosters Next-Generation Software Production and Business. *Software Factory Magazine, 1,* 5.

Kezsbom, D. S., & Edward, K. A. (2001). *The New Dynamic Project Management: Winning Through the Competitive Advantage* (2nd ed.). Hoboken, New Jersey, USA: John Wiley & Sons.

Kitchenham, B. (1996a). *Software Metrics: Measurement for Software Process Improvement*: Blackwell Pub.

Kitchenham, B. (1997). Counterpoint: The Problem with Function Points. *IEEE Software, 14*(2), 29-31. doi:10.1109/ms.1997.582972

Kitchenham, B., & Charters, S. (2007). *Guidelines for Performing Systematic Literature Reviews in Software Engineering* (EBSE-2007-01). Retrieved from http://www.elsevier.com/framework_products/promis_misc/525444systematicreviewsguide.pdf

Kitchenham, B., & Känsäla, K. (1993). *Inter-item Correlations Among Function Points*. Paper presented at the 15th international conference on Software Engineering, Baltimore, Maryland, USA.

Kitchenham, B., & Pfleeger, S. L. (2002). Principles of Survey Research: part 5: Populations and Samples. *SIGSOFT Softw. Eng. Notes, 27*(5), 17-20. doi:10.1145/571681.571686

Kitchenham, B., Pickard, L., & Pfleeger, S. L. (1995). Case Studies for Method and Tool Evaluation. *Software, IEEE, 12*(4), 52-62.

Kitchenham, B. A. (1996b). Evaluating Software Engineering Methods and Tool Part 1: The Evaluation Context and Evaluation Methods. *SIGSOFT Softw. Eng. Notes, 21*(1), 11-14. doi:10.1145/381790.381795

Kitchenham, B. A. (2004, 11-17 Sept. 2004). *Systematic Reviews.* Paper presented at the Software Metrics, 2004. Proceedings. 10th International Symposium on.

Kitchenham, B. A., Dyba, T., & Jorgensen, M. (2004, 23-28 May 2004). *Evidence-based Software Engineering.* Paper presented at the ICSE 2004 - 26th International Conference on Software Engineering, 2004, Edinburgh, Scotland.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K., & Rosenberg, J. (2002). Preliminary Guidelines for Empirical Research in Software Engineering. *Software Engineering, IEEE Transactions on, 28*(8), 721-734.

Kornecki, A. J., Khajenoori, S., Gluch, D., & Kameli, N. (2003, March 20-22). *On a Partnership Between Software Industry and Academia.* Paper presented at the Proceedings of the 16th Conference on Software Engineering Education and Training (CSEE&T) Madrid, Spain.

Koskela, L., & Howell, G. (2002, June 2002). *The Underlying Theory of Project Management is Obsolete.* Paper presented at the The PMI Research Conference 2002, Seattle, Washington, USA.

Krigsman, M. (2012). Worldwide cost of IT failure (revisited): $3 trillion. Retrieved 2022-04-10 from https://www.zdnet.com/article/worldwide-cost-of-it-failure-revisited-3-trillion/

Kumawat, P., & Sharma, N. (2019). Design and Development of Cost Measurement Mechanism for Re-Engineering Project Using Function Point Analysis. *Advances in Intelligent Systems and Computing, 870*, 311-319. doi:10.1007/978-981-13-2673-8_33

Kurbel, K., Krybus, I., & Nowakowski, K. (2021). Lehrstuhl für Wirtschaftsinformatik. Retrieved 2021-09-10 from http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/uebergreifendes/

Kurbel, K. E. (2008). *The Making of Information Systems: Software Engineering and Management in a Globalized World*. Berlin, Germany: Springer.

Kurniadi, D., Hendric, H. L., Gaol, F. L., & Soewito, B. (2017). *Software Size Measurement of Student Information Terminal With Use Case Point*. Paper presented at the 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), Phuket, Thailand.

Kurniadi, D., Mulyani, A., Septiana, Y., & Aulawi, H. (2018). Estimated Software Measurement Base on Use Case for Online Admission System. *IOP Conference Series: Materials Science and Engineering, 434*, 1-7. doi:10.1088/1757-899x/434/1/012062

Kutsch, E., Denyer, D., Hall, M., & Lee-Kelley, E. (2013). Does Risk Matter? Disengagement from Risk Management Practices in Information Systems Projects. *European Journal of Information Systems, 22*(6), 637-649. doi:10.1057/ejis.2012.6

Larman, C. (2004). *Agile and Iterative Development: A Manager's Guide* (1st ed.). Boston, USA: Addison-Wesley.

Larman, C., & Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *Computer, 36*(6), 47. Retrieved from <Go to ISI>://000183263200012

Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., . . . Zelkowitz, M. (2002, August 4-7). *Empirical Findings in Agile Methods.* Paper presented at the Extreme Programming and Agile Methods — XP/Agile Universe 2002, Berlin, Heidelberg.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., . . . Kahkonen, T. (2004). Agile Software Development in Large Organizations. *Computer, 37*(12), 26-34. doi:10.1109/MC.2004.231

Lokan, C. J. (2000). An Empirical Analysis of Function Point Adjustment Factors. *Information and Software Technology, 42*(9), 649-659. doi:http://dx.doi.org/10.1016/S0950-5849(00)00108-7

Longstreet, D. (2005). *Fundamentals of Function Point Analysis*. Software Metrics. Longstreet Consulting Inc. Walnut, USA. Retrieved from https://www.math.unipd.it/~tullio/IS-1/2004/Approfondimenti/Function_Point_Analysis.pdf

Lycett, M., Macredie, R. D., Patel, C., & Paul, R. J. (2003). Migrating Agile Methods to Standardized Development Practice. *Computer, 36*(6), 79-85. doi:10.1109/MC.2003.1204379

Mahmood, Y., Kama, N., & Azmi, A. (2020). A Systematic Review of Studies on Use Case Points and Expert-based Estimation of Software Development Effort. *Journal of Software: Evolution and Process*, 1-20. doi:10.1002/smr.2245

Malouin, J.-L., & Landry, M. (1983). The Miracle of Universal Methods in Systems Design. *Journal of Applied Systems Analysis, 10*, 47-62.

Mandić, V., Markkula, J., & Oivo, M. (2009). Towards Multi-Method Research Approach in Empirical Software Engineering. In F. Bomarius, M. Oivo, P. Jaring, & P. Abrahamsson (Eds.), *Product-Focused Software Process Improvement* (Vol. 32, pp. 96-110): Springer Berlin Heidelberg.

Mandjam, F. (2011). *Avaliação do Impacto das Práticas do CMMI no Desempenho de Equipas de Desenvolvimento de Software no Ensino.* (Master Degree Thesis). Escola de Engenharia, Universidade do Minho, Portugal.

McBride, N. (2003). A Viewpoint on Software Engineering and Information Systems: Integrating the Disciplines. *Information and Software Technology, 45*(5), 281-287. doi:http://dx.doi.org/10.1016/S0950-5849(02)00213-6

McCracken, D. D., & Jackson, M. A. (1982). Life Cycle Concept Considered Harmful. *ACM SIGSOFT Software Engineering Notes, 7*(2), 29–32. doi:10.1145/1005937.1005943

McManus, J. (2004). *Risk Management in Software Development Projects* (1st ed.). New York, USA: Routledge.

Microsoft. (2022). Microsoft Project. Retrieved 2022-06-01 from https://www.microsoft.com/en-us/microsoft-365/project/project-management-software

Miguel, A. (2019). *Gestão Moderna de Projetos - Melhores Técnicas e Práticas* (8th ed.). Lisboa, Portugal: FCA - Editora Informática.

Miguel, A. S. G. (2002). *O Risco e a Gestão do Risco em Projectos de Desenvolvimento de Sistemas de Informação.* (PhD). Universidade do Minho, Retrieved from http://repositorium.sdum.uminho.pt/bitstream/1822/26594/1/Tese%20de%20Doutoramento-Ant%c3%b3nio%20Miguel.pdf

Misic, V. B., & Tesic, D. (1998, September 22-25). *Downsizing the Estimation of Software Quality: a Small Object-oriented Case Study.* Paper presented at the TOOLS 27, Beijing, China.

Monteiro, P., Borges, P., Machado, R. J., & Ribeiro, P. (2012, June 2-3). *A Reduced Set of RUP Roles to Small Software Development Teams.* Paper presented at the International Conference on Software and System Process (ICSSP), Zurich, Switzerland.

Monteiro, P., Machado, R., Kazman, R., Lima, A., Simões, C., & Ribeiro, P. (2013). Mapping CMMI and RUP Process Frameworks for the Context of Elaborating Software Project Proposals. In D. Winkler, S. Biffl, & J. Bergsmann

(Eds.), *Software Quality. Increasing Value in Software and Systems Development* (Vol. 133, pp. 191-214): Springer Berlin Heidelberg.

Montgomery, D. (2012). *Design and Analysis of Experiments* (8 th ed.). New York, USA: John Wiley & Sons, Inc.

Morien, R. (2005, Aug 10-12). *Agile Management and the Toyota way for software project management.* Paper presented at the 3rd IEEE International Conference on Industrial Informatics (INDIN), Perth, AUSTRALIA.

NATO Science Committee. (1969). *Software Engineering Report on a Conference Sponsored by the NATO Science Committee* Retrieved from Garmisch: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/

Naveda, J. F. (1999, November 10-13). *Teaching Architectural Design in an Undergraduate Software Engineering Curriculum.* Paper presented at the Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference (FIE), San Juan, Puerto Rico.

Nawrocki, J., Walter, B., & Wojciechowski, A. (2001). *Toward Maturity Model for Extreme Programming*. Paper presented at the Proceedings of the 27th EUROMICRO Conference, Warsaw, Poland.

NESMA. (2021). Netherlands Software Metrics Association. Retrieved 2021-09-24 from https://nesma.org/

Nogueira, M. (2009). *Engenharia de Software: Um Framework Para a Gestão de Riscos em Projetos de Software*. Rio de Janeiro, Brasil: Editora Ciência Moderna.

Nogueira, M., & Machado, R. J. (2012, September 3-6). *A Importância da Adoção da Abordagem de Riscos no Ensino da Engenharia de Software.* Paper presented at the XL - Congresso Brasileiro de Educação em Engenharia, Belém, Brasil.

Nonaka, I., & Takeuchi, H. (1995). *The Knowledge-Creating Company*. New York, USA: Oxford University Press.

NTNU. (2021). Norwegian University of Science and Technology. Retrieved 2021-07-10 from https://www.ntnu.edu/

NTNU_ISSE. (2021). Information Systems and Software Engineering (ISSE). Retrieved 2021-07-12 from https://www.ntnu.edu/idi/isse

Nunan, T. (1999). *Graduate Qualities, Employment and Mass Higher Education*. Paper presented at the HERDSA Annual International Conference, Melbourne, Australia.

Ochodek, M., Nawrocki, J., & Kwarciak, K. (2011). Simplifying Effort Estimation Based on Use Case Points. *Information and Software Technology, 53*(3), 200-213. doi:http://dx.doi.org/10.1016/j.infsof.2010.10.005

Oleson, J. D. (1998). *Pathways to Agility: Mass Customization in Action*. New York, USA: John Wiley & Sons, Inc.

Ott, L. M., Kinnula, A., Seaman, C., & Wohlin, C. (1999). The Role of Empirical Studies in Process Improvement. *Empirical Software Engineering, 4*(4), 381-386. doi:10.1023/A:1009821806231

Papke-Shields, K. E., Beise, C., & Quan, J. (2010). Do Project Managers Practice What they Preach, and Does it Matter to Project Success? *International Journal of Project Management, 28*(7), 650-662. doi:https://doi.org/10.1016/j.ijproman.2009.11.002

Paulk, M. C. (2001). Extreme Programming from a CMM Perspective. *Ieee Software, 18*(6), 19-26. doi:10.1109/52.965798

Perry, D. E., Porter, A. A., & Votta, L. G. (2000). *Empirical Studies of Software Engineering: a Roadmap*. Paper presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland.

Pfleeger, S. L. (1999). Albert Einstein and Empirical Software Engineering. *Computer, 32*(10), 32-38. doi:10.1109/2.796106

Pinto, J. K. (2019). *Project Management: Achieving Competitive Advantage* (5th ed.). New York, USA: Pearson Education, Inc.

Pinto, J. K., & Kharbanda, O. P. (1995). *Successful Project Managers: Leading Your Team to Success*. New York, USA: Van Nostrand Reinhold.

Platt, J. R. (2011). Career Focus: Software Engineering. *IEEE-USA Today's Engineer*. Retrieved from http://www.cis.umassd.edu/~hxu/courses/cis582/CareerFocus.html

PMI. (2017a). *A Guide to the Project Management Body of Knowledge (PMBOK Guide)* (6th ed.). Newtown Square, Pennsylvania, USA: Project Management Institute, Inc.

PMI. (2017b). *Project Management Job Growth and Talent Gap Report*. Retrieved from Newtown Square, PA, USA: https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/job-growth-report.pdf

PMI. (2021). *A Guide to the Project Management Body of Knowledge (PMBOK Guide)* (7th ed.). Newtown Square, Pennsylvania, USA: Project Management Institute, Inc.

PMI. (2022). Project Management Institute. Retrieved 2022-05-13 from http://www.pmi.org

Popovic, J., Bojic, D., & Korolija, N. (2015). Analysis of Task Effort Estimation Accuracy Based on Use Case Point Size. *IET Software, 9*(6), 166-173. doi:10.1049/iet-sen.2014.0254

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Boston, USA: Addison-Wesley.

Port, D., & Boehm, B. (2001, February 19-21). *Using a Model Framework in Developing and Delivering a Family of Software Engineering Project Courses.* Paper presented at the Proceedings of the 14th Conference on Software Engineering Education and Training (CSEE&T), Charlotte, North Carolina, USA.

Port, D., & Klappholz, D. (2004, 1-3 March 2004). *Empirical Research in the Software Engineering Classroom.* Paper presented at the 17th Conference on Software Engineering Education and Training (CSEE&T 2014), Norfolk, Virginia, USA.

Powell, W., Powell, P., & Weenk, W. (2003). *Project-Led Engineering Education* (1st ed.). Netherland: Lemma Publishers.

Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach* (7th ed.). New York, USA: McGraw-Hill.

Pressman, R. S., & Maxim, B. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). New York, USA: McGraw-Hill Education.

Punter, T., Ciolkowski, M., Freimut, B., & John, I. (2003, 30 Sept.-1 Oct. 2003). *Conducting on-line Surveys in Software Engineering.* Paper presented at the 2003 International Symposium on Empirical Software Engineering (ISESE 2003), Rio de Janeiro, Brazil.

Rajankrupa, C., & Srinath, M. V. (2015). Effort Estimation on e-learning Projects Using Function Point Analysis. *International Journal of Applied Engineering Research, 10*(13), 33888-33892. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84940204142&partnerID=40&md5=67255a12897902af242b97cdde323729

Rak, K., Car, Ž., & Lovrek, I. (2018). Effort Estimation Model for Software Development Projects Based on Use Case Reuse. *Journal of Software: Evolution and Process, 31*(2), 1-17. doi:doi:10.1002/smr.2119

Ram, D. J., & Raju, S. V. G. K. (2000, Oct. 30-31). *Object Oriented Design Function Points.* Paper presented at the First Asia-Pacific Conference on Quality Software, Hong Kong, China.

Raz, T., Shenhar, A. J., & Dvir, D. (2002). Risk Management, Project Success, and Technological Uncertainty. *R&D Management, 32*(2), 101-109. doi:10.1111/1467-9310.00243

Reifer, D. J. (2003). XP and the CMM. *Ieee Software, 20*(3), 14-15. doi:10.1109/MS.2003.1196314

Retzer, S., Fisher, J., & Lamp, J. (2003). *Information Systems and Business Informatics: An Australian German Comparison.* Paper presented at the 14th Australasian Conference on Information Systems, Perth, Western Australia.

Roche, J. M. (1994). Software Metrics and Measurement Principles. *SIGSOFT Softw. Eng. Notes, 19*(1), 77–85. doi:10.1145/181610.181625

Rombach, D. (2000). *Fraunhofer: the German Model for Applied Research and Technology Transfer.* Paper presented at the 2000 International Conference on Software Engineering, Limerick, Ireland.

Rosa, W., & Wallshein, C. (2017, June 12-15). *Software Effort Estimation Models for Contract Cost Proposal Evaluation.* Paper presented at the 2017 ICEAA Professional Development & Training Workshop, Phoenix, Arizona, USA.

Rosenthal, R. (1994). Science and Ethics in Conducting, Analyzing, and Reporting Psychological Research. *Psychological Science (Wiley-Blackwell), 5*(3), 127-134. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=8561061&site=ehost-live&scope=site

Ross, A., & Francis, D. (2003). Lean is Not Enough. *Manufacturing Engineer, 82*(4), 14-17. doi:10.1049/me:20030402

Ross, M. (2003). *Size Does Matter: Continuous Size Estimating and Tracking*. White Paper. Quantitative Software Management, Inc. Glendale. Retrieved from https://www.qsm.com/qw99cse.pdf

Royce, W. (1970). Managing the Development of Large Software Systems. *The proceedings of the WESCON, IEEE CS Press, San Francisco*, 328-339. Retrieved from https://www.praxisframework.org/files/royce1970.pdf

Safrizal, Warnars, H. L. H. S., Gaol, F. L., & Abdurachman, E. (2017, Nov. 20-22). *Use Case Point as Software Size Measurement with Study Case of Academic Information System.* Paper presented at the 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), Phuket, Thailand.

Santos, E. D., & Oliveira, S. R. B. (2018). *Gamification and Evaluation the Use of the Function Points Analysis Technique in Software Quality Subjects: The Experimental Studies*. Paper presented at the Proceedings of the 17th Brazilian Symposium on Software Quality, Curitiba, Brazil.

Santos, E. D., & Oliveira, S. R. B. (2019, July 26 - 28). *The Use of Game Elements and Scenarios for Teaching and Learning the Function Point Analysis Technique: A Experimental Study.* Paper presented at the 14th International Conference on Software Technologies (ICSOFT 2019), Prague, Czech Republic.

Schneider, G., & Winters, J. P. (2001). *Applying Use Cases: A Practical Guide* (2nd ed.): Addison-Wesley.

Schwaber, K. (1995). *SCRUM Development Process.* Paper presented at the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA' 95), Austin, Texas, USA.

Schwaber, K. (2004). *Agile Project Management with Scrum* (1st ed.). Washington, USA: Microsoft Press.

Schwaber, K., & Beedle, M. (2002). *Agile Software Development with Scrum*. Upper Saddle River, New Jersey, USA: Prentice-Hall.

SEI. (2010). *CMMI® for Development, Version 1.3* (CMU/SEI-2010-TR-033 ESC-TR-2010-033). Retrieved from Pittsburgh: http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9661

Selby, R. (1993). Software Measurement and Experimentation Frameworks, Mechanisms, and Infrastructure. In H. D. Rombach, V. Basili, & R. Selby (Eds.), *Experimental Software Engineering Issues: Critical Assessment and Future Directions* (Vol. 706, pp. 87-106): Springer Berlin Heidelberg.

SEMAG. (2001). Software-based Information Systems Engineering and Management Group. Retrieved 2021-07-27 from https://sites.google.com/a/dsi.uminho.pt/semag/

SERG. (2021). Software Engineering Research Group. Retrieved 2021-07-15 from http://serg.cs.lth.se/

Shah, J., Kama, N., & Ismail, S. A. (2018). *An Empirical Study with Function Point Analysis for Software Development Phase Method.* Paper presented at the 7th International Conference on Software and Information Engineering (ICSIE 2018), Cairo, Egypt.

Shaw, M. (2002). What Makes Good Research in Software Engineering? *International Journal on Software Tools for Technology Transfer (STTT), 4*(1), 1-7. doi:10.1007/s10009-002-0083-4

Shere, K. D. (1988). *Software Engineering Management*. Upper Saddle River, New Jersey, USA: Prentice Hall.

Shull, F., Basili, V., Carver, J., Maldonado, J. C., Travassos, G. H., Mendonca, M., & Fabbri, S. (2002, October 3-4). *Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem.* Paper presented at the International Symposium on Empirical Software Engineering (ISESE 2002), Nara, Japan.

Shull, F., Carver, J., & Travassos, G. H. (2001). *An Empirical Methodology for Introducing Software Processes*. Paper presented at the Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, Vienna, Austria.

Shull, F., Mendoncça, M. G., Basili, V., Carver, J., Maldonado, J. C., Fabbri, S., . . . Ferreira, M. C. (2004). Knowledge-Sharing Issues in Experimental Software Engineering. *Empirical Software Engineering, 9*(1), 111-137. doi:10.1023/b:emse.0000013516.80487.33

Silhavy, R., Silhavy, P., & Prokopova, Z. (2018). Evaluating Subset Selection Methods for Use Case Points Estimation. *Information and Software Technology, 97*, 1-9. doi:https://doi.org/10.1016/j.infsof.2017.12.009

Simula. (2021). Simula Research Laboratory, Norway. Retrieved 2021-07-14 from https://www.simula.no/

Siqueira, F. L., Barbaran, G. M. C., & Becerra, J. L. R. (2008, April 14-17). *A Software Factory for Education in Software Engineering.* Paper presented at the Proceedings of the 21st Conference on Software Engineering Education & Training (CSEE&T), Charleston, South Carolina, USA.

Sjøberg, D. I. K., Anda, B., Arisholm, E., Dyba, T., Jorgensen, M., Karahasanovic, A., . . . Vokac, M. (2002, 3-4 October). *Conducting Realistic Experiments in Software Engineering.* Paper presented at the ISESE' 02, Nara, Japan.

Sjøberg, D. I. K., Hannay, J. E., Hansen, O., Kampenes, V. B., Karahasanovic, A., Liborg, N. K., & Rekdal, A. C. (2005). A Survey of Controlled Experiments in Software Eengineering. *IEEE Transactions on Software Engineering, 31*(9), 733-753.

Slaten, K. M., Droujkova, M., Berenson, S. B., Williams, L., & Layman, L. (2005, July 24-29). *Undergraduate Student Perceptions of Pair Programming and Agile Software Methodologies: Verifying a Model of Social Interaction.* Paper presented at the Proceedings of the Agile Development Conference (ADC'05), Denver, Colorado, USA.

Smith, J. (1999). *The Estimation of Effort Based on Use Cases*. Rational Software Corporation. USA.

Software Factory. (2021). Department of Computer Science of the University of Helsinki. Retrieved 2021-07-11 from http://www.softwarefactory.cc/

Sommerville, I. (2016). *Software Engineering (Global Edition)* (10th ed.). Harlow, England: Pearson Education Limited.

Souza, G. C. (2016). *Gestão de Risco em Projetos Académicos de TI: Estudo de Caso.* (MSc). Universidade do Minho, Guimarães, Portugal. Retrieved from http://hdl.handle.net/1822/54891

Stapleton, J. (2003). *DSDM: Business Focused Development* (2nd ed.): Addison-Wesley.

Stiller, E., & LeBlanc, C. (2002). Effective Software Engineering Pedagogy. *Journal of Computing Sciences in Colleges, 17*(6), 124-134.

Suri, D., & Sebern, M. J. (2004, March 1-3). *Incorporating Software Process in an Undergraduate Software Engineering Curriculum: Challenges and Rewards.* Paper presented at the Proceedings of the 17th Conference on Software Engineering Education and Training (CSEE&T), Norfolk, USA.

Svahnberg, M., Aurum, A., & Wohlin, C. (2008, 9-10 October). *Using Students as Subjects - an Empirical Evaluation.* Paper presented at the ESEM 2008, Kaiserslautern, Germany.

Svinicki, M., & McKeachie, W. J. (2014). *McKeachie's Teaching Tips: Strategies, Research, and Theory for College and University Teachers* (14th ed.). Wadsworth, USA: Cengage Learning.

Symons, C. (2020). *COSMIC - A Guide to Software Size Measurement*. Retrieved from https://cosmic-sizing.org/wp-content/uploads/2020/08/Measuring-software-size-v1.0-August-2020.pdf

Symons, C. R. (1988). Function Point Analysis: Difficulties and Improvements. *Software Engineering, IEEE Transactions on, 14*(1), 2-11. doi:10.1109/32.4618

Symons, C. R. (1991). *Software Sizing and Estimating: MK II FPA (Function Point Analysis)* (1st Edition ed.). N. Y., USA: John Wiley & Sons.

Tatnall, A., & Burgess, S. (2009). Evolution of Information Systems Curriculum in an Australian University over the Last Twenty-Five Years. In A. Tatnall & A. Jones (Eds.), *Education and Technology for a Better World* (Vol. 302, pp. 238-246): Springer Berlin Heidelberg.

Teamwork Project Manager. (2021). Teamwork. Retrieved 2021-09-24 from http://www.teamworkpm.net/

The Standish Group International. (2015). *Chaos Report 2015*. Retrieved from https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

Tichenor, C. B. (1997, August 11-15). *The Internal Revenue Service Function Point Analysis Program: a Brief.* Paper presented at the Twenty-First Annual International Computer Software and Applications Conference (COMPSAC '97), Washington, DC, USA.

Tichy, W. F. (1998). Should Computer Scientists Experiment More? *Computer, 31*(5), 32-40.

Tichy, W. F. (2000). Hints for Reviewing Empirical Work in Software Engineering. *Empirical Software Engineering, 5*(4), 309-312. doi:10.1023/a:1009844119158

Tichy, W. F., Lukowicz, P., Prechelt, L., & Heinz, E. A. (1995). Experimental Evaluation in Computer Science: A Quantitative Study. *Journal of Systems and Software, 28*(1), 9-18. doi:10.1016/0164-1212(94)00111-y

Tockey, S. (1997). A Missing Link in Software Engineering. *IEEE Software, 14*(6), 31-36.

Travassos, G. H., & Barros, M. d. O. (2003). *Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering*. Paper presented at the 2nd Workshop on Empirical Software Engineering (WSESE'03) - The Future of Empirical Studies in Software Engineering, Rome, Italy.

Trendowicz, A., Heidrich, J., & Shintani, K. (2011, November 3-4). *Aligning Software Projects with Business Objectives.* Paper presented at the Proceedings of the 2011 Joint Conference of the 21st International Workshop on

Software Measurement and the 6th International Conference on Software Process and Product Measurement (IWSM-MENSURA), Nara, Japan.

Turner, J. R. (2009). *The Handbook of Project-based Management: Leading Strategic Change in Organizations* (3rd ed.). London, United Kingdom: McGraw-Hill.

Uemura, T., Kusumoto, S., & Inoue, K. (1999). *Function Point Measurement Tool for UML Design Specification.* Paper presented at the Sixth International Software Metrics Symposium (ISMS 1999), Boca Raton, FL, USA.

Visaggio, G. (2008). Empirical Experimentation in Software Engineering. In A. D. Lucia, F. Ferrucci, G. Tortora, & M. Tucci (Eds.), *Emerging Methods, Technologies and Process Management in Software Engineering* (pp. 227). Hoboken, New Jersey: John Wiley & Sons, Inc.

Whitmire, S. (1992). *3D Function Points: Scientific and Real-time Extensions to Function Points*. Paper presented at the Pacific Northwest Software Quality Conf., Portland, USA.

Whitmire, S. A. (1993). Applying Function Points to Object-oriented Software Models. In *Software engineering productivity handbook* (pp. 229-244): McGraw-Hill.

Williams, L., & Cockburn, A. (2003). Agile Software Development: It's About Feedback and Change. *Computer, 36*(6), 39-43. Retrieved from 10.1109/MC.2003.1204373

Wohlin, C. (1997, April 13-16). *Meeting the Challenge of Large-scale Software Development in an Educational Environment.* Paper presented at the Proceedings of the 10th Conference on Software Engineering Education and Training (CSEE&T), Virginia Beach, Virginia, USA.

Wohlin, C., Höst, M., & Henningsson, K. (2003). Empirical Research Methods in Software Engineering. In R. Conradi & A. Wang (Eds.), *Empirical Methods and Studies in Software Engineering* (Vol. 2765, pp. 7-23): Springer, Berlin, Heidelberg.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*: Springer-Verlag Berlin Heidelberg.

Wood, M., Daly, J., Miller, J., & Roper, M. (1999). Multi-method Research: An Empirical Investigation of Object-oriented Technology. *Journal of Systems and Software, 48*(1), 13-26. doi:10.1016/s0164-1212(99)00042-4

Wysocki, R. K. (2010). *Adaptive Project Framework: Managing Complexity in the Face of Uncertainty* (1st ed.). Boston, USA: Addison-Wesley.

Yavari, Y., Afsharchi, M., & Karami, M. (2011, Dec. 13-14). *Software Complexity Level Determination Using Software Effort Estimation Use Case Points Metrics.* Paper presented at the 5th Malaysian Conference Software Engineering (MySEC 2011), Johor Bahru, Malaysia.

Yin, R. K. (2014). *Case Study Research: Design and Methods* (5th ed. Vol. 5). Thousand Oaks, California, USA: SAGE Publications.

Zelkowitz, M. V., & Wallace, D. R. (1998). Experimental Models for Validating Technology. *Computer, 31*(5), 23-31.

Zelkowitz, M. V., Wallace, D. R., & Binkley, D. W. (2003). Experimental Validation of New Software Technology. In N. Juristo & A. M. Moreno (Eds.), *Lecture Notes on Empirical Software Engineering* (Vol. 12, pp. 229-263). New Jersey, USA: World Scientific Publishing.

Živkovič, A., Rozman, I., & Heričko, M. (2005). Automated Software Size Estimation Based on Function Points Using UML Models. *Information and Software Technology, 47*(13), 881-890. doi:http://dx.doi.org/10.1016/j.infsof.2005.02.008

Zwikael, O., & Globerson, S. (2006). From Critical Success Factors to Critical Success Processes. *International Journal of Production Research, 44*(17), 3433-3449. doi:10.1080/00207540500536921

Zwikael, O., & Sadeh, A. (2007). Planning Effort as an Effective Risk Management Tool. *Journal of Operations Management, 25*(4), 755-767. doi:https://doi.org/10.1016/j.jom.2006.12.001