IT Risks in Measure and Number

Erald Kulk
This research was supported by the Netherlands Organisation for Scientific Research (NWO) under the Jacquard project number:

638.004.405 EQUITY: Exploring Quantifiable Information Technology Yields

ISBN: 978 90 8659 385 9
© Erald Kulk 2009
VRIJE UNIVERSITEIT

IT Risks in Measure and Number

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op woensdag 18 november 2009 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Gerardus Pieter Kulk

geboren te Middelburg
Alea iacta est. Before this page is turned and the reading of the introduction commences, some people need to be thanked.

First of all, my promoter Chris Verhoef and my co-promoter Rob Peters. I am greatly indebted to them for all their guidance through the past four years of research. Their inspiration, knowledge, feedback, discussions and support have enormously aided in achieving this milestone.

Next, I would like to thank the members of the reading committee for their valuable comments and positive feedback. The reading committee for this thesis consists of: prof.dr. Guido Dedene, prof.dr. Wan Fokkink, dr. Hilda Folkerts, and prof.dr. Bert Kersten.

Moreover, Jean Kleijnen, Wilmar Hassoldt and the anonymous reviewers are thanked for their important feedback, professional insights, encouraging words and very good suggestions on the research presented here.

Furthermore, I would like to thank Joeri van Hoeve for his preliminary work that was useful for the research presented in Chapter 3, my colleagues at the VU for their feedback, and Elly Lammers for all her knowledge on organizing conferences.

Certainly, I want to thank all family, especially my wife Margo, and all my friends for their support throughout the past four years during which I ventured the project that resulted in this thesis.

Erald Kulk
Summer 2009
## Contents

**Preface**

**1 Introduction**

1.1 Production for ages ........................................... 1  
1.2 Quantifying Risk Appetites ................................. 2  
1.3 Research Rationale ......................................... 2  
1.4 Exemplary IT projects ...................................... 3  
1.5 Exploring Quantifiable IT Yields ......................... 3  
1.6 Research question ......................................... 3  
1.7 Zen and the art of changing specifications .......... 4  
1.8 Behind the scenes of estimation errors .............. 5  
1.9 Origin of the chapters ..................................... 6  

**2 Quantifying Requirements Volatility Effects**

2.1 Introduction .................................................. 7  
2.1.1 Overview .................................................. 11  
2.2 Creep, Scrap and Churn .................................... 12  
2.3 Modeling volatility as compound interest ............ 13  
2.3.1 Compound requirements ................................ 15  
2.3.2 Determining requirements volatility ............... 16  
2.3.3 Sizing with function points and lines of code ...... 17  
2.3.4 Related work ............................................. 17  
2.4 Doing the math .............................................. 19  
2.4.1 Lambert’s W function ................................. 23  
2.4.2 Tolerance factor $p$ .................................... 26  
2.4.3 Requirements volatility metric: the $\pi$-ratio .... 29  
2.4.4 Requirements volatility metric: $\rho$-ratio ......... 29
3.6 Goodness of fit and predictive power ........................................ 113
  3.6.1 Goodness of fit ............................................................. 113
  3.6.2 Predictive power of the cost misestimation risk models .......... 120
3.7 Separating misestimation into overrun and underrun ............... 123
3.8 Conclusions ................................................................. 124

Samenvatting (summary in Dutch) 127

Bibliography 133
1.1 Production for ages

In economics the traditional production factors are natural resources, labor and capital [105]. The first additional most important production factor to combine these factors in a profitable way into produced goods is certainly entrepreneurship. If one owns enough land for growing crops and labor and tools like hoes and scythes are abundant for planting the seeds and harvest the crops, this needs to be done in the right seasons so value is added to the seeds and profit can be made on selling the crops. In this traditional setting, more profit can be made if one owns more land, or cheaper laborers or more sustainable tools.

But, entrepreneurship involves taking risks. When seeds are planted, the farmer does not know whether there will be enough rain and sunshine for growing the seeds. Of course, the farmer can aid the growing process by adding more capital to the growing process, for instance a watering can, combined with more resources, water or even soluble fertilizers, but this will cut back the eventual profit because of the costs of the water and watering can. Doing so, the farmer mitigates the risk of dead or not fully grown plants that probably have little or no value. Another risk of the farmer is illness of employees, if on harvest day all employees are in bed with influenza, the unharvested crops might rot on the land. And when the handle of the hoe breaks due to intensive usage new capital needs to be attracted for replacing the broken handle.

Then the question arises, how many days of illness strikes a regular employee during the harvest season on average? And what is the chance that a handle breaks and needs to be replaced? Which brand of fertilizer influences the growth of the crop significantly? And at what point is the number of plants too large to harvest with the current number of employees?
1.2 Quantifying Risk Appetites

Usually, these operational and strategic questions are tackled with historic data. A smart entrepreneur keeps track of yearly turnovers, production rates, replaced tools, etc. If one is new in a region of market and no proprietary data is available, the neighbor can be inquired about last year’s raining days, thus obtaining benchmark figures. Modern day Internet even provides a lot of weather and demographic information directly on-line.

Having historical data in your hands, risks can be calculated and the drivers of the risk can be identified and the risk mitigated. The calculated risk is then taken by the entrepreneur, avoiding sudden jack-in-the-boxes. Identifying risks is a good first step in cautious entrepreneurship, but calculating the chance that a risk materializes is an important second.

An obvious example where risks were not properly quantified is the recent financial crisis. Suddenly unstable portfolios of securities backed with subprime mortgages lost their values. Many banks around the world had to amortize parts of their equities, resulting in an unstable global financial market. Nowadays banks do realize that they were not fully aware of the risks of certain securities that resulted in the financial crisis.

1.3 Research Rationale

For the last decades software has become an important production factor as well. Many production processes depend on different software systems. Systems that support decisions or even make decisions and produce without human intervention. Software often helps in reducing the workload of humans; ranging from placing a box on a certain shelf in a large warehouse to searching for the file of a certain patient in a doctor’s database. In software engineering the same questions emerge as in agriculture. Projects involving the production of software systems are often announced with a flourish of trumpets detailing the improvements the project will result as soon as the project is finished. Promising faster travel from A to B due to improved train schedules or fewer lost suitcases because of a more efficient luggage control system on the airport. Too often these projects are later encountered on newspaper front pages because the projects are costing too much, taking too long and not delivering the desired functionality. Getting a grip on these important key performance indicators (KPIs) will at least help in understanding where and when a project derailed from its tracks. But what common methods and metrics can be used for project portfolios with considerable IT components? At what point can a project be marked as never ending or derailed? What are the industry benchmarks for projects of a certain size? Why do certain projects cost more than estimated? Which metrics need to be included if an IT dashboard is set up in a company or governmental organization? How can an IT portfolio manager take calculated risks? All important questions needing answers.

As in other research areas, measuring is key to take control over a project portfolio. For proper analysis mathematical hurdles need to be covered. And if no data set is available and historical data is urgently necessary, industry benchmarks are necessary.
1.4 Exemplary IT projects

Projects that did not meet their estimated costs, duration or schedule proliferate newspapers around the world. A well known case of a never ending project is the FAA case, described in more detail in Chapter 2. The project concerns the modernization of the US Air Traffic Control systems. After more than two decades of schedule delays and changing and growing requirements, this project was still on the US Government Accountability Office’s list of high-risk programs, where it has been listed since 1995 [32, 2, 35, 31]. What tools or metrics should be used for monitoring the requirements volatility of such projects? Another infamous example is the Denver Airport’s automated baggage handling system supposed to reduce flight delays. The system was delivered partly, two years later and with a huge cost overrun, and even misplacing a considerable percentage of the luggage that entered the handling system [71]. What silver bullet can tame these werewolves of teeming cost, duration and functionality overruns?

1.5 Exploring Quantifiable IT Yields

The research project EQUITY studies connections between production yields and information technology, so that production with software will be enabled in a calculated manner. Decisions whether or not to invest in more or different software projects need to be deprived of gut feeling and nourished with reason based on sound information. EQUITY aims to trace what the actual contribution is of IT to the creation and destruction of value. In this quest for quantification the Bermuda Triangle of project management plays an important role: costs, duration and functionality. What environment variables are important factors in the over- and underrunning of estimates of these IT KPIs? If a project manager focuses on one of these three KPIs, the control over the other two is easily lost. For instance, if the costs are tried to be kept low, not all expected functionality is delivered as the project runs out of funding too soon. Or, the project takes more time than planned if the developers who are assigned to the project are not experienced enough, because of a cost cutting effort.

1.6 Research question

This thesis is the product of research in this field of quantifiable IT KPIs. The focus in this thesis is on the quantification of IT risks: how to avoid unacceptable risks, keep projects on track and discover irregularities as early as possible. More detailed questions are as follows. At what point is a project unbalanced by creeping requirements in such a way that it tumbles into the tar pit of unmanageable projects? What are the risk drivers that influence the disparity between the expected and actual project costs? How can risky projects be identified as early in the project life cycle as possible? To answer these important everyday questions of IT portfolio and project managers the necessary mathematical hurdles are jumped. To obtain usable real-world answers to these questions, real-world data is used as input. Project information from different organizations in different industries is used in the analyses. The real-world data shows what IT risks are to be expected
and helps in future decision making. Feedback from the organizations is used in both analyses for better understanding and explaining the results that were obtained during the research. The research presented here resulted in both new mathematical methods for modeling IT risks as well as obtaining improved industry benchmarks. As they are based on real-world data, these results are valuable for companies keen on improving their IT portfolio management practice.

1.7 Zen and the art of changing specifications

If the farmer from the first paragraph stores the profits from selling crops in a savings account with a reasonable 5% interest per year, how many years does he have to wait for his money to double? If the farmer buys 2% additional land every six months, after how many months are the granaries too small to fit the harvests and the lands too large to plow and the farmer loses control?

The second chapter of this thesis shows how to quantify the rate of increase or decrease of the specifications of an IT system during its construction period. We refer to this phenomenon as requirements volatility. The requirements of IT projects often change during the execution. Sometimes because the clients have new desires, sometimes through changing legislation, but also because of restricted budgets. To illustrate how to quantify the requirements volatility and its effects, real-world data from different industries were used.

In an organization operating in the financial services industry a low-risk IT subportfolio of 84 IT projects is identified comprising together 16,500 function points, each project varying in size and duration. For these projects the volatility of the requirements was quantified by using the available function point countings and the duration between the countings.

The compound monthly volatility rate, which is analogue to the compound interest model in banking, is used as the requirements volatility metric. This metric was coined by Jones in 1996 [60], we improved the metric by incorporating project duration. The volatility figures that were found were aggregated into a requirements volatility benchmark.

Currently known industrial averages only consider the growth rate, not the period during which the growth rate enlarges a project’s requirements. In Chapter 2 it is explained that the maximum tolerable requirements volatility rates depend on both size and duration. For instance, a monthly growth rate of 5% is commonly considered a critical failure factor, but in the low-risk portfolio more than 21% of the successful projects has a volatility larger than 5%. A mathematical model is proposed taking into account size and duration thus providing a maximum healthy volatility rate that is more in line with the reality of low-risk IT portfolios. Based on the model, a tolerance factor is proposed expressing the maximal volatility tolerance for a project or portfolio. Two volatility ratios are derived from this model, the \( \pi \)-ratio and the \( \rho \)-ratio. These ratios express how close the volatility of a project has approached the zone of critical failure rates. For the low-risk portfolio under consideration the empirically found tolerance is acceptable, and values exceeding this tolerance level are useful to trigger IT decision makers.
In a second case the volatility data of a governmental IT portfolio were juxtaposed to the financial services industry benchmark, immediately exposing a problematic project, which was corroborated by its actual failure. If function points are less common, e.g. in the embedded industry, daily source code size measures are used and it is illustrated how to govern the volatility of a software product line of a hardware manufacturer. This third case is discussed in detail as well.

With the three real-world portfolios it is shown that the results serve the purpose of an early warning system for projects that are bound to fail due to excessive volatility. Moreover, essential requirements volatility metrics were developed that belong on an IT governance dashboard and such a volatility dashboard is also presented.

1.8 Behind the scenes of estimation errors

Another earlier mentioned question of the farmer is which fertilizer would best aid the growing process of seeded crops. To answer that question the farmer would need data on different land areas where different fertilizers have been used and the amount of full grown plants. Other factors that could have influenced the growing process need to be included in the analysis like hours of sun, millimeters of rain, etc. The results of the analysis are used in the next year for influencing the growth of the crops.

In chapter 3 a statistical method is proposed for quantifying the impact of factors that influence the quality of cost estimation for IT-enabled business projects. We call these factors risk drivers if they can be influenced by project management before or in the early stage of a project. The explained method can effortlessly be transposed for usage with other important IT key performance indicators, such as schedule misestimation or functionality underdelivery.

Logistic regression is used as a modeling technique for estimating the quantitative impact of risk factors. This was done so, because logistic regression has been applied successfully in the field of, among others, medical science, e.g. perinatal epidemiology, for answering questions that show a striking resemblance to the questions regarding project-risk management. In our study a data set of a large organization in the financial services industry is used for assessing the applicability of logistic modeling in quantifying IT estimation risks. This research has shown that it is possible to properly quantify IT estimation risks, even with the help of crude data. With a real-world example it is illustrated how to scrutinize the quality and plausibility of the available data.

We also explain how to deal with risk factors that cannot be influenced by project management, but have an influence on the outcome of the estimation process. The detection of influential risk factors using logistic regression is demonstrated with a real-world data set. After a discussion of the interpretation of the found models it is showed that the findings are helpful in decision making on measures to be taken for reducing the chance of misestimation and thus mitigate IT risks for individual projects.

The analyses reveal that projects must not be overstaffed and the ratio of external developers must be kept small for obtaining better cost estimates. The research also shows that business units that report on financial information tend to be risk mitigating, because they have more cost underruns as opposed to business units without reporting that are
CHAPTER 1. INTRODUCTION

Risk seeking, the latter displaying more cost overruns. Moreover, a maturity mismatch is discovered; an increase from the IT maturity level CMM 1 to CMM 2 does not influence, although expected, the cost misestimations. This missing improvement in estimating is explained by the fact that the maturity of the business is not increased alongside with the IT department resulting in a maturity mismatch not affecting the quality of cost estimates.

The research findings are also valuable for increasing the efficiency of the auditing process. The found cost misestimation models are used for classifying the projects in risky projects and non-risky projects before the start of auditing.

1.9 Origin of the chapters

Parts of this Ph.D. thesis have been published previously. In this section the different origins are listed.

Chapter 2 on requirements volatility effects has been published in Elsevier’s journal Science of Computer Programming. It was included in the third issue of volume 72 [74].

Chapter 3 has been accepted and will appear in Elsevier’s Science of Computer Programming [73] as well.

A part of the summary in Dutch has been published in the Jaarboek ICT enSamenleving: Omzien naar de toekomst [38].
Quantifying Requirements Volatility Effects

The creation of software requirements is reminiscent of hiking in a fog that is gradually lifting.
~T. Capers Jones [64]

2.1 Introduction

Software is ubiquitous [74], it has penetrated our society into all its capillaries. Since our society is continuously evolving and changing its demands, the foundations on which it stands have to adapt to the inevitable and often abrupt changes. Therefore, software evolves, and not only after it has been delivered, but also during the development phase. An illustrative example of evolving software is internet banking. Where most banking systems were developed in COBOL in the sixties and seventies of the 20th century when the World Wide Web did not even exist, the need for Internet banking arose with the increasing popularity of Internet in the nineties of the same century. In an era when object-oriented languages gained popularity, these new on-line banking services, mostly written in object-oriented languages, had to be connected to the existing and aging, but above all omnipresent [7], procedural COBOL systems. Over a time span of 40 years the banking systems had evolved, COBOL systems were maintained and updated, and linked to new systems, thus adapting to the needs of clients.

Requirements To impose boundaries on a project and to create consensus about its scope, the stakeholders’ wishes and desires are translated into requirements. Requirements engineering is not something new and many books have been written on this subject, some well-known being: [28, 40, 55, 72, 98, 99, 107, 127, 130]. Weinberg succinctly states the rationale for requirements [127]:

Requirements are made for a common purpose: to change vague desires into explicit and unambiguous statements of what the customers want.
CHAPTER 2. QUANTIFYING REQUIREMENTS VOLATILITY EFFECTS

But, before the ink of the requirements document is dry and the design phase has commenced, the first meetings generate questions and new insights that immediately influence the decisions just taken and the agreed upon requirements. To continue Weinberg:

These statements are then used to compare what was built and what was desired—the fundamental measurement of any feedback controlled process.

During a feedback controlled software development process the requirements that were initially frozen are stretched until a desired and satisfying scope is reached. Oftentimes, budget, schedule and requirements are not in alignment. We want it all, we want it yesterday and we want if for free. As a consequence, requirements are pressed together to fit into a certain project budget or parts of the requirements are hewed away from the initial scope. Requirements volatility is a fact of life, and therefore important to manage. Ideally this requirements stretching, squeezing and hewing is kept within certain bounds. But what bounds are acceptable? Some requirements volatility will be necessary and healthy, but burgeoning requirements or the opposite, requirements suffering from severe scrap can cause a project to get out of sight and off track. Continuously changing requirements can drive project management crazy, make clients angry, developers irritated, budget holders disappointed and everybody distressed. Then again, we encountered projects for which a high volatility of the requirements had a positive influence on productivity. However, these are exceptions when volatility is supported by a development process and tools.

Creep angst  Weinberg [127] illustrates the explosive growth of myriads of requirements with the project that is afraid to finish, an initially six month project turned out to be dragging on for two years. The end-date was continuously postponed, keeping the project from finishing. One of the main problems stated in Weinberg’s example was that management failed to keep requirements volatility under control. Jones [60] confirms Weinberg with his statement:

One of the most chronic problems in software development is the fact that application requirements are almost never stable and fixed.

Jones further emphasizes the problem with:

Although creeping requirements are troublesome, they are often a technical necessity.

So, requirements volatility is a fact of life, but how to control the volatility? Although Weinberg states that volatility is always a problem leading to projects that do not finish, we found that creep can be healthy when it positively influences productivity or when there is a business case to stretch the requirements. When stakeholders decide to change the requirements after they were agreed upon, and there is a business case to do so [91], all project stakeholders need to be aware of this, so everybody knows what has to be delivered at the end of the project and how much the client is going to be paying for the resulting product. A natural question that arises is how much volatility is healthy and how much volatility is going to create more havoc than solve problems. In this chapter we propose how requirements volatility can be described and quantified with simple statistical methods and how projects with unhealthy volatility can be identified.
**Plans and planes**  To illustrate that even with the best intentions it is very hard to keep requirements volatility under control, we recall a famous real-world case of a project that is at the time of writing still afraid to finish. It concerns the Advanced Automation System (AAS), an initially quoted $4.3 billion, 1.5 million lines of code, 10-year project announced in 1981 by the US Federal Aviation Administration (FAA) [19, 31, 42]. The project concerns modernization of the Air Traffic Control system to:

- meet projected increases in traffic volumes,
- enhance the systems margin of safety, and
- increase the efficiency of the Air Traffic Control system.

In 1981 the FAA announced their plans for this program. After nine years of requirements engineering and design competition, the contract was awarded in 1988 and signed two years later in 1990, almost ten years after the announcement. Two years after commencement, the project schedule was extended by 19 months due to, among others, unresolved differences in system specifications caused by changes to the requirements. In 1994 after an additional 14-month schedule delay the project was declared *out of control* by FAA management. At the time of writing this chapter, after more than two decades of schedule delays and changing and growing requirements, this project is still on the US Government Accountability Office’s list of high-risk programs, where it has been listed since 1995.

What tools or metrics can be used to monitor the requirements volatility of such projects?

**Requirements reality**  We recall Jones who already wrote in 1996 [60] that in reality requirements change. Requirements change is problematic, so executives are tempted to consider an IT enabled business investment ideal when there are no more changes once the requirements phase is finished. After signing off the requirements, the implementation of the project should be without further change or delay. However, according to Kotonya and Sommerville [72, pp.113–114]:

> It is often the case that more than 50% of a system’s requirements will be modified before it is put into production.
and that a recent European survey of 4000 companies found that the management of customer requirements was one of the principal problem areas in software development and production.

Leffingwell states in a summarizing article [77] that between 41% [104] and 56% [114] of all defects can be traced back to errors made during the requirements phase. Robert Glass traced in a small empirical study [41] the origin of 5.5% of the persistent defects to inadequate requirements. Frederick Brooks proclaims in his seminal book on management of computer programming projects ‘The Mythical Man-Month’ that the only constancy is change itself [17]. So, a zero-change policy looks good on management charts, but does not necessarily help in achieving the best possible results. The challenge is to allow for healthy volatility and to prune excessive requirements growth or scrap. Some requirements volatility will certainly lead to a much better end result, whereas high volatility often indicates serious problems. When requirements errors are unveiled, it is sometimes necessary to scrap or grow more rapidly, in order to regain a sound project. In this chapter we will see examples of both healthy and unhealthy volatility.

**Function Points** Not all requirements are equivalent in difficulty. That is why function points are useful when requirements volatility is discussed. Sizing IT projects in function points is a widely used synthetic measure to express the amount of functionality that will be or has been built. Function points are independent of programming languages, and therefore very suitable for cost estimation, comparisons, benchmarking [1, 3, 4, 30, 39] and also for comparing requirements change. Each function point is comparable with another function point counted with the same method. Function points analysis is a certified functional sizing method of IT projects [39]. Function points can be used consistently and with an acceptable degree of accuracy [66, 67, 118]; the intermethod and interrater reliability is sufficiently high to compare function point totals that resulted from more than one counting method or from different counters.

There are different techniques known for conducting function point countings [58, 84]. The International Function Point Users Group provides an ISO standard for function point analysis [53]. It is known that function points can be counted at a rate of about one hundred function points per hour depending on the quality of the requirements counted, or, if divided by a standard rate of $100 per hour, only a dollar per function point. If the quality of the requirements is low, this rate can decrease to 30 or 40 function points per hour. Since lines of code are not directly comparable, for example one line of code can be empty and another can contain a difficult boolean expression, we will convert lines of code to function points later on to keep the size functionality discussion consistent. This conversion to function points will make changes in lines of code comparable in terms of function points.

**Volatility in three environments** We present three case studies on requirements volatility. In a large organization in the bancassurance sector we identified a low-risk IT sub-portfolio for which we were able to quantify its requirements volatility. An IT portfolio

---

1>Bancassurance: an, originally French, portmanteau of banking and assurance [131]
is called low-risk when almost all projects are successful: within time, within budget and delivering the desired functionality at the right quality levels which was the case in the bancassurance portfolio. We start with a low-risk portfolio to know what requirements volatility is acceptable. In our bancassurance portfolio the size of each individual IT project was measured at least twice through function point countings; one counting after initial requirements sign-off and the other at the end of the project. A few projects had even three countings, an additional counting at the end of the design phase. The intermittent function point countings of 84 different bancassurance projects were then combined to analyze the characteristics of the requirements volatility at the IT portfolio level. All these countings were used to calculate the so-called volatility rate, or compound monthly requirements volatility rate to analyze the requirements volatility of projects from the entire IT portfolio. The bancassurance portfolio that is used in this chapter is a real-world and known to be low-risk portfolio. The results presented in this chapter can thus be used as a yardstick to benchmark other portfolios. We juxtaposed governmental projects from a known high-risk portfolio, our second case study, with our bancassurance benchmark. The high-risk aspect is defined by large projects with high failure rates. This resulted in the immediate identification of sure-fire failure projects that were unnoticed, proceeding with an unhealthy growth of the requirements. Our third case study is a software product line for embedded software of a hardware manufacturer. By applying backfiring on daily source code volumes we calculated volatility rates and used these to create a volatility benchmark for embedded systems. With our proposed volatility metrics, we easily identified large adaptations in this portfolio and were able to focus directly on potential problems.

2.1.1 Overview

The remainder of this chapter is divided into the following sections.

Taxonomy We start with a taxonomy of requirements volatility in Section 2.2.

Basics and related work In Section 2.4 we explain the basics of the volatility rate of requirements, \( r \), how it can be measured, and we discuss earlier findings of industry averages published by Jones and other related work.

Mathematical background The reader interested in the mathematical background of our models and ratios is referred to Section 2.4.1. This section introduces the model to calculate the maximal healthy requirements volatility for a project and a metric to calculate a project’s tolerance \( p \) for volatility. This section presents also the \( p \)-proportional volatility ratio \( \pi \) to compare projects of different duration and the requirements volatility ratio \( \rho \) to compare the volatility of projects of different duration and size.

Case studies The reader looking for case studies of requirements volatility can find three extensive cases in sections 2.5, 2.6 and 2.9. The fifth section discusses volatility in the bancassurance sector. Here, the analysis of the requirements volatility of a 23.5 million
dollar costing real-world low-risk bancassurance portfolio representing 16,500 function points in 84 IT projects originating from a large bancassurance portfolio is discussed. Section 2.6 analyzes the volatility of high-risk government projects and compares these projects with our bancassurance benchmark created in Section 2.5. Section 2.9 discusses the volatility of a software product line.

In-depth studies More in-depth studies of requirements volatility can be found in Section 2.7 and 2.8. The seventh section dives into volatility variations for in-house and outsourced projects of the bancassurance portfolio. It turned out that outsourced projects displayed similar requirements volatility characteristics as in-house projects. In Section 2.8 we perform a root-cause analysis on the bancassurance portfolio to clarify differences in volatility.

Lines of code Section 2.9 presents instruments how to monitor the volatility of a software product line when function point analyses are not available. We do this in a third case study of a software product line by applying backfiring on daily size measurements of the source code.

Dashboard Practitioners looking for tools that can be applied directly, are helped with Section 2.10. This section introduces a requirements volatility dashboard to visually represent all volatility metrics in tables and graphs.

Conclusions Finally, we summarize and conclude in the last section.

2.2 Creep, Scrap and Churn

The lingo of requirements change has many variations, such as the previously mentioned stretching, hewing and squeezing, or as Weinberg names it [127, 126], requirements leak: projects that are in a state of perpetual pregnancy, never quite giving birth. All these different forms of requirements volatility change the result of a project. We summarize the most well-known terms in the following core glossary for requirements volatility, and abide by these three different forms of requirements change in this chapter. Requirements volatility is defined by any combination of the following three forms of requirements change and is used as a general term for requirements change in this chapter.

Requirements Creep When the scope of a project increases, requirements are added, because additional features surfaced or extra interfaces need to be build. This is called requirements creep, also known as scope creep. Requirements creep can be caused by loosely defined initial requirements, an incomplete analysis when some stakeholders were overlooked or changing legislation during the project. For instance, in the earlier mentioned FAA case [19] additional systems needed to be interfaced with as soon as they became available, whether these were additional satellite systems or on-line available weather information systems, increasing the scope of the system. Even systems that were
developed at the beginning of the project became legacy during the project, creating additional requirements to cope with the emerged legacy.

**Requirements Scrap** Requirements do not always increase, sometimes they decrease when initially stated requirements were too broad and the stated goal could also be reached with fewer requirements. Or sometimes unnecessary requirements can be left out during the project. Due to shrinking budgets or running out of schedule it is also possible that certain parts of the requirements are left out of the current project. Even though deferred or scrapped requirements do not have the same impact on the requirements themselves, they do have the same mathematical negative influence on the number of requirements for the running project. The number of requirements decreases: it is decided to drop certain parts of the requirements to finish the project on time and within budget and keep the project manageable. The FAA announced requirements scrap in 1996. They requested a reduced-functionality Initial Sector Suite System, a key component of the new system, the centerpiece of FAA’s efforts, under a restructured and curtailed program. It was renamed Display System Replacement, DSR, and downsizing was done with the intention to complete deployment in May 2000 [19, 32]. All DSR installations were eventually completed in 2000 [35].

**Requirements Churn** When the size of requirements during the project changes like the bellows of an accordion during a polka, requirements churn is occurring: requirements have been added and removed. For instance, when at the end of the project the size of the project was not different from the size at the beginning, but the requirements have been changed and not been stable throughout the project, this is called requirements churn. Examples of requirements churn are, for example, when colors in an interface need to be changed or buttons should be placed in a different position in an interface. Different functionality is necessary, but it does not influence the size directly.

### 2.3 Modeling volatility as compound interest

Jones [60] introduces a measure to compare the change rates of requirements by calculating compound monthly requirements volatility rates. Jones does not use an average percentage of change of the overall volume, because these numbers can be misleading, and are making it very hard to compare the volatility of different projects or portfolios. An average percentage of change of the overall volume lacks information, namely the time in which the change occurred. The compound monthly requirements volatility rate coined by Jones does express the time aspect. This volatility rate expresses the rate by which the requirements have grown or decreased every month throughout the project. The aspect of compoundness of requirement changes in this rate is illustrated by the following. Consider the requirements elicitation process, whenever new requirements come in, maybe not all, but some requirements added in earlier stages have to be taken into account. Moreover, every requirement that is added will trigger people to introduce other requirements that they did not think of before. When requirements are added later during a project, they will have also a larger impact on productivity than earlier requirements.
and lower the productivity, because more work is put into the same time frame, which is known as time compression [93, 94, 119]. All these aspects are expressed by having a compound monthly volatility rate and not an average linear monthly rate.

Calculating monthly requirements volatility rates, as defined by Jones, is a transposition from the financial world. The time value, or future value of money is in the field of accounting well-known as compound interest or CAGR, short for compound annual growth rate. By transposing from compound growth rate in finance we assume that requirements are compound within a project as we explained above. We will refer back to the financial origin throughout this chapter to help explain requirements volatility.

In finance annual growth rates are very common in interest calculations. A safe way to earn money is by putting a certain amount of money in a savings account at a bank. The amount of money grows when the account is accredited with the bank’s gratefulness for leaving your money in their accounts, i.e. simple interest. If the earned interest is left on the bank account for another interest period the total amount will create more interest than was generated in the first period, i.e. compound interest, or simply money creating more money. The mirrored version of this process would be an unattended debt. When a loan is submitted by a bank, the amount of money that has to be returned to the bank increases with the accumulating interest that the bank charges for having the loan. As we will see shortly, this transposes effortlessly into IT. Therefore, we explain briefly the basics of calculating compound interest.

Making money  If one wants to know beforehand how much a certain amount of money will be worth ten years from now, the future value of this amount that is deposited in a bank account today can be calculated easily with the following well-known accounting formula [101], with \( r \) being the periodical interest rate that is accredited every interval and \( t \) denoting the number of intervals the \( \text{StartAmount} \) stays within the bank, which is usually denoted in years.

\[
\text{FutureAmount} = \text{StartAmount} \cdot (1 + r)^t \tag{2.1}
\]

By applying some standard algebraic manipulations the aforementioned Formula 2.1 can be rewritten to the following equivalent Formula 2.2.

\[
r = \sqrt[\text{Periods}]{\frac{\text{FutureAmount}}{\text{StartAmount}}} - 1 \tag{2.2}
\]

Instead of calculating the future value of a certain amount as can be done with Formula 2.1, the required interest rate is now calculated. Formula 2.2 can be used to calculate the required periodical interest rate from a start amount, a desired future amount and the number of periods the start amount will stay in a bank account. For instance, if we start with $1000 and we would like to have $1500 after ten years, we can calculate from Formula 2.2 that an annual interest rate of approximately 4.14% would be needed to achieve our goal.


2.3.1 Compound requirements

The CAGR originating from accounting transposes effortlessly into information technology on the subject of requirements engineering, as introduced by Jones [60]. Where in finance the compound annual growth rate is used to calculate the future value of money, in requirements engineering the formula is used to calculate the compound monthly volatility rate \( r \) of requirements. Although the formula is common in finance, it is rarely used in IT portfolio management. Therefore, not much related work is found in the literature. Formula 2.3 shows the requirements equivalent of Formula 2.1 and in Formula 2.4 we show the equivalent of Formula 2.2, to calculate the compound monthly volatility rate for a project with a duration of \( t \) months and using a size estimate at the beginning and at the end of the project.

\[
\text{SizeAtEnd} = \text{SizeAtStart} \cdot \left(1 + \frac{r}{100}\right)^t \quad (2.3)
\]

\[
r = \left(\frac{\text{SizeAtEnd}}{\text{SizeAtStart}} - 1\right) \cdot 100 \quad (2.4)
\]

**Jones's averages** Few people provide data about the requirements volatility characteristic, in fact we are only aware of Capers Jones who provides industrial averages in various publications. We will discuss his and other related work in Section 2.3.4. Such historical information is useful and serves when no function point analyses are available in an organization when agreements about requirements volatility are being made.

**Volatility as a control factor** Volatility is an important software control factor since its value gives a strong indication for project success or failure. Just as a financial construction with interest above 30% is almost always suspicious, also IT projects that are highly volatile are often in trouble. This chapter provides intuition to what extent volatility is healthy and when things are signaling further investigation, so that outright failure can be prevented by bringing volatility under control before it is too late. With a little more data than Jones provides it is already possible to obtain more insight into IT projects and to improve their control. In this chapter we discuss three cases. One in which two function point analyses per project were available for many projects in a low-risk portfolio, resulting in a bancassurance benchmark. In the second case a limited number of data points were available for a high-risk governmental portfolio, and the third case describes a low-risk portfolio in the systems industry, for which no function point analyses were available. In the latter case we used the source code volume as a proxy to requirements volatility.

Having multiple project measures available, one can detect also requirements churn when a project was first enlarged and later decreased. This is important, since when the final and initial project size are equal, the project undergoing churn is not completely comparable with an equally sized project without churn. This can be compared to a company with 100 employees when 10 employees are fired and replaced with 10 new employees. The number of employees stays the same, but replacing the employees did cost time and money. In addition to establishing elaborate post-mortem volatility benchmarks, we will
also show how to assess ongoing projects with respect to volatility in Section 2.9.1. High volatility then serves as an early-warning system requiring further qualitative and quantitative investigations. This low-cost assessment can prevent spending huge amounts of money on software development that is almost certainly going to fail due to the alarming value of requirements volatility. For IT projects that have more intermediate size estimates it is possible to detect the derailing IT projects already in an early phase by using our newly proposed metrics.

2.3.2 Determining requirements volatility

To calculate a compound monthly growth rate the size of the requirements has to be determined at least at two different moments in time. With two time-stamped size estimates an overall volatility multiplying factor can be calculated. We now take as a third variable, the duration of the project expressed in months. By using Formula 2.4, the compound monthly requirements volatility rate \( r \), or short volatility rate, can be calculated on any handheld scientific calculator or with a spreadsheet program. The resulting figure expresses the monthly percentage of requirements change. A high requirements volatility rate indicates highly volatile requirements, negative rates often indicate zealous requirements scrapping due to budget constraints or schedule overruns. With only two measurements it is not possible to measure the earlier mentioned requirements churn, since if we have a 1000 function point project and after one month the requirements have increased with some 100 function points and another month later about 100 function points of requirements are withdrawn, the project ends up having a size of 1000 function points. When analyzing the measurements without creep or scrap knowledge of the project itself, it is impossible to measure churn, since from start to end there was no requirements volatility with Formula 2.2. Later on in this chapter we will also show volatility analyses of projects with intermediate countings and of a software product line for which we use daily source code volumes and requirements churn will become visible.

Distribution over time

Although we cannot assume in general that requirements volatility is equally distributed over time, our volatility measure is a reasonable approximation for extensive analysis of requirements volatility. One of the reasons being that when requirements change substantially during a project, this will impact cost, duration and other important Key Performance Indicators (KPIs), making a recount reasonable. So, volatility is perhaps unequally distributed over start and end date, but equally distributed over the intermediate, and shorter, time frames. This also applies if the volatility of the requirements occurred at the beginning of the project or at the end of the project. Since, as we stated earlier, adding requirements at the end of a project seems more costly and appears to have more impact than adding requirements in the beginning of a project when certain design decisions still have to be made [12, page 40], it is therefore important to know at what moment in a project the requirements have been added. Jones [64] suggests to set a sliding scale of costs in software development contracts as a way of dealing with changing user requirements. With this scale, requirements added in later phases of the software process are more expensive than requirements added in earlier phases. It is therefore recommended to also have a look at absolute differences of several countings
or requirements volatility over time during a project. But this would require more than two function point countings and function point analyses cost time and money, so it is not realistic to continuously demand these. It is also probable that requirements growth will lead to deadline extension or time compression.

2.3.3 Sizing with function points and lines of code

In this study we analyze the volatility of finalized projects for which we calculate the requirements volatility based on size estimates and final project duration. We do not consider project durations that were only stated in project planning. Since we are using real industry data from finished projects, we are able to create volatility benchmarks. Subsequently, we will use these benchmarks to analyze the volatility of ongoing projects when early size estimates have been made, and we use our benchmarks to compare between various industries, in our case the volatility of a software product line by backfiring source code sizes. Size estimates in all volatility calculations are expressed in function points. For two portfolios function point analysis was used for size estimating. Intermediate countings in the bancassurance portfolio were only performed when large volatility was expected. Apart from this kind of analysis, one can also conduct daily size measurements. We do this by translating the lines of code back to function points, a technique called backfiring. In Section 2.9.1 we give an example of how we were able to closely monitor the volatility of an IT portfolio comprised of a software product line for similar, but different, embedded systems, each system containing millions of lines of code. The function points under scrutiny in the bancassurance portfolio are a subset of a portfolio in which function points were counted consistently. This result was inferred in another paper \[118\]. In brief, boxplots and Kolmogorov-Smirnov tests were used in that paper to test the validity of the function point counting data. Moreover, 10% of the function points were recounted by an independent certified function point analyst. No significant deviations were detected. For more information on this audit we refer the interested reader to \[118\].

**DSDM** A method to manage requirements is the MoSCoW principle from the DSDM method \[109, 108\] short for Dynamic Systems Development Method. This method categorizes the requirements into must haves (M), should haves (S), could haves (C) and want to have, but won’t have for this project (W). The capitals jointly form the acronym MoSCoW. The projects in the analyzed bancassurance portfolio have utilized this method. The usage of MoSCoW in the bancassurance portfolio contributes in keeping the high volatility rates, that we encounter later on, healthy by creating small subprojects and only doing the projects that are most important.

2.3.4 Related work

There are papers originating from different fields of science, among others agriculture, biomedicine and astronomy, stating that requirements creep is inevitable and that excessive requirements creep is a turbulence or even a failure factor for IT projects. In those papers it is advocated that requirements creep should be avoided or managed \[6, 123, 69, 81, 9, 37, 68, 124, 56, 129, 69, 76, 96, 112, 47, 46\]. However, those papers fail to state
how much requirements creep is actually occurring or how much creep is unhealthy and excessive or even how requirements volatility should be monitored. Moreover, only a few publications discuss quantifications of requirements volatility, most of them by counting requirements and not the impact of requirements [26, 100, 46]. In a paper by Zowghi and Nurmuliani [134] in which we do see some data, it is perception data. However, we are not using perception, but actual project data. Their perceptive study finds a negative influence of perceived requirements volatility on project performance. However, we found the opposite as well in our volatility data. Loconsole and Börstler [80, 79] quantify requirements volatility through changes to use case models in a case study in the automotive industry. In that research one project is studied in which only use cases were used to describe requirements, changes were measured as changes to the use case diagram. Their paper is based on a single project comprising fourteen use cases. Our study quantifies the volatility based on function points and lines of code in three different industries totaling over 80 projects.

Other papers [57, 82, 113] stress the importance of measuring requirements volatility to investigate its presumed relation with defect density. These studies suggest that changes to requirements can have a significant effect on defect density. In this chapter we present metrics to measure requirements volatility. They are validated by real-life data and identify excessive change in an early stage.

As we cited Jones earlier, requirements volatility is a technical necessity. Therefore, methods are needed to deal with the apparent volatility. In this section we will review what compound monthly requirements volatility rates have been found in previous research. Since not a lot of industry data about requirements volatility rates are known, we will summarize all the data in the field that we know of. Houston et al. [51] describe a perceptive study on six software development risk factors that had 458 respondents. The respondents perceived that for 60% of their projects, requirements creep was a problem. Requirements creep in this study was mentioned to be a problem when requirements growth exceeded 10%, or caused more than 10% rework. However, we will see in this chapter that you cannot uniformly state that 10% volatility is a problem. We will show that 10% can be healthy, but also unhealthy depending on certain characteristics of the project. In their stochastic model [51], requirements growth is modeled as a continuous flow that increases linearly during the project until it reaches a maximum and then decreases linearly. The presented model does not mention the possibility of requirements scrap, which we did encounter in the portfolios we analyzed in this chapter. Stark et al. [111, 110] discuss the overall volatility of some 40 deliverables, but volatility is calculated as changes in the number of requirements, not taking effort into account. Over 60 percent of the deliverables encountered requirements volatility, with an average volatility of 48%. A qualitative cause analysis model on change request data is discussed by Nurmuliani et al. [86]. The model studies requirements documents, but they study only one project, whereas we base ourselves on the function point analyses of many projects and many data points of ongoing changes to existing systems. Since Nurmuliani et al. study documents and not functionality or lines of code, the impact of change is not quantified.

An article by Anthes [6] mentions that the top reason for requirements creep, in 44% of the cases, is a poor definition of initial requirements. A small analysis on the impact of changes after requirements sign-off on the productivity of maintenance projects
IT Risks in Measure and Number

for military software was done by Henry and Henry [48]. This study reveals decreasing productivity when having small requirement changes due to overhead in processing and documenting changes.

In a paper by Sneed and Brössler [106] the growth statistics of a commercial software package are presented. Although they do not present compound monthly growth rates, it is easy to calculate the volatility rates with the five presented function point sizes and time stamps. The volatility rates are an overall monthly rate of 1.47% over 4 years and for the 4 different intermediate years respectively 1.34%, 2.99%, 0.37% and 1.21%.

Capers Jones [60, 61, 64, 63] states industry averages of monthly requirements growth rates for different types of software systems and projects. The latest averages in his work are summarized in Table 2.1. Besides the averages of requirements change, Jones states encountered maximum volatility rates in [62, Table 7.4, Table 8.5, Table 9.4, Table 10.3, Table 11.4], summarized in the third column of Table 2.1 and in [62, Table 7.9, Table 8.10, Table 9.9, Table 11.9] Jones states maximum requirements stability rates as project failure factors. These failure factors are summarized in the last column of Table 2.1. In Table 2.1 MIS refers to Management Information Systems. The accompanying failure factor is the event that the requirements creep is out of control.

<table>
<thead>
<tr>
<th>Software Type</th>
<th>avg r (%)</th>
<th>max r (%)</th>
<th>out of control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract or outsourced software</td>
<td>1.1%</td>
<td>3.4%</td>
<td>&gt; 5%</td>
</tr>
<tr>
<td>MIS Software</td>
<td>1.2%</td>
<td>5.1%</td>
<td>&gt; 5%</td>
</tr>
<tr>
<td>Systems software</td>
<td>2.0%</td>
<td>4.6%</td>
<td>&gt; 5%</td>
</tr>
<tr>
<td>Military software</td>
<td>2.0%</td>
<td>4.5%</td>
<td>&gt; 15%</td>
</tr>
<tr>
<td>Commercial software</td>
<td>2.5%</td>
<td>6%</td>
<td>–</td>
</tr>
<tr>
<td>Civilian government software</td>
<td>2.5%</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Web-based software</td>
<td>12%</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 2.1: Jones’s industry averages, encountered maximum rates and failure rates in various industries.

The figures in Table 2.1 were derived from function point countings and calculated on the differences between the initial function point size at the completion of the requirements phase and the function point size after completion of the software project. Jones states in [64] also an average creep rate of 12% for agile projects. However, Table 2.1 offers only failure rates independent of project duration. As we will argue shortly, size and duration can change a particular volatility rate from healthy to unhealthy. So, by incorporating them in the quantification of requirements volatility, they help to understand what volatility is healthy and what volatility is not. In the next section we will show the impact of project duration on requirements volatility.

2.4 Doing the math

To provide the reader with an intuition of growing requirements, we give an example in which we use a volatility of 2% per month, which is the median of the second column in Table 2.1 showing average volatility rates and recommended by Jones if you have
no additional information on a project. The 2% is calculated afterwards on a 36-month project with an initial size of 10000 function points. It turns out that twice the original requirements end up—often undocumented—in the final system. Namely, the growth factor for a 36-month project with a compound monthly growth rate of 2%, is \(1.02^{36} \approx 2.04\). For our 10000 function point project this results, when using Formula 2.3, in 10000 \(\times\) 2.04 = 20400 function points after three years. Obviously an unhealthy project, since we have a project that doubles its requirements during development, which can hardly be a healthy project. This was already mentioned by Jones in [60] in which he mentions projects with overall changes of 100 and even 270 percent growth.

Table 2.2 illustrates the total requirements increase encountered afterwards for software projects initially taking one, two or three years, with a growth rate of 2%. Obviously the longer the initial project duration, the larger the chance that it fails due to surging requirements.

<table>
<thead>
<tr>
<th>Year</th>
<th>Requirements increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27 %</td>
</tr>
<tr>
<td>2</td>
<td>61 %</td>
</tr>
<tr>
<td>3</td>
<td>104 %</td>
</tr>
</tbody>
</table>

Table 2.2: Various total requirements increases for a volatility of 2% per month.

Projects having a larger compound monthly growth rate will reach the point of doubled requirements sooner. See Table 2.3 for some typical examples. From Tables 2.2 and 2.3 we see that the point of requirements doubling occurs earlier in a project when a higher growth rate is encountered, since this point depends on both the growth rate and the duration of the project. The point of doubling requirements can be calculated from the growth function with the size at the end being 2 times the size at the start. This results in solving the equation \((1 + r/100)^t = 2\). The solution is presented in Function 2.5. The logarithm used in these functions and in all formulas in the remainder of this article is the natural logarithm; the logarithm with base \(e\).

\[
 t = \frac{\log 2}{\log (1 + r/100)} \quad (2.5)
\]

A simplification of Function 2.5 is often referred to as the 72-rule [87, p. 181, Theorem 44], used in compound interest calculations in accounting to calculate the time when an investment doubles its value. This simple approximation is presented in Function 2.6.

\[
 t = \frac{72}{r} \approx \frac{\log 2}{\log (1 + r/100)} \quad (2.6)
\]

The number 72 is used, since the value of \(\log(2)\) is close to 0.72, 72 has many divisors, and when \(r\) is small, the value of \(\log(1 + r/100)\) approximates \(r/100\), since \(\log(1)\) equals 0 and the derivative of \(\log(x)\), being \(1/x\), equals 1 if \(x = 1\).

In Table 2.3 some values of doubling requirements are shown as an example. The numbers in Table 2.3 are from the exponential function \(b^t\) in which the multiplier \(b = 1 + \frac{r}{100}\) with \(r\) being the compound monthly volatility rate of a project and \(t\) the duration of the project.
measured in months. Jones already showed maximum growth rates for certain project types as we have seen in Table 2.1, but mentioning neither the project size nor duration. In this paragraph we will explain how to calculate for a certain project duration an upper bound for the requirements growth rate when healthy volatility becomes unhealthy.

With a 2% volatility we can already be in the danger zone, when requirements volatility reaches critical failure rates for longer projects, but for shorter projects this rate needs not be any problem. Figure 2.1 shows the growth function for different volatility rates. In this plot we show the rates 1%, 2%, 5%, 10%, 20%, and a theoretical $100 \cdot (e - 1)\% \approx 171.8\%$. The latter theoretical curve has exploding requirements from $t = 0$, because it has a more than proportional growth from day zero. More than proportional growth means adding more than initially was present every time frame. In financial terms this means that one receives the initial amount and more as interest every year, which is the interest period in our financial example. All projects in Figure 2.1 have a starting value of a hypothetical single function point project. For a project with a different starting value the growth curves in Figure 2.1 are exactly the same, the multiplier $b$ simply needs to be multiplied by the start value to obtain the future value. The horizontal dot-dash line represents the line for which the workload has doubled, and shows the corresponding doubling moments from Table 2.3 with vertical dot-dash lines.

An interesting characteristic about the growth function $b^t$ of the requirements or workload of a project in Figure 2.1 is the point for which the derivative equals 1, or equally, when the time-elasticity of volatility equals 1. Since, before this point every time increment adds a lesser increase in function points than the amount of function points present at $t = 0$ and after this point every time unit increase adds more function points than were initially present. We solve the equation of the derivative of this growth function $f(t) = b^t$ equalling 1 to find this point. Recall that $b = 1 + \frac{r}{100}$. The derivative is the following function for a certain multiplier $b$ and a duration $t$ between the initial and last size estimate:

$$f'(t) = \log (b) \cdot b^t$$  \hspace{1cm} (2.7)

We need to solve the equation $f'(t) = 1$ to find the point $\hat{t}$ for which the function $f(t) = b^t$ intersects a certain tangent. This intersecting tangent is of the form $g(t) = t + \hat{t} + b^{\hat{t}}$. In Figure 2.2 we have redrawn Figure 2.1. The figure is zoomed out a little, but now with the line connecting the points for which the derivative is 1. This point is the first period in which the size of requirements added in this period is equal to the original size at $t = 0$. Therefore, after this point the requirements increase explodes, creating more requirements.
CHAPTER 2. QUANTIFYING REQUIREMENTS VOLATILITY EFFECTS

Figure 2.1: Requirements growth.

every month than were counted at the beginning of the project. So, the line in Figure 2.2 shows the duration when proportional growth starts. Actual measurement of such rates for a project is an indicator for apparent utter failure. This point is in our financial intro
the point when your bank account capital increases fast, since every month the absolute amount of interest that you get from the bank is higher than the amount you initially put in the bank account.

The solution of Equation 2.7 is Formula 2.8 for a certain multiplier \( b \) and is found as follows:

\[
\log (b) \cdot b^t = 1 \quad \text{divide by } \log b
\]

\[
b^t = \frac{1}{\log b} \quad \text{logarithm of both sides if } b^t > 0
\]
IT Risks in Measure and Number

Figure 2.2: Connecting the tangents of proportional growth.

\[ \log (b^t) = \log \left( \frac{1}{\log (b)} \right) \]

\[ t \cdot \log (b) = -\log (\log (b)) \]

\[ t = -\frac{\log (\log (b))}{\log (b)} \quad b^t > 0 \quad (2.8) \]

### 2.4.1 Lambert’s W Function

The solution presented in Equation 2.8 can be used to calculate a maximum project duration in months for a given rate \( r \). Recall that \( b = 1 + \frac{r}{100} \). Figure 2.1 shows the tangent
point for the functions $1.10^t$, $1.20^t$ and $e^t$. For the other shown functions these tangents are outside the bounds of the figure. Function 2.8 intersects the horizontal axis at the point $b = e$, since for any value larger than $e$ the function returns a negative value. The function has a minimum at $e^e$ and is asymptotically going towards 0 for higher values. So for any rate higher than $e - 1 \approx 1.72$, the growth function has a tangent with a slope higher than 1 immediately starting at $t = 0$.

The inverse function of Function 2.8 is Function 2.9 when solving for $b$ and Function 2.10 when solving for $r$. Function 2.9 results in a multiplier factor $b$, with $b = 1 + \frac{r}{100}$ and $r$ being the compound monthly volatility rate. In Figure 2.4 we will draw Function 2.10. Function 2.10, resulting in a multiplier $b$ for a project with duration $t$, can be found solving equation 2.8 using Lambert’s $W$ function [25, 33, 75]. Later on we will show how we arrived at this solution.

\[ b = e^{W(t)} \quad b^t > 0, \quad b = 1 + \left(\frac{r}{100}\right) \quad (2.9) \]

or, equivalently for rate $r$

\[ r = \left(\frac{e^{W(t)}}{t} - 1\right) \cdot 100 \quad r, t > 0 \quad (2.10) \]

Lambert’s $W$, or Omega function, used in Equation 2.9, is the inverse of the function $f(x) = x \cdot e^x$ and is named after Johann Heinrich Lambert. The Omega function derives its name from the constant $\Omega$, that is defined by Equation 2.11, the value of $W(1)$.

\[ \Omega \cdot e^\Omega = 1 \implies \Omega \approx 0.567143290409 \quad (2.11) \]

The Lambert $W$ function is multi-valued in the interval $[-\frac{1}{e}, 0]$ as can be seen in Figure 2.3. Since we will use only positive values as input for the $W$ function, the multi-valued part will not be of any hindrance in our calculations. Most mathematical packages
support the usage of the Lambert \( W \) function with standard functionality or with an additional package. Or, it is also possible to calculate the inverse of the function \( x \cdot e^x \) for necessary \( x \)-values and put these values in a table and then switch the \( x \) and \( y \) values. In Section 2.10 we will show a table of Lambert \( W \) values to aid in quick calculations. To give the reader a better understanding of the \( W \) function, the function is drawn in Figure 2.3 for the interval \([ -\frac{1}{e}, 10] \).

**Solving with Lambert** This paragraph shows how the \( W \) function has been used in solving Equation 2.7. Since every function of the form \( Y = X \cdot e^X \) can be rewritten to \( X = W(Y) \) [25], we will rewrite Equation 2.7 in that format in order to solve the equation for \( b \):

\[
\log (b) \cdot b^t = 1 \quad \Rightarrow \quad e^{\log a \cdot a} = a
\]

\[
\log (b) \cdot e^{\log (b')} = 1 \quad \Rightarrow \quad a \cdot \log \frac{b}{a} \cdot \log \frac{b'}{a}
\]

\[
\log (b) \cdot e^{t \log (b)} = 1 \quad \text{multiply by} \ t
\]

\[
t \cdot \log b \cdot e^{t \log b} = t \quad X = t \cdot \log b \text{ and } Y = t \text{ and } Y = X \cdot e^X \Rightarrow X = W(Y)
\]

\[
t \cdot \log b = W(t) \quad \text{divide by} \ t
\]

\[
\log b = \frac{W(t)}{t} \quad e^{\text{left- and right-hand side}}
\]

\[
b = e^\frac{W(t)}{t} \quad \text{(2.12)}
\]

The limitation we have created by solving for variable \( b \) is \( b^t > 0 \). But, we have only positive durations, since \( t \) is time expressed in months and \( b \) expresses the multiplication factor \( 1 + \left( \frac{r}{100} \right) \). A negative \( b \) has no physical meaning in our model, since that would imply a project with a negative number of function points. Therefore, \( b^t \) will always be larger than zero in our model. With Equation 2.12 we now have a function to calculate the maximum allowable volatility rate \( r \), with \( b = 1 + \left( \frac{r}{100} \right) \) when we have a project duration \( t \) available.

**Example of volatility approaching the danger zone** The resulting volatility function in Equation 2.10 calculates the monthly volatility rate \( r \) for a certain duration \( t \) with which the growth function \( (1 + r)^t \) will have a tangent of 1 at moment \( t \). Function 2.10 can be illustrated with the following example. Let’s consider two projects. The first is a two month project initially estimated at 100 function points and the second a 20 month project initially estimated at 1000 function points. Equation 2.10 dictates for the first project a maximum monthly requirements growth of 53% to barely avoid proportional growth and for the
latter a maximum of 16.5% per month. These percentages are calculated with exact values of the Lambert \( W \)-function. In Section 2.10 we will present a table of the Lambert \( W \)-function for different values of \( t \) for quick reference and to ease calculations. So, for a project with a short duration, larger volatility rates are acceptable than for longer projects. This is also visible if we look at the absolute sizes of both projects when the volatility rates have been equal. At the end of the projects we calculate the requirements volatility for both projects. In this example both projects underwent a 15% monthly requirements growth. For the first project this is very close to the dictated maximum, then again, it resulted in a final size of 132 function points. The second project, on the other hand, ends up with more than 16,000 function points, or in other words has grown by a factor 16, definitively indicating that it is out of control. Clearly, healthy growth depends on project duration. What is an acceptable rate for the first short project is totally unacceptable for the other, longer project. Therefore, Jones’s industrial averages are inadequate as an early warning system to detect unhealthy volatility.

2.4.2 Tolerance factor \( p \)

The point in the growth function for which the derivative function equals 1 was already interesting to look at, since it indicates the start of more than proportional growth. Indeed, no one argues that growth rates leading to 100% requirements increase are a strong indicator of projects going astray. But what growth is still healthy? We can only know by analyzing this in a low-risk portfolio. A higher coefficient of the tangent of the growth function is possible, but a lower value of this tangent is more desirable when looking at requirements that are out of control. Therefore, we will look at \( p \)-proportional growth in this section. We calculated for the aforementioned Function 2.9 also the point for which the derivative equals 0.5 or 2. To create a function similar to Equation 2.9 we will need to solve a new equation, see Equation 2.13, in which \( p \), replacing 1 in Equation 2.9 indicates the slope of the tangent of the growth function, \( t \) the duration between size estimates and \( b \) the multiplicative factor defined by \( b = 1 + (r/100) \).

\[
p = p(b, t) = \log b \cdot b^t
\]

We call \( p \) the tolerance factor of the requirements volatility of a project. If \( p \) is low the tolerance for high requirements growth rates is low and a high \( p \) implies that high requirements growth is acceptable. If \( p \) equals 1 the plot is equal to the previously shown Figure 2.4. The actual value of the tolerance factor \( p \) for a specific completed project can be calculated with Equation 2.13. If you calculate this tolerance factor for all projects \( p_i \) in a portfolio you can take the portfolio’s maximum tolerance factor \( p \). When you have found this specific tolerance factor \( p \), you have found the maximum tolerance factor \( P \) of a portfolio of projects, see Equation 2.14.

\[
P = \max(p_i)
\]

The factor \( P \) depends on a portfolio of projects on the processes and tools that exist to create software, thus representing a lower or higher tolerance for high volatility. And if you know you have a low-risk portfolio, then calculating the portfolio’s maximum tolerance
factor $P$ indicates what $p$-proportional growth is acceptable. Such volatility boundaries used to be unavailable, but now we can calculate a measure for healthy and unhealthy growth.

Solving Function 2.13 for $b$, $t$ or $r$ results in the Equations 2.15, 2.16 and 2.17 with $b$, $t$ and $r$ defined as before. In fact, these equations are similar to the previous Equations 2.9 and 2.10 with tolerance factor $p$ added to them. When $p$ equals 1 we end up with the previously described formulas.

$$t_\pi = \frac{\log p - \log (\log b)}{\log b} \quad (2.15)$$

$$b_\pi = b_\pi(p, t) = e^{W(p \cdot t)} \quad b^t > 0, b = 1 + (r/100) \quad (2.16)$$
We decorate the solved variables that indicate $p$-proportional growth with the subscript $\pi$ for usage in the following section. In Figure 2.5 a plot is shown of Equation 2.17 for $p = 0.25, 0.5, 1$ and 2. The intercounting duration in Figure 2.5 is the duration between two size estimates. Figure 2.5 shows that for lower values of $p$ we get a lower tolerance for high volatilities. For higher values of $p$ the plot shows a higher tolerance for high volatilities.
2.4.3 Requirements volatility metric: the $\pi$-ratio

In order to judge projects on their volatility we need to be able to compare projects of different duration and size. We have just seen that comparing solely their rates is incorrect: what is an acceptable rate for one project is problematic for another project. So we need a measure to judge the requirements volatility of projects even if they have different durations and different sizes. In this paragraph we will start with a metric to compare projects of different duration, but not yet take size into account. We do this by dividing a project’s actual requirements volatility rate $r_{act}$ by its maximal healthy volatility rate calculated by Equation 2.10, or, when a tolerance factor different from 1 is being used Equation 2.17. The calculation of this volatility ratio is illustrated in Figure 2.6. We illustrated with the slender arrow the $p$-proportional maximum volatility of 22.22% for $p = 1$. The thick arrow shows the actual measured volatility for this project, which is 10%. The volatility ratio $\pi$ is then calculated by dividing these numbers, resulting in a so-called $\pi$-ratio of 0.45.

The $p$-proportional volatility ratio $\pi$, presented in Equation 2.18, being a number larger than 0, now provides us with an indication of how close a project has approached the maximal healthy $p$-proportional volatility curve, or, equivalently, how close the project has approached the danger zone. The $\pi$-ratio is calculated for a project with a duration $t$, a tolerance factor $p$ and an actual requirements volatility $r_{act}$. If you calculate the $\pi$-ratios for at least two projects it is possible to compare their volatility. If you encounter a $\pi$-ratio larger than 1 with $p$ equaling 1, then the project has experienced more than proportional requirements growth. Later we will calculate the $\pi$-ratio for various projects stemming from different portfolios and encounter projects that have experienced more than proportional requirements growth.

$$\pi_p = \pi_p(r_{act}, t, p) = \frac{r_{act}}{\pi} = \frac{r_{act}}{\left(e^{-\frac{W(p, t)}{t}} - 1\right) \cdot 100} \quad (2.18)$$

2.4.4 Requirements volatility metric: $\rho$-ratio

Although we have already created a model to compare projects of different duration, we are now going to extend the model. Instead of taking only our just proposed tolerance factor $p$ and duration into account, we will now introduce the size of a project as an additional variable in the volatility ratio. The rationale of this metric extension is the following. If we have a project of 100 function points realized within 12 months and it doubles in requirements to 200 function points throughout the project, then this is not completely comparable to a project of 1000 function points doubling to 2000 function points, also realized within 12 months. Although the time span was equal for both projects, the latter project is much more out of control because of the extremely large resulting size. In [61, p. 202] Jones presented a benchmark for the relationship between duration $d$ and size $f$ of IT projects, we test this existing benchmark on plausibility for the bancassurance portfolio. In Equation 2.19 the relationship is presented in which the exponent of the formula is based on a non-linear least-square estimation of the total project duration and the final function point size for our bancassurance portfolio. The value of the exponent for the
bancassurance portfolio is 0.359, which is similar, but a little bit smaller than the values presented by Jones in [61].

\[ f^{0.359} = d \] (2.19)

The \( p \)-statistic of the least-square estimation is very small. The \( p \)-statistic equals 5.848095 \times 10^{-58}, implying a rejection of the null-hypothesis. Therefore, the relationship presented in Figure 2.7 is not a result of chance alone. In Figure 2.7 we have plotted the values for \( f, d \) and the by non-linear regression obtained function \( f^{0.359} = d \) for our bancassurance portfolio. As a side note we want to remark that the removal of the data point in the upper right corner only slightly changes the regression line: the exponent in function in that case becomes 0.358. As can be seen from Figure 2.7 not all variation in the data is explained by the regression function. Therefore, we cannot omit size from our volatility metric and will add it to the equation. Omitting size and thus falling back to the \( \pi \)-ratio is in fact only
possible if all the influence of the project size is explained by the project duration.

Figure 2.7: Project size versus project duration combined with estimated function.

In Equation 2.20 we express the influence of size on the volatility in mathematical terms, by incorporating the logarithm of the function point size $f$ into the volatility calculation of a project. The other variables $p$ and $t$ are defined as before. By introducing size, the volatility metric will give a higher weight to larger projects, becoming more sensitive for higher requirements volatility occurring at larger projects. We opted for the logarithm so the volatility metric will not become immediately allergic for project size, but gradually more sensitive. The logarithm is a monotonic and slowly rising function, so its value will be higher for larger project sizes. The root function, for instance, is also slowly rising, but the logarithm expresses a project’s order of magnitude which we want to incorporate. The logarithm function results in a discriminating effect between larger and smaller projects. Not having a function that gradually rises, results in a metric that drops to zero unrealistically quickly, thus making it allergic and useless for project sizes already above $10^3$. 

31
function points. Of course we could have taken another choice instead of the logarithm, but that does not change the relative comparisons between projects when taking their size and duration into account.

\[ r_\rho = \left( e^{\frac{w(p,t)}{t^2}} - 1 \right) \cdot 100 \quad (1 + r/100)^t > 0, \ f > 1 \quad (2.20) \]

We illustrate the maximum requirements volatility presented in Equation 2.20 with the three-dimensional Figure 2.8. Figure 2.8 shows the safe zone for different sized projects with different durations. The surface of the plot represents the point for which unhealthy requirements growth starts for projects of size \( f \) and intercounting duration \( t \).

![Figure 2.8: \( r_\rho \) for \( p = 1 \) and intercounting duration \( t \) and size \( f \).](image)

With this extended model we can introduce a new volatility metric, the requirements volatility ratio \( \rho \) for a project based on the actual volatility rate \( r_{act} \), the size of the
IT Risks in Measure and Number

project \( f \), the intercounting duration \( t \) and a tolerance factor \( p \). This new ratio is defined in Equation 2.21 and is used to calculate a duration and size invariant volatility metric.

\[
\rho_p = \rho_p(r_{\text{act}}, t, p, f) = \frac{r_{\text{act}}}{r_{\rho}} = \frac{r_{\text{act}}}{e^{\left(\frac{w(p \cdot t \cdot \log f)}{p^2} - 1\right)} - 1} \cdot 100
\] (2.21)

**Minimum requirements volatility**  
For completeness’ sake we state that the lower bound in Figure 2.4 for requirements volatility is not 0%, i.e. no volatility, but the lower bound is \(-100\%\) for the duration \( t = 1 \). Complete requirements scrap has then taken place at \( t = 1 \), no requirements are left. A project with a longer duration than 1 month cannot have a compound monthly requirements volatility of \(-100\%\). This is, because as with Zeno’s paradox, with a monthly requirements volatility higher than \(-100\%\), we are losing most of the requirements every month with a negative volatility percentage, but never all requirements. The moment that all requirements have been scrapped, the paradoxical point in the race between Achilles and the Tortoise, is not expressible in a compound monthly rate for a project with a duration longer than 1 month.

**Summary**  
From this section we conclude that a maximum requirements volatility rate for a project is not a uniform fit for all project sizes and durations. So, the currently published failure factors for requirements growth by Jones, see Table 2.1, do not predict failure in practice. Therefore, we do not recommend their use. From the figures and equations it is evident that projects with a shorter duration accept a higher requirements volatility rate than projects with a longer duration. We have created the \( p \)-proportional \( \pi \)-ratio to compare the volatility of different projects. Furthermore, we proposed the volatility metric \( \rho \) taking the size of a project into account as well, since size and duration do not completely correlate, therefore, the more complex \( \rho \)-ratio will be preferable over the \( \pi \)-ratio. The \( \pi \) and \( \rho \)-ratio enable true volatility comparisons between projects and benchmarks.

### 2.5 Case study: measuring requirements volatility

As was observed in [118] many important software KPIs display stochastic behavior. In fact KPIs like cost, duration, size, etc. often have an asymmetric leptokurtic possibly heterogeneous probability density function (PDF). Leptokurtic means that the probability density function has a positive kurtosis, i.e. the form of the statistical frequency curve near the mean of the distribution is more peaked. *Kurtosis* is Greek for *bulging or curvature* and *lepto* is Greek for *slender or peaky*. So, it means a more slender frequency curve near the mean compared to the normal distribution. To discover stochastic nature it is necessary to look at the characteristics of any indicator over a complete portfolio. Our empirical research revealed that in accordance with other important software KPIs our proposed volatility rates \( r, p, \pi \) and \( \rho \) resemble the family of Generalized Pareto Distributions (GPD). For the volatility \( r \) we will show this in detail, and the others are treated analogously but not shown here. Of course, we cannot conclude this from a few projects,
but we can from an entire portfolio of 84 projects. The conclusions in this chapter are a result from considering the characteristics of the volatility of several portfolios of projects instead of the averages presented in Table 2.1. All project data from different organizations that are the basis for this chapter are real-world cases. The 23.5 million dollar costing IT portfolio consisting of 84 projects stems from a larger portfolio of IT projects in the bancassurance sector. This larger portfolio has been subject to more research in which it is identified as being low-risk [118]. The projects in the bancassurance portfolio represent together some 16,500 function points, 64 of the projects were developed in-house and 20 projects were partly or completely outsourced. The 84 projects are subprojects from larger projects. Projects in this bancassurance portfolio are split up to reduce risks and improve control into smaller subprojects for which size estimates are made and are used in our analysis. The subprojects with smaller sizes also allow for higher volatility rates which we will see later, are higher than Jones’s failure factors, see Table 2.1. The analyzed portfolio concerns not only the development of new applications, but also changes and significant enhancements to existing applications. The function point totals of the considered projects in this portfolio have the characteristics represented in Table 2.4. We can see that most projects are of a size between 67 and 227 function points, since the first and third quartile represent respectively the 25% and 75% interval of the data set that contains the function point totals. Figure 2.9 shows the empirical probability density function of the project sizes in function points. The integral of the empirical density function over any data set gives the probability that random data points in the data set will fall within that certain interval. As can be seen from this figure, the curve is very slender, indicating a portfolio with a lot of small sized projects and very few projects that have a size over 500 function points, with a maximum size of 2282 function points. The figure is in accordance with earlier findings with projects of a similar portfolio presented in [118].

<table>
<thead>
<tr>
<th>min.</th>
<th>first quartile</th>
<th>median</th>
<th>mean</th>
<th>third quartile</th>
<th>max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.0</td>
<td>66.8</td>
<td>123.5</td>
<td>197.1</td>
<td>227.5</td>
<td>2282.0</td>
</tr>
</tbody>
</table>

Table 2.4: Function point countings characteristics.

For each project in this portfolio the function points were counted twice or thrice by certified function point counters that counted consistently as we stated earlier, for details see [118]. The first counting was done based on the requirements that were produced in the initial phase of the project right before functional design started. After delivery of the project the definitive requirements were measured in the second function point analysis. The sum of function points of all first analyses is about 15,687 function points, this portfolio grows to the total sum of 16,605 function points at the last counting, an overall growth of about 6%. For some projects after the functional design an extra size estimate of the requirements was done. In Section 2.5.4 we will further investigate these projects. For all the projects the different function point countings were considered to gather a rigorous basis for an analysis of the compound volatility rate. Because the requirements documents sometimes contained some temporary lacunae, the function point counters added a certain percentage, based on experience, to a function point counting. These percentages were needed to represent the requirements that were, although present requirements, not completely elaborated and were incorporated in the analysis to obtain a more realistic
volatility. The main characteristics of the thus calculated compound monthly growth rates are presented in Table 2.5.

<table>
<thead>
<tr>
<th>difference</th>
<th>min.</th>
<th>first quartile</th>
<th>median</th>
<th>mean</th>
<th>third quartile</th>
<th>max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>relative $r$</td>
<td>-23.7</td>
<td>-3.6</td>
<td>0.0</td>
<td>0.9</td>
<td>3.7</td>
<td>31.5</td>
</tr>
<tr>
<td>absolute $r$</td>
<td>0.0</td>
<td>1.4</td>
<td>3.7</td>
<td>5.8</td>
<td>6.3</td>
<td>31.5</td>
</tr>
</tbody>
</table>

Table 2.5: Compound monthly volatility rate $r$ characteristics for the bancassurance portfolio.

From Table 2.5 it is apparent that the requirements volatility rate has a median of 0% and an average of 0.903%. This average is close to Jones’s averages in Table 2.1. The last row of Table 2.5 displays the absolute values of the requirements volatility rate, all negative volatility rates were made positive, as a measure to express change. Combining the
64 in-house projects with Jones’s average of 1.2% requirements volatility and the 20 outsourced projects with Jones’s average of 1.1% requirements volatility, results in a benchmarked volatility of this IT portfolio that amounts to \((64 \cdot 1.2 + 20 \cdot 1.1)/84 = 1.176\) according to Jones’s industrial averages. This volatility based on Jones’s averages is only 0.27% percent point off of our own measured volatility average of 0.903%. This small deviation is induced by using Jones’s benchmark for management information systems on a portfolio of bancassurance specific management information systems. We infer from the coincidence of Jones’s latest averages with the averages of our detailed information that in itself it is an indication that it is useful to use Jones’s work in the absence of the detailed data that we have. But, since the density is akin to generalized Pareto distributions, it displays heavy tails. In that case, median and mean can differ strongly, so that an average is not that useful and it is therefore strongly recommended to construct your own benchmark based on your own projects with the metrics presented in this chapter or use the benchmarks presented here as a surrogate, rather than an average of a presumably strongly asymmetric data set.

Jones has measured maximums that are displayed in Table 2.1, but for 21% of the projects we have measured higher volatilities than Jones, even though our bancassurance portfolio is of low-risk. Fifty percent of the volatility rates in our bancassurance portfolio are in the interval between -3.553% and 3.708% compound monthly volatility rate.

As we have seen, omitting project duration in volatility assessments, results in too optimistic figures for long projects and too pessimistic figures for short projects, making them not very useful for assessments of individual projects, whereas when using our volatility models this is conveniently possible. Our proposed approach includes the duration of a project and thus provides a more granular view of maximum volatility rates, so it creates a better distinction between healthy and unhealthy projects. And thus this method is a useful tool for IT governors, whether making volatility specifications in an outsourcing contract situation or whether creating an IT dashboard to monitor volatility and signal abnormalities.

Table 2.5 shows maximum and minimum volatility rates that are much larger than zero percent volatility, the mythical no-change-no-delay policy. We will see later that these were projects, with a relatively short duration; we recall that high volatility rates are acceptable for short projects. For example, in our bancassurance portfolio, 21% of the projects have an absolute monthly requirements volatility higher than 5%, the failure rate stated by Jones. With a significant project size, such projects qualify directly for executive attention when applying Jones failure factors. But for projects with shorter duration or smaller sizes this is not the boundary that should be used. Depending on the amount of projects we can define boundaries among the projects using our volatility ratio \(\pi\), the metric that takes into account also a project’s duration besides the experienced volatility \(r\). With a boundary of \(-0.5 < \pi < 0.5\) we end up with 8% that needs attention instead of 21% of the projects. This will be shown in Figure 2.14 that we will discuss later on. These projects are candidates for immediate management attention since our metric is highly correlated with projects out of control. Our tool is therefore a sieve that gives IT executives a hint as to where to put their valuable time and effort. Management attention that is directed to the capped portfolio of projects that require management attention results in putting the identified derailing projects back on track before it is too late.
The merit of our method is that we identify these projects early within large IT project portfolios.

### 2.5.1 Volatility Density Function

Next, we will analyze the volatility indicator’s probability density function and its outliers for the bancassurance portfolio. Figure 2.10 shows initial visual explorations. In the upper left corner a Box-and-Whisker plot is drawn for all the volatility values. Boxplots were originally introduced by John Tukey in 1977 [115] and are used to visually depict the five-number summaries as illustrated in Table 2.5.

Boxplots can help in identifying the skewness or the variance of an underlying probability function. The box in a boxplot represents the first and third quartile of the data set, the so-called interquartile range, and the horizontal lines outside the box, the whiskers, are
defined by the last observed data point that lies within 1.5 times the interquartile range. The outliers in a data set are represented in a boxplot with dots.

In the upper-right corner a histogram of the bancassurance volatility rates is drawn together with the probability density function. In this plot can be seen, as the boxplot and [118] already hinted, that the probability is leptokurtic and heterogeneous with heavy tails on both sides. To further illustrate this assumption we provide also Quantile–Quantile plots, or short Q–Q plots in Figure 2.10. Q–Q plots are a graphical tool to diagnose the distribution of samples. When the plot does not show a straight line, the underlying data is not drawn from the same distribution as the distribution on the horizontal axis. The quantiles of the left-hand side of our density plot, i.e. everything left of 0%, and the quantiles of the right-hand side, everything right of 0%, have been plotted against the quantiles of the exponential distribution in the lower two plots in Figure 2.10. We use the exponential distribution as a reference here to confirm the heavy tail of the requirements volatility with visual means. These quantiles appear to approach a straight line, thus indicating heavy tails on both sides. These heavy tails can be interpreted by having occurrences with extreme values, or in other words—these values are most likely not stray values, but actual occurrences.

Care should be exercised if a probability function has several peaks, since this could indicate that this is a natural characteristic of the data. To make a comparison with charity donations, it is customary to find two peaks in the probability density functions of the donations. The lower peak is induced by contributions from individuals and a different peak at a higher value, for donations made by companies.

### 2.5.2 Kurtosis and Skewness

Despite the actual occurrences that are causing small peaks on the right-hand side of the probability function it is still possible to calculate the actual kurtosis and the skewness of the data. The kurtosis of sample data is defined as the fourth sample moment about the mean divided by the square of the second moment about the mean, the sample variance. See Formula 2.22 for the kurtosis formula for a data set with \( n \) values \( x_i \). In some definitions of the kurtosis formula there is a subtraction of three from the result. With this subtraction the formula results in a kurtosis of zero for the normal distribution, which is also called mesokurtic. Functions with a kurtosis higher than zero are called leptokurtic. The skewness of a probability distribution is the degree to which a distribution departs from symmetry about the mean. Sample skewness is defined by Formula 2.23 for a data set containing \( n \) values \( x_i \); i.e. the third moment divided by the third power of the square root of the second moment. A skewness of zero indicates a symmetric probability function. The results for the requirements volatility using Equation 2.22 and 2.23 are presented in Table 2.6.

\[
\text{Kurtosis} = \frac{m_4}{m_2^2} - 3 = \frac{n \sum_{i=1}^{n} (x_i - \bar{x})^4}{(\sum_{i=1}^{n} (x_i - \bar{x})^2)^2} - 3 \quad (2.22)
\]

\[
\text{Skewness} = \frac{m_3}{m_2^{3/2}} = \frac{\sqrt{n} \sum_{i=1}^{n} (x_i - \bar{x})^3}{(\sum_{i=1}^{n} (x_i - \bar{x})^2)^{3/2}} \quad (2.23)
\]
Table 2.6: Kurtosis and skewness of requirements volatility for the bancassurance portfolio.

Table 2.6 confirms with a kurtosis of 3.27 the slenderness of the requirements volatility rate. And the skewness of 1.09 indicates that the requirements volatility is right-skewed, the right tail of the distribution is heavier than the left tail.

### 2.5.3 Actual tolerance factors

![Various tolerance factors for the bancassurance portfolio.](image)

Figure 2.11: Various tolerance factors for the bancassurance portfolio.

For each project’s tolerance factor \( p \) a function as drawn in Figure 2.4 can be made. In
CHAPTER 2. QUANTIFYING REQUIREMENTS VOLATILITY EFFECTS

Figure 2.11 we have drawn a picture with the same axes as Figure 2.4 for the bancassurance portfolio. In Figure 2.11 the tolerance factors $p$ of the highest tolerance factor $0.68$ is shown with the accompanying function 2.17, this line displays the placement of equal tolerance factors, $0.68$, for larger and smaller projects. The same function is also displayed for the tree following high tolerance factors. The figure displays the spread of actual tolerance factors for projects in this portfolio.

<table>
<thead>
<tr>
<th></th>
<th>min.</th>
<th>first quartile</th>
<th>median</th>
<th>mean</th>
<th>third quartile</th>
<th>max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>-0.10</td>
<td>-0.028</td>
<td>0</td>
<td>0.040</td>
<td>0.046</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Table 2.7: Tolerance factor $p$ characteristics for the bancassurance portfolio.

In Table 2.7 the statistical summary of the tolerance factors $p$ for the bancassurance portfolio is shown. We remind the reader that a tolerance factor of 1 was equivalent with starting to experience a more than proportional requirements growth. The tolerance metric $p$ can be used to calculate the tolerance of a portfolio. This is done by calculating the tolerance factor $p$ with Equation 2.13 for all projects and then taking the maximum, denoted $P$. Since our bancassurance portfolio is of low-risk, we conclude that the empirical maximum tolerance factor of $P = 0.68$ that we encountered is an acceptable value for healthy requirements volatility in the bancassurance sector and can serve as a benchmark for other bancassurance portfolios. At the same time the tolerance factor can be used to pinpoint the projects with a high tolerance for volatility. Projects with a high tolerance for volatility can then be further investigated for the causes of this high tolerance.

In Figure 2.12 some general characteristics of our bancassurance portfolio are shown. The upper-left plot shows a scatterplot of the function point totals versus the spent hours for each project. The drawn regression line, $h = f^{1.446}$ has a very small p-value, indicating a good fit for this data set. The upper-right plot shows the function point size against the absolute value of the compound monthly requirements volatility. This figure shows that high volatility rates occur only at small-sized projects, therefore the regression line $r = 1/0.003f$ with a p-value of $4.4 \cdot 10^{-48}$ is also drawn. The lower two plots confirm this for the duration and the hours spent. The lower-left plot shows project duration against the absolute value of the volatility rate and the lower-right plot shows the amount of hours spent versus the absolute value of $r$. These bottom two pictures show that volatility rates are lower for projects that have longer durations, supporting that duration needs to be incorporated in a volatility metric. We know that we are dealing with a low-risk portfolio and we see high volatilities occurring at small projects. Indeed, further inquiries within the organization revealed that high volatility was managed by using DSDM and creating small projects thus isolating risks. So, their approach is a possible way to mitigate unhealthy requirements creep. The plots shown can aid in identifying projects that need management attention. Especially if we are dealing with an unknown portfolio it is possible to make a quick assessment to identify these projects by comparing them to our bancassurance benchmark. We will do so in Section 2.6 in which we analyze the volatility of a high-risk portfolio.
2.5.4 Differences between counting twice or thrice

Some projects in the bancassurance portfolio had an additional function point counting besides the counting at the end of the requirements phase and the counting after completion of the project. In all cases that the project manager requested an additional function point analysis, it took place at the end of the design phase. In our analysis the first and last counting were used to calculate the compound monthly requirements volatility rate.

Figure 2.13 juxtaposes different boxplots of the project portfolio to show the differences between twice and thrice counted projects. The first two boxplots in Figure 2.13 show the requirements volatility rate for these projects, revealing that the volatility rate is slightly higher for thrice counted projects, but at the same time the extremely high and low volatility rates are occurring at the twice counted projects. Previously we have seen that projects with high or low volatility rates were the projects with a short duration which
Figure 2.13: Differences between twice or thrice counted bancassurance projects.

is confirmed by the boxplots in the second plot in Figure 2.13. The left-hand side plot also shows with the two boxplots that projects having a relatively high volatility are usually being counted a third time. Because project management was already aware of potentially volatile requirements, they requested an additional function point analysis to have their feelings of requirements volatility materialized in a function point analysis; they did not want to wait for the final counting at the end of the project, to keep plan accuracy at the highest possible levels. This results in a higher volatility rate for thrice counted projects, since the volatility rate is based on the first and last counting. The interquartile range of the function point size for twice counted projects is from 65 to 227 function points, the interquartile range of the function point size for thrice counted projects is between 92 and 257 function points.

2.5.5 Inspecting the volatility ratios

Now we turn back to our volatility metric $\pi$ from Equation 2.18. We have calculated the $\pi_1$-ratio from Equation 2.18 for all projects in the portfolio and placed these ratios among a horizontal axis representing the intercounting duration of a project. This is demonstrated in the first plot in Figure 2.14. The same exercise can be done for our $\rho$-ratio metric presented in Equation 2.21 which also takes the project size into account. This results in the second picture in Figure 2.14 showing $\rho_1$. The resulting plots are similar but not completely equal, although the different projects are scattered more or less equally in the pictures. These plots display the spread of the $\pi$ and $\rho$-ratio for a low-risk portfolio and are useful for benchmarking other projects. Volatility risks in this portfolio were managed by having small projects, and each project adding value to the portfolio. Therefore, the size of a project is less influential, since the size is almost always less than 200 function points. In a high-risk portfolio there are larger variations between $\pi$ and $\rho$ due to larger sizes and thus creating more different scattering between the $\pi$ and $\rho$ plots. In Table 2.8
IT Risks in Measure and Number

We have listed a statistical summary of the volatility ratios $\pi_1$ and $\rho_1$.

<table>
<thead>
<tr>
<th>ratio</th>
<th>min.</th>
<th>first quartile</th>
<th>median</th>
<th>mean</th>
<th>third quartile</th>
<th>max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_1$</td>
<td>-0.64</td>
<td>-0.11</td>
<td>0</td>
<td>0.025</td>
<td>0.14</td>
<td>0.80</td>
</tr>
<tr>
<td>$\rho_1$</td>
<td>-1.43</td>
<td>-0.29</td>
<td>0</td>
<td>0.083</td>
<td>0.33</td>
<td>2.34</td>
</tr>
</tbody>
</table>

Table 2.8: Volatility ratios $\pi_1$ and $\rho_1$ for the bancassurance portfolio.

In Figure 2.14 the value of the tolerance factor $p$ for both ratios is 1. The scattering of $\pi$ and $\rho$ in figures like Figure 2.14 will remain the same for different values of $p$, but the numbers on the vertical axes will change if $p$ is different. Projects that are located on the upper edges of these drawings are projects with a high volatility, and should be further inspected to find the cause of the high volatility. Especially projects that have a more than proportional growth are in dire need of management attention. In Figure 2.14 we have drawn a solid line for proportional growth in the $\pi$-ratio plot and dashed lines that contain 95% of the projects in the $\rho$-ratio plot. By creating these boundaries the volatility outliers are easily identified. Subsequently of making these plots, a root-cause analysis needs to be performed on the requirements volatility of all projects that are outside these boundaries. By constricting these boundaries, more highly volatile projects will be included for inspection. Creating figures of volatility ratios like Figure 2.14 is insightful to decide on organization specific volatility boundaries that express tolerable project volatility. These volatility metrics can thus be used to identify projects with unbalanced behavior regarding the requirements volatility and therefore belong in an IT dashboard that is addressing the requirements volatility. In Section 2.10 we will present such a requirements volatility dashboard. Moreover, these volatility boundaries can be used in outsourcing contracts, agreeing both parties on a maximum $\pi$-ratio, invariant for project duration or, a maximum $\rho$-ratio invariant for duration and size. Since we have created a strong indicator for dan-
CHAPTER 2. QUANTIFYING REQUIREMENTS VOLATILITY EFFECTS

gerously high volatility, this metric certainly belongs in IT dashboards for IT governors. In Section 2.10 we present a simplified method to calculate the volatility ratios $\pi$ and $\rho$ yourself as well as how to create a requirements volatility dashboard.

2.5.6 Cost per function point

Figure 2.15: Cost per function point versus requirements volatility for the bancassurance portfolio.

In Figure 2.15 a scatterplot is made of the compound monthly requirements volatility rate against a cost index per function point. The costs are indexed for confidentiality. This plot shows a large blob of projects around the zero percent requirements volatility rate; the volatility rate of the projects has a standard deviation of 10.1 and 51 projects (80%) have an absolute volatility rate smaller than 10.1. Besides the blob, it also shows that some projects having a high positive volatility rate, tend to have a low cost per function...
point, which is non-intuitive. On the left side of Figure 2.15 the projects are displayed for which requirements scrap occurred and these tend to have a higher cost per function point. This indicates that when requirements are scrapped from a project the cost per function point increases. Contrary to intuition, removing requirements turned out to be associated with higher costs whereas you would expect lower costs due to designs that do not need to be made, functionality that does not need to be made and tests that do not need to be made nor need to be run. In the above intuitions we assume that requirements are scrapped early. In practice this is almost never the case. After the requirements are partly implemented, designed, tested, and so on, the stakeholders learn that not all initial requirements were optimal. So scrap is accompanied with the waste of partly done work not being expressed in the final function point analysis. This waste is costly, and will lead to a non-optimal design even if some requirements are taken out.

2.6 Benchmarking government projects

In this section we will use the empirical probability density function constructed from the bancassurance portfolio as a benchmark for projects from a government portfolio for which in some cases intermittent function point analyses were conducted for audit purposes. We know that the projects that are analyzed in this section come from a high-risk government portfolio and that one of the government projects that we analyze failed. We therefore perform this meaningful ex-post analysis on the requirements volatility of the government portfolio, to illustrate that the failure of some of its projects that actually did occur was predicted ex-ante by using our techniques. We predicted the failure while it was still ongoing by computing the requirements volatility using our techniques.

The portfolio consists of six projects that are evenly sized. Among the projects is a failing project with a size of about 1000 function points and a subproject consisting of about 300 function points. Countings for most of the projects have taken place at three points in time. From all resulting volatility rates we created a probability density function. This function and our bancassurance benchmark are plotted in Figure 2.16. Comparison of the probability density function with the bancassurance portfolio shows three differences: a lower peak, a peak that is more to the right and a heavier tail for the governmental project data. All these signs are indicating that the governmental IT portfolio is presumably out of control, and that some projects are totally derailing. To confirm the differences in the probability density function between the governmental and the bancassurance portfolio, we calculated for the governmental portfolio and the bancassurance portfolio the Kolmogorov-Smirnov test statistic, to test whether both data sets stem from the same continuous distribution. A distance $D = 0.51$ with a $p$-statistic of 0.00024 resulted, implying that we can reject the hypothesis that these data sets come from the same distribution. To give the reader insight into the growth of ongoing projects like our known to have failed governmental project and its subprojects we plotted the size of the project and the subproject over time in Figure 2.17. In this is figure also a dashed line visible for a hypothetical project of the same size that experiences a volatility rate of 2% and a dotted line to indicate 5% growth. In Figure 2.18 we have plotted the size variations of the other five government projects that we analyzed. Since we did not have timestamps of the five
intermediate function point analyses, they were placed in the middle of the total project. In Figure 2.18 we see that one of the projects is increasing very fast in a short period of time, and some other projects are experiencing churn.

With our benchmarked volatility ratios at hand and data on duration and size we can also position the governmental projects against our bancassurance portfolio in the plots we presented in Figure 2.14. When we calculate the \( \pi \) and \( \rho \)-ratios for the complete government project, \( \pi_{0.68} \) equals 0.34 and \( \rho_{0.68} \) equals 1.11. These are on the high side, but not yet entering the danger zone. If we on the other hand look at the volatility ratios for the subproject we see differences with the failing governmental project making the comparison interesting. We obtain \( \pi_{0.68} = 1.37 \) and \( \rho_{0.68} = 4.03 \) for the entire subproject and \( \pi_{0.68} = 1.76 \) and \( \rho_{0.68} = 6.98 \) for the first period of the subproject, points that are well over the edge of Figure 2.14 indicating that this subproject was well into the danger zone already after its intermediate size estimate. These governmental volatilities
Figure 2.17: Requirements growth over time.

are plotted alongside our benchmark data in Figure 2.19 as solid dots. We remind the reader that in this plot we plotted $\pi_{0.68}$ and $\rho_{0.68}$, whereas in Figure 2.14 we plotted $\pi_1$ and $\rho_1$. This results in an equal scattering of the bancassurance projects, but with a different scale on the vertical axis, and the project with the highest tolerance factor is now positioned on the line $\pi = 1$. The open circles in Figure 2.19 represent the bancassurance benchmark data. In Figure 2.19 we can see the subproject positioned above the line of healthy requirements growth in the bancassurance sector. The left solid dot is the $\pi$-ratio for the first part of the subproject, the right solid dot above the horizontal line is the $\pi$-ratio for the complete subproject. The maximum tolerance factor $P$ for all governmental projects is 1.71, which is the tolerance factor belonging to the first part of the failed subproject.

As we can spot right away in Figure 2.19 the $\pi$-ratio belonging to the intermediate function point analysis of the subproject as well as the $\pi$-ratio belonging to the last size
measure of the subproject are lying above the line $\pi = 1$, indicating that the subproject is beyond control and needs to be stopped immediately. Indeed in this case the project was eventually killed, and a complete overhaul was necessary since the already partly operational system was beyond its best-before date before it was even finished. Scatter plots like Figure 2.19 can thus be used as a volatility litmus test for ongoing projects within an IT portfolio, to signal for projects that are in the danger zone, or worse: beyond control. In this governmental situation, consecutive function point analyses were misinterpreted both by the auditors and management. Instead of being alarmed by the grandiose volatility, they used the growth to, erroneously, extrapolate the function point countings to estimate the function point total at planned project delivery. This illustrates that even when the function point size measures of a project are available, uninitiated people can draw totally false conclusions, and will walk with open eyes off the requirements volatility cliff.

The approach taken in the bancassurance portfolio helps to avoid these problems. In
the bancassurance portfolio larger projects are split up in smaller subprojects to mitigate risks. Jones [59] states that requirements creep is sometimes outside the control of the entire software organization and that it can be anticipated, but seldom it can be reversed once it occurs. The measures provided in this chapter serve as an early warning sign for high volatility rates. And when high volatility rates are encountered for a project, management needs to consider to split up the project in smaller more manageable projects as is common practice in the low-risk bancassurance portfolio in order to mitigate risk.

2.7 Volatility variations for outsourcing

Many organizations outsource their IT function or parts of it. As Jones mentions in his book [64], outsourcing of software development seems to decrease the compound monthly requirements volatility rate. There are several reasons clarifying this. First, outsourced projects are probably managed better on the client side. As the company that is giving the requirements to sourcing partners it is impractical and costly to have too many meetings about unclear requirements, so the company outsourcing its work will try to make requirements documents as clear and complete as possible. Second, it is the company receiving the development assignment that will try to complete the project as soon as possible to maximize their profit and thus clarify unclear requirements in the beginning of the project. Third, the sourcing partner will not mind charging more hours than initially estimated for the hours needed for clarifying the requirements. And fourth, there is a contract in between, and to prevent litigation conflicts, there is a tendency to turn the project into a success no matter what, which probably results in unpaid and unrecorded overtime.

Sometimes organizations tend to apply a zero-change policy at the start of an outsourcing deal with the idea of staying in control, also known as no-change-no-delay. But requirements vary by their nature, so a zero-change policy does not always lead to an
ideal end product. See Peters and Verhoef [91] for a discussion on business cases for requirements creep with a positive return on investment.

The bancassurance organization in question was curious to know whether a zero-change-no-delay policy would work out positively for their routine IT work. Business management thought that this would once and for all solve their problems they historically had with their in-house development. The bancassurance portfolio contains enough data to explore such questions. We submitted the bancassurance portfolio to an analysis whether or not outsourcing had an effect on the compound monthly requirements volatility rate. It turned out that outsourced projects display similar requirements volatility characteristics as in-house developed projects. We will show in this section how we arrived at this conclusion.

<table>
<thead>
<tr>
<th>project type</th>
<th>min.</th>
<th>first quartile</th>
<th>median</th>
<th>mean</th>
<th>third quartile</th>
<th>max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-house</td>
<td>14</td>
<td>64.75</td>
<td>116.5</td>
<td>141.2</td>
<td>190.2</td>
<td>552</td>
</tr>
<tr>
<td>outsourced</td>
<td>55</td>
<td>111.2</td>
<td>223</td>
<td>376</td>
<td>374.2</td>
<td>2282</td>
</tr>
</tbody>
</table>

Table 2.9: Size variations for in-house and outsourced projects from the bancassurance portfolio.

In Table 2.9 we show the size characteristics for the in-house versus outsourced projects. We see in Table 2.9 that outsourced projects are usually larger than in-house projects. This is to be expected, since when outsourcing work, larger portions of labor are more prone to be candidates for sourcing and smaller projects are done in-house.

The first plot in Figure 2.20 shows three different probability functions for the bancassurance portfolio. The dashed distribution represents the volatility rate for projects that were outsourced. In the case of this portfolio the projects were nearsourced, i.e. outsourced to a local company, as opposed to offshore outsourcing when projects are outsourced to other countries. The dotted probability function represents the in-house
volatility rate; the solid function is the probability function of the complete portfolio. As appears from Figure 2.20, outsourcing seems to make a difference, but we will see shortly that this is insignificant for our data set. In the second plot in Figure 2.20 the box-and-whisker plots for the same three portfolio partitions are shown. Here we see that the outliers with a high absolute volatility rate, \(|r| > 20\%\), are in-house developed projects. Table 2.10 presents the kurtosis and skewness of the three data sets to further illustrate the differences between the three. We remind the reader that the kurtosis and skewness of a normal distribution are zero.

<table>
<thead>
<tr>
<th></th>
<th>in-house</th>
<th>outsourced</th>
<th>combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kurtosis</td>
<td>2.17</td>
<td>0.67</td>
<td>3.27</td>
</tr>
<tr>
<td>Skewness</td>
<td>0.96</td>
<td>-0.73</td>
<td>1.09</td>
</tr>
</tbody>
</table>

Table 2.10: Kurtosis and Skewness of volatility of in-house versus outsourced projects.

Table 2.10 states that the form of the probability function is more slender than a normal distribution, both for the in-house and outsourced projects and that the functions are right-skewed. As we concluded before the outliers in Figure 2.20 are the smaller sized projects. In this portfolio, the outsourced projects are larger in function point size than the in-house developed projects.

### 2.7.1 In-house or outsourced?

As Table 2.9 states, the mean for in-house projects is 141 function points and for outsourced projects 360 function points. The plots in Figure 2.20 suggest that the projects in the outsourced partition are showing a lower requirements volatility than the in-house bancassurance projects, which concurs with Jones results summarized in Table 2.1 and his findings on differences in average sizes of in-house and outsourced projects [62, Table 7.7 and 8.5]. To validate this statement we have done a statistical test on the data, a Kolmogorov-Smirnov test, to detect differences in the probability density function for the two data sets. Despite the slight visual differences, there is no statistical proof of our hypothesis that they are different. The resulting test statistic \(D\), the maximum vertical difference between cumulative distribution functions, equals 0.25 with a \(p\)-value of 0.2968. This result does not give us statistical proof to conclude that the two data sets, in-house development and outsourced, are drawn from a different continuous distribution. We can clarify this statistical insignificance since the outsourced labor for these projects was not just thrown over a fence, but done on-site with the customer. External IT staff adopted quickly to the in-house situation, more or less providing about the same requirements volatility as a regular on-site development. To further analyze our conclusion, we partitioned the complete set of project data randomly many times and revealed that when having more comparable data available, the difference in volatility between in-house and outsourced projects can become significant; corroborating Jones’s results that there is a difference. Resampling without replacement was performed by slicing the complete data set in two separate sets of the same size as the in-house and outsourced data, a subset of 20 projects and a subset of 64 projects. This slicing, sampling without replacement, was performed 10,000 times. For each partition the Kolmogorov-Smirnov test was done on
the two subsets, each time resulting in a value for the test statistic $D$. The probability
density function of all 10,000 $D$ values is plotted in Figure 2.21 with the test statistic for
the in-house versus outsourced partition plotted as a vertical line. The percentage of ran-
domly created partitions that had a KS test statistic larger than the actual value is only 7%.
So, in 93% of the random slices that were created a smaller maximal distance between
the cumulative distribution function was found than in the actual slice of in-house and
outsourced. The division of the data set into in-house and outsourced projects creates a
relatively large distance between the subsets when concerning the distribution function.
This indicates that partitioning the projects in an in-house and outsourced subset creates
a larger difference than most random partitions. So, it is unlikely that the volatility differ-
ences induced by this division are purely coincidental. Even though the current difference
is not statistically significant, it does not appear to be purely random either. When more
similar distributed data is available it is important to look at this partition, because the cur-
rent large distance, although not statistically significant, can barely be induced by sheer
coincidence.

2.8 The boomerang

Comparing the different amount of staff hours for projects usually shows that larger
projects have a lower productivity than smaller projects, because of increasing fixed costs
like communication. We will investigate this in the bancassurance portfolio. In Fig-
ure 2.22 the number of hours spent on a project are compared to the productivity for that
project. Since there was no complete data on hours spent for outsourced projects this
analysis was performed only on in-house projects.

This results in highly elastic productivity for projects of a size smaller than 100 func-
tion points, which means that a small proportional difference in size can have a large
proportional difference in productivity for the range from 1 to 100 function points. Func-
tion 2.24 states for the bancassurance portfolio the statistically fitted productivity function
$fppm$, a simple statistical fit which is a variation of the benchmark in [117, p. 61, For-
mula 46]. Function 2.24 benchmarks for a certain size $f$, expressed in function points,
its productivity expressed in function points per staff month. Equation 2.24 is plotted in
Figure 2.22.

$$fppm = \frac{677}{f} \quad (2.24)$$

This function, based on the project data, assumes a theoretical asymptote of zero function
points per staff month and a productivity smaller than 1 function point per staff month for
projects that are larger than 677.66 function points. This is not in accordance with real-
ity, however in this portfolio we can use this productivity relation for analysis purposes.
Next, we compare productivity with the absolute value of the compound monthly growth
rate. Surprisingly, some underlying effects tend to create clusters of projects along a
boomerang as can be seen in Figure 2.23. In the lower half of the figure a cluster of
projects can be seen that have a mutually increasing compound monthly growth rate, but
display the same productivity. In other words, these projects displayed mutually increas-
ing requirements creep, but were not able to manage the requirements change, resulting in
a low productivity. In the upper half of the figure there is a cluster of projects that have opposite characteristics. Whilst these projects do have a high compound monthly growth rate, the projects also have a high productivity. These projects seem to flourish better with higher requirements volatility. In the following paragraph we will discuss the causes of the boomerang in Figure 2.23.

### 2.8.1 Size and volatility

To assure that the differences in volatility in Figures 2.23 and 2.25 did not stem from size differences in the different projects Figure 2.24 has been made. In Figure 2.24 the size in function points is added to the plot, in which larger sizes imply a circle with a larger diameter. Figure 2.24 shows that both large and small projects appear in the different partitions. An analysis with the internal project portfolio of the different projects plotted,
CHAPTER 2. QUANTIFYING REQUIREMENTS VOLATILITY EFFECTS

Figure 2.22: Project hours versus productivity.

turned out to explain the boomerang shape.

In Figure 2.25 we present the boomerang picture again, but now for only three sub-portfolios each containing 7 to 13 projects. This figure shows a factor that supposedly has created the observed boomerang. The three partitions of the bancassurance portfolio have the following characteristics. We call the cluster of projects with high volatility, but low productivity, partition $x$. It turned out that these projects supported stock trading. Partition $x$ contains very complex projects tightly interwoven with a large legacy portfolio of existing IT assets mainly written in COBOL with several interfaces and the portfolio evolved over many years making enhancement difficult, thus resulting in a low productivity. This coincides with Jones’s type 5 enhancements [64]. Jones describes type 5 modifications as the classic form of maintenance of aging legacy applications with a low productivity of 0.5 to 3 function points per staff month. The high requirements volatility can be explained by the high cohesion between IT and business in this part of the ban-
The cluster of projects that display high requirements change, partition $y$, but also a high productivity, are in a partition of the portfolio that deals with back-office systems. In several project iterations new versions of the products were created. The projects succeeded in using mostly out-of-the-box tools to generate the necessary forms, thus resulting in high productivity. High volatility stemmed from different causes. First of all, the requirements usually changed during project iterations that were executed in close contact with the client. Secondly, the first function point counting took usually place in an earlier stage of the development project in this part of the portfolio than in other parts. Therefore, the requirements appear less stable when compared to projects from other parts of the portfolio. The organization confirmed that the development method was geared towards
CHAPTER 2. QUANTIFYING REQUIREMENTS VOLATILITY EFFECTS

Figure 2.24: Volatility variations versus the productivity of projects and size in function points.

Dealing with volatility. The last group, partition $z$, does not show a high requirements change, less than 6%, but shows a moderate productivity. This group is a partition of the portfolio concerning international payments. Requirements for international payments are based on international standards and must therefore have clearly defined requirements, which explains the low compound monthly volatility rate.

We partitioned Figure 2.23 in four quadrants, creating a diagram with four different types of projects. In Table 2.11 we listed the values corresponding to the quadrant division. The matrix in Table 2.11 corresponds with the quadrants in Figure 2.23. The high productivity–high volatility projects correspond to the upper-right quadrant, the low productivity–low volatility to the lower-left quadrant and their opposites the high productivity–low volatility projects correspond to the upper-left quadrant and the low productivity–high volatility to the lower-right quadrant. With these quadrants and the diagram it
is possible to score projects and provide a list of potential projects that need additional management attention.
CHAPTER 2. QUANTIFYING REQUIREMENTS VOLATILITY EFFECTS

2.9 Monitoring the volatility of an ongoing portfolio

When the requirements volatility of an ongoing portfolio needs to be continuously monitored, there is an alternative to intermittent function point analyses as size estimates. This section describes how to assess the volatility of ongoing projects at low cost, but with the drawback that, since we are monitoring production, outliers in the analysis are not necessarily due to volatile requirements. Our low cost method does allow quickly pinpointing of production variabilities for root-cause analysis of requirements volatility or other reasons.

Doing two function point measures for a project results in a cost of only two times the function point size expressed in estimation dollars. However, when it is necessary to have a constant grip on the volatility, monthly or weekly measurements become expensive. It is then cheaper to revert to daily reports of lines of code that are checked in. By using the total number of lines of code, LOC, from daily reports produced by configuration management tools, it is possible to estimate the size of a project on a daily basis through backfiring. Backfiring was introduced by Jones [65, 64, 62], and is simply a function point conversion rate for the lines of code and the programming language involved. In different programming languages it takes a different amount of lines of code to program one function point. In [62, page 78] Jones lists function point conversion rates for various programming languages.

2.9.1 Volatility of a software product line

In Figure 2.26 we show the size over a period of 206 days of a portfolio of correlated systems that together form a software product line for embedded software of a hardware manufacturer. Some of the subsystems were already deployed before the start date in Figure 2.26. Within the shown period a few values were missing in the data files, but not more than 9%, since it was not every day the report was created that listed the physical magnitude of the portfolio in lines of code. With interpolation of the surrounding data the few missing values were added. By applying backfiring we calculated the size in function points of the software product line. This number is shown in Figure 2.26 for all days. A line intersecting all points is also drawn in this figure in which we immediately notice the size shocks at certain points in time. The substantial discontinuities signal large volatility in very short time frames, and deserve direct attention to investigate whether something is going astray. Fortunately, most of these size jumps turned out to be explainable. The leap around day 58 occurred due to forking of a subsystem. From the forking at day 58 until day 93 of the observed period two versions of the same subsystem had to be maintained, to temporarily support two variants of an embedded system that is residing in the IT portfolio. At day 93 the additional subsystem was removed, therefore the decrease in size. Between day 141 and day 142 a part of the software product line that had been outsourced and was finished, was checked in, radically increasing the size of the software product line. So, the largest outliers were perfectly legitimate. This means that we can omit these values from our analysis to dive into other volatility signals that become perhaps invisible due to the large outliers now present in the picture.

From the daily backfired size estimates we can calculate the corresponding monthly
volatility rate with Formula 2.4. For the number of months \( t \) we use the value 1/30.5, since we take 30.5 as the average number of days in a month and therefore 1/30.5 as the number of months for 1 day. In Formula 2.4 we have to divide 1 by the intercounting duration, 1/30.5, and get 30.5 in the exponent of the size divisions. So, to monitor the volatility \( r \) for a daily difference we use Formula 2.25 using the \( \text{SizeAtDay} \) on day \( n \) and day \( n + 1 \).

\[
r = \left( \frac{\text{SizeAtDay}_{n+1}}{\text{SizeAtDay}_n} \right)^{30.5} - 1 \right) \cdot 100
\]

(2.25)

Having a high amount of volatility data available on this product line, we can now apply our previously shown analysis methods. Although we are not analyzing just the requirements volatility with these data, but the integral production volatility, this method is a very cheap possibility to observe changes compared to doing function point analyses. Daily
source code sizes are easily obtained from a source code version control system, and with our formulas easily converted to compound monthly growth rates. Volatility outliers in the thus obtained data should be inspected for root causes. We are in this manner creating maximum volatility inspection results with minimal effort.

In Table 2.12 we show the characteristics of the daily measured volatility changes expressed in monthly percentages. Of course the minimum and maximum are much lower and higher for the data that contain the previously explained outliers, and we see a change in the average that is 4.92% for the data containing the outliers and 1.69% for the cleansed data. Jones’s average for system software, 2%, is again close to our empirically found average.

<table>
<thead>
<tr>
<th>r</th>
<th>min.</th>
<th>first quartile</th>
<th>median</th>
<th>mean</th>
<th>third quartile</th>
<th>max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>-73.97%</td>
<td>0.04%</td>
<td>0.70%</td>
<td>4.92%</td>
<td>1.70%</td>
<td>571.70%</td>
</tr>
<tr>
<td>cleaned</td>
<td>-29.57%</td>
<td>0.05%</td>
<td>0.69%</td>
<td>1.69%</td>
<td>1.63%</td>
<td>39.21%</td>
</tr>
</tbody>
</table>

Table 2.12: Software product line requirements volatility characteristics.

In this case, the averages should have been lower though, since the management’s target was set to no growth at all, and preferably a code volume shrink. The shown data concerns a so-called reactive product line [36], in which many instances of similar systems are consolidated into a single system: a software product line. Of course, you need new code for that, but the expectation is that other parts of the IT portfolio can shrink. Either by code removal or by generic code that replaces multiple clones or near clones. Therefore, management desired not only merging multiple instances into a single product line, but also a decrease of the total code volume. With our metrics, it is possible to monitor the conformance of the volatility of the reactive product line to the desired overall maximum of zero percent volatility.

In these kind of portfolio assessments of the volatility it is also important to have a look at the skewness of the density function of $r$. For example, our bancassurance portfolio has a median of 0% requirements volatility, but its skewness is positive, 1.088. So the chance on growth is larger than the chance on shrinkage. This is not a problem since new functionality was added, which is in line with the overall growth of about 6% in function points. The skewness of the density function of $r$ of the cleaned data of the software product line is 1.86, therefore the chances on code volume increase are higher than the decrease that is desired by management.

Despite the conformance to Jones’s averages, the probability density function of this data also has a GPD shape, and with GPD shapes the mean and median usually differ, therefore, we recommend to create your own benchmark or use our systems benchmark as a surrogate instead of Jones’s averages. The maximum tolerance factor $P$ of cleaned data is 0.33, which can serve as a benchmark for other systems software portfolios.

**Volatility density** To further investigate the volatility rate of the software product line, we have first drawn a probability density plot of volatility data including the outliers. This density plot can be found in Figure 2.27. In this density plot it is hard to distinguish the numbers around zero percent. Therefore, we have also plotted a density plot for the data
without the outliers, since we stated the causes of these outliers earlier. This resulting density plot is the solid line in Figure 2.28. For comparison we have also included our bancassurance portfolio as a dashed line and the density plot of the governmental data as a dotted line. Figure 2.28 results in an extended benchmark for the expected probability density of the volatility for different professional environments:

- governmental environments with fixed political deadlines, continuously changing requirements through continuously changing legislation;
- product line environments with small changes and occasional high outliers;
- the volatility of the bancassurance environment which has a wider range than the software product line, but is centered around zero instead of the governmental environment that is centered around a 10% monthly requirements volatility.

Figure 2.27: Probability density plot for the software product line.
2.9.2 Volatility metrics

With the product line as an additional requirements volatility benchmark, we continue with calculating the volatility ratios $\pi$ and $\rho$. In Figure 2.29 we present the $p$-proportional volatility ratios $\pi$, calculated with Formula 2.18. The data containing the outliers is on the left-hand side, and the data without the outliers on the right-hand side. The slightly larger solid dot in both plots on the right is the overall $\pi$-ratio from beginning to end. The overall volatility from begin to end is 2.16\% per month and the overall volatility $\pi$-ratio is 0.087. In the plot on the left we can easily identify the leaps and plummets that were shown also in Figure 2.26. However, we see also some ratios that were relatively far away from the general mean. These are the observations that in an IT governance situation need to be appointed for further investigation, since they are diverting from the expected situation. To aid a portfolio manager we have plotted also the $\pi$-ratios for the
data without the outliers on the right-hand side of Figure 2.29. In the right-hand side plot it is easier to identify the days diverting from the regular volatility rate. The same plots for the requirements volatility ratio $\rho$ can be found in Figure 2.30. These plots contain the overall $\rho$-ratios from begin to end as solid dots. On the left-hand side of Figure 2.30 we plotted the $\rho$-ratios for the daily volatilities expressed in monthly rates for all data. On the right-hand side of Figure 2.30 the $\rho$-ratios are plotted for data cleansed from the outliers. Besides the ability to identify daily changes that differ from the normal situation, these plots show also a tendency to smaller changes during the observed period. This same tendency was also slightly visible in Figure 2.26, in which the slope of the line was overall inclining.

**Summary** By applying backfiring on daily size measures of the physical lines of code we monitored the volatility of a software product line. Backfiring of source code is more cost efficient than function point size analysis. Obviously we obtain more data points through the daily feed of data. With the introduction of the newly acquired volatility data line we established a systems software volatility benchmark.

![Figure 2.29: The $\pi_1$-ratio for a systems software portfolio.](image)

### 2.10 Requirements volatility dashboard

After introducing new volatility metrics and analyzing volatility data from different industries with these metrics and thus establishing volatility benchmarks, we will now propose a requirements volatility dashboard for IT governance. The interested reader can find other quantitative tools supporting IT governance in Verhoef’s article on IT governance [121]. Industry benchmarks are always helpful when little or no data is available in an organization, but when data starts to accumulate within an IT organization it is time to start to develop its own metrics. Therefore this section helps the IT governor how to represent
and organize the organization’s own data. It helps also as a comparative tool for a small number of projects or to calculate the volatility of a single project.

A requirements volatility dashboard must contain four volatility metrics: requirements volatility \( r \), volatility tolerance \( p \), the \( \pi \)-ratio and the \( \rho \)-ratio. In Table 2.13 an example volatility dashboard is shown with the volatility data from the previously shown governmental project and its subproject. For the \( \pi \) and \( \rho \)-ratios the tolerance factor \( p = 0.68 \) is used, which is the maximum tolerance factor \( P \) shown in Figure 2.11 from the low-risk bancassurance portfolio. Besides the four volatility metrics we have also shown in Table 2.13 the maximum volatility rates \( r_\pi \) and \( r_\rho \) based on the tolerance factor \( P = 0.68 \) that are used in calculations of the \( \pi \) and \( \rho \)-ratios.

<table>
<thead>
<tr>
<th>project</th>
<th>( r ) (%)</th>
<th>( p )</th>
<th>( r_\pi ) (%)</th>
<th>( \pi_{0.68} )</th>
<th>( r_\rho ) (%)</th>
<th>( \rho_{0.68} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project X</td>
<td>6.52</td>
<td>0.105</td>
<td>18.68</td>
<td>0.349</td>
<td>5.89</td>
<td>1.107</td>
</tr>
<tr>
<td>Project Y</td>
<td>25.63</td>
<td>1.433</td>
<td>18.68</td>
<td>1.372</td>
<td>6.36</td>
<td>4.030</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 2.13: Example of a requirements volatility dashboard.

Project X in Table 2.13 is the project in Figure 2.17 discussed in Section 2.6 and Project Y is the subproject which we earlier identified as being out of control. Both projects X and Y have an exact intercounting duration of 8.055 months which we will use in the following calculations. We recall from Figure 2.17 that the subproject had an initial size of 131 function points and grew in 8 months to a size of 823 function points. Project X was initially 1076 function points and surged to 1790 function points, to eventually collapse. By using Formula 2.4, project X has a volatility rate \( r \) of \( (\frac{1790}{1076})^{(1/8.055)} - 1 \cdot 100 = 6.52\% \) and the subproject an \( r \) of \( (\frac{823}{131})^{(1/8.055)} - 1 \cdot 100 = 25.63\% \). With the following table that shows values of the Lambert \( W \) function, close approximations of the dashboard values of \( \pi \) and \( \rho \) with \( p = 0.68 \) can easily be calculated.
The values in Table 2.14 can alternatively be calculated with the goal-seek function of a spreadsheet program. The function must then be set to \( x \cdot e^x \) and the goal must be set to the duration \( t \). The variable \( x \) needs to be varied to find the goal value of \( t \). The thus obtained value of \( x \) is the needed \( W(t) \).

\[
\pi_{0.68} = \frac{r_{act}}{(e^{W(0)} - 1) \cdot 100} \\
\rho_{0.68} = \frac{r_{act}}{(e^{W(0)} - 1) \cdot 100}
\]

The intercounting duration of project X and also Y was little over 8 months, so \( p \cdot t = 0.68 \cdot 8.055 = 5.48 \) and we need to interpolate over the values \( W(5) \) and \( W(6) \) from Table 2.14: 0.48 \cdot (W(6) - W(5)) + W(5) = 1.377. By using the previous results, the \( p \)-proportional danger zone for \( p = 0.68 \) is \((e^{1.377/8.055} - 1) \cdot 100 = 18.64\). The approximated \( \pi_{0.68} \)-ratio for project X then results with Equation 2.26 in 6.52/18.64 = 0.35 and for project Y in 25.62/18.64 = 1.37.

For the \( \rho_{0.68} \)-ratio we need to include the size of the projects as shown in Equation 2.27; the size estimate for project X before cancellation was 1790 function points and for Y 823 function points. The input for the Lambert \( W \) function is then \((0.68 \cdot 8.055)/\log(1790) = 0.731 \) and \((0.68 \cdot 8.055)/\log(823) = 0.816 \) for X and Y respectively. With simple interpolation on Table 2.14 and \( W(0) = 0 \) we get as a result: 0.731 \cdot (0.567 - 0) + 0 = 0.414 and 0.816 \cdot (0.567 - 0) + 0 = 0.462 respectively. The value of the denominator for the \( \rho_{0.68} \)-ratio becomes \((e^{0.414/8.055} - 1) \cdot 100 = 5.27 \) for project X and \((e^{0.462/8.055} - 1) \cdot 100 = 5.90 \) for project Y, resulting in an approximation of the \( \rho_{0.68} \)-ratio of 6.52/5.27 = 1.19 for project X and 25.62/5.90 = 4.34 for project Y. These are close to the exact values from Table 2.13 that are 0.349 and 1.372 for the \( \pi \)-ratio and 1.107 and 4.030 for the \( \rho \)-ratio for X and Y respectively.

Suppose that the governors analyzing this dashboard accept a risk that is comparable to the bancassurance industry, and therefore take the bancassurance tolerance factor of 0.68, then all projects with higher values must be colored red as displayed in Table 2.13 for project Y. On the other hand an amber color is assigned for values of \( p \) that are in the interval between the third quartile, 0.046 for the bancassurance portfolio, and the maximum, 0.68, resulting in an amber color for the \( p \) value of project X. For the other metrics in the dashboard similar coloring should be agreed upon as we have done in Table 2.13.

<table>
<thead>
<tr>
<th>( t )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W(t) )</td>
<td>0.567</td>
<td>0.852</td>
<td>1.050</td>
<td>1.202</td>
<td>1.327</td>
<td>1.432</td>
</tr>
<tr>
<td>( t )</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>( W(t) )</td>
<td>1.524</td>
<td>1.606</td>
<td>1.679</td>
<td>1.746</td>
<td>1.863</td>
<td>1.964</td>
</tr>
<tr>
<td>( t )</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>( W(t) )</td>
<td>2.053</td>
<td>2.133</td>
<td>2.205</td>
<td>2.271</td>
<td>2.332</td>
<td>2.388</td>
</tr>
</tbody>
</table>

Table 2.14: Lambert \( W \) function quick reference for intercounting duration \( t \) in months.
using the third quartiles and maximums of the bancassurance portfolio. Creating such a dashboard for all projects in an IT organization gives a quick overview of the projects out of control. The boundaries for the usage of the different colors red, amber and green are industry specific and should be based on your own internal data or our benchmarks.

---

**Figure 2.31:** Graphical representation of a volatility dashboard for $r$, $p$, $\pi_{0.68}$ and $\rho_{0.68}$.

Every project in a volatility dashboard needs a link to a figure with four plots, each plot containing data points with the internal or our bancassurance benchmark and the value of the current project highlighted. Figure 2.31 shows an example of the dashboard plots for project Y and restates our conclusions of Section 2.6. These dashboard plots of requirements volatility combined with the dashboard in Table 2.13 are insightful to compare a project with its peers on all four volatility levels and is a management tool for monitoring requirements volatility.
2.10.1 Data summary

In this paragraph we provide all statistical summaries of the volatility metrics $r$, $p$, $\pi$ and $\rho$ that we have found during the analyses of the three portfolios: bancassurance, government and systems software. Table 2.15 combines all summaries, of which many were already provided in earlier sections.

<table>
<thead>
<tr>
<th></th>
<th>min.</th>
<th>first quartile</th>
<th>median</th>
<th>mean</th>
<th>third quartile</th>
<th>max.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bancassurance</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r$ (%)</td>
<td>-23.66</td>
<td>-3.55</td>
<td>0.00</td>
<td>0.90</td>
<td>3.71</td>
<td>31.50</td>
</tr>
<tr>
<td>$p$</td>
<td>-0.10</td>
<td>-0.028</td>
<td>0.00</td>
<td>0.040</td>
<td>0.046</td>
<td>0.68</td>
</tr>
<tr>
<td>$\pi_1$</td>
<td>-0.64</td>
<td>-0.11</td>
<td>0.00</td>
<td>0.025</td>
<td>0.14</td>
<td>0.80</td>
</tr>
<tr>
<td>$\rho_1$</td>
<td>-1.43</td>
<td>-0.29</td>
<td>0.00</td>
<td>0.083</td>
<td>0.33</td>
<td>2.34</td>
</tr>
<tr>
<td><strong>government</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r$ (%)</td>
<td>-9.14</td>
<td>3.82</td>
<td>8.48</td>
<td>14.73</td>
<td>18.74</td>
<td>65.46</td>
</tr>
<tr>
<td>$p$</td>
<td>-0.044</td>
<td>0.056</td>
<td>0.15</td>
<td>0.39</td>
<td>0.31</td>
<td>1.71</td>
</tr>
<tr>
<td>$\pi_{0.68}$</td>
<td>-0.49</td>
<td>0.18</td>
<td>0.35</td>
<td>0.43</td>
<td>0.52</td>
<td>1.76</td>
</tr>
<tr>
<td>$\rho_{0.68}$</td>
<td>-1.32</td>
<td>0.64</td>
<td>1.30</td>
<td>1.73</td>
<td>1.85</td>
<td>6.98</td>
</tr>
<tr>
<td>$\pi_1$</td>
<td>-0.41</td>
<td>0.15</td>
<td>0.30</td>
<td>0.34</td>
<td>0.44</td>
<td>1.38</td>
</tr>
<tr>
<td>$\rho_1$</td>
<td>-1.03</td>
<td>0.49</td>
<td>1.02</td>
<td>1.26</td>
<td>1.27</td>
<td>5.05</td>
</tr>
<tr>
<td><strong>systems software, cleaned data set</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r$ (%)</td>
<td>-29.6</td>
<td>0.05</td>
<td>0.69</td>
<td>1.69</td>
<td>1.63</td>
<td>39.2</td>
</tr>
<tr>
<td>$p$</td>
<td>-0.35</td>
<td>0.00048</td>
<td>0.0069</td>
<td>0.014</td>
<td>0.016</td>
<td>0.33</td>
</tr>
<tr>
<td>$\pi_1$</td>
<td>-0.18</td>
<td>0.00029</td>
<td>0.004</td>
<td>0.01</td>
<td>0.01</td>
<td>0.24</td>
</tr>
<tr>
<td>$\rho_1$</td>
<td>-3.0</td>
<td>0.005</td>
<td>0.07</td>
<td>0.17</td>
<td>0.17</td>
<td>4.0</td>
</tr>
</tbody>
</table>

Table 2.15: Statistical summary of the analyzed data.

2.11 Conclusions & findings

Volatile requirements have been a problem in software engineering for decades, but volatile requirements are a fact of life and changes are often essential. In this chapter a number of converging events come together, as one of the reviewers pointed out to us.

- Projects are getting larger and larger and, with the increase in size, there is the associated increase of risk of failure.

- The nature of projects is changing. While once the largest projects were back-office batch jobs, now the more difficult, and user-intense interactive projects are eclipsing their back-office cousins in size and complexity. Worse, their close-to-the-user nature leads to more requirements volatility than the back-office systems do.

- The light on the horizon is the IT industry’s recent interest in project management and project development reporting. Vendors and their customers are looking for
CHAPTER 2. QUANTIFYING REQUIREMENTS VOLATILITY EFFECTS

ways to assess and report on project health. They are collecting information, sometimes daily, about how a project is progressing.

What the industry does not have is an accepted and useful way of interpreting the data for project managers and senior IT management. This is where our results can be helpful. This chapter proposes an interpretation needed to make project volatility data meaningful and actionable. The combination of the heightened awareness by senior business management of the likelihood and cost of project failure, the desire by the IT community to invest in project assessments, and the development of sophisticated performance and risk metrics, provide a powerful project management weapon for the IT community. In this chapter we proposed methods and metrics to help support these events.

We have developed project management methods to quickly pinpoint volatile projects that are in the danger zone of unmanageability. Besides, we have also proposed metrics to monitor the requirements volatility of IT projects. By using accurate size estimates, function point analyses, executed at different moments in the project life cycle, we were able to calculate the compound monthly requirements volatility \( r \) for different industries among which a real-world bancassurance low-risk 23.5 million dollar costing portfolio consisting of 84 projects representing together 16,500 function points. We have shown the various characteristics of requirements volatility and analyzed it for various industries: bancassurance, in-house and outsourced, systems software and civil government. We have not found a significant difference in volatility between in-house and on-site outsourced projects. Projects that were counted thrice showed a higher volatility than projects counted twice, and we encountered projects with a high requirements scrap that tended to have a higher cost per function points. Moreover, we saw that a high volatility combined with a high productivity is possible when the development process and tools are completely focused on both targets.

We have proposed a new mathematical model to identify the requirements volatility danger zone of IT projects. With this model it is possible to calculate a project’s tolerance for volatility based on size estimates at different moments in time and the duration between them. The various models are instrumental in comparing the volatility of projects with different durations and size. It turned out that short projects are less sensitive to high volatility than projects with long durations and large sizes. Therefore, the models allow for early identification of healthy and unhealthy requirements growth. This is of essential use in serving the industry’s need to monitor project progress. We have shown how to calculate a project’s tolerance for requirements growth and named it the tolerance factor \( p \). The maximum encountered tolerance factor in a portfolio was named \( P \). We introduced the \( p \)-proportional requirements volatility ratio \( \pi \) and its usage. A \( \pi_p \)-ratio larger than 1 indicates excessive growth, with \( p \) an industry or portfolio specific value. We found \( P = 0.68 \) for the bancassurance portfolio and \( P = 0.33 \) for the systems software portfolio as acceptable tolerance factors. The high-risk governmental portfolio containing the failing project had a maximum tolerance factor of \( P = 1.71 \). Because of the failing project this is a factor when requirements creep is causing havoc. All ratios and tolerance factors were summarized in Table 2.15.

We used the \( \pi \)-ratio to assess the volatility of IT portfolios of projects of different duration. With this metric we were able to pinpoint government projects that encountered excessive requirements growth. An additional metric that we proposed was named the
requirements volatility ratio $\rho$ that also takes into account the size of a project besides the duration and volatility. The $\rho$-ratio takes size into account since project duration and size have no complete correspondence. The volatility metric $\rho$ puts more emphasis on high volatility occurring at larger projects than smaller projects experiencing the same volatility. We have applied these metrics on portfolios stemming from various industries, emphasizing the applicability of our methods on projects with a constant changing nature. With our proposed metrics we were able to identify projects with unhealthy volatility.

In our analyses we established a number of benchmarks most notably a bancassurance, governmental and a systems software benchmark. We proposed a requirements volatility dashboard as a means to monitor the volatility of a portfolio of projects. All the different analyses resulted in different benchmarks for different industries. These benchmarks can be used to compare the requirements volatility of other IT portfolios. With the presented methods and metrics we have created a new tool to quantify requirements volatility with which it is possible to compare the volatility of projects of different duration and different origin and to pinpoint projects that are getting larger and larger and are, or are getting, out of control. Finally, the technical reality of software development is that we will continue to have volatile requirements. We think that our method brings us a step closer to managing that reality by being able to discriminate in a very early stage between manageable and unmanageable requirements volatility.
3.1 Introduction

It is common knowledge that investing in IT is often a risky undertaking. Frequently the targets set at the start of a project are not met, budgets are stretched, IT key performance indicators (KPIs) are grossly underestimated and business KPIs overestimated. IT has an amazingly poor reputation for estimating the costs, effort and duration of IT projects. We can think of cost and schedule overrun and not delivering the requirements agreed upon. A considerable number of IT investments even fail completely. Investing in general depends on, for example, the expected return. The more precisely you can project the probability distribution of the return on investment (ROI), the better you can predict whether an investment makes sense at all. The probability distribution of the expected return is a representation of the returns of individual projects that are part of a portfolio. Misestimating the Bermuda Triangle of project management [20], namely incorrectly estimating the costs, the schedule and the functionality to be delivered, results in a false picture for the expected return, and therefore a false picture of its probability distribution.

Incorrect estimates lead to unexpected results and therefore incorrect estimates of ROIs. Attempts to get an objective assessment of the problem of misestimation have languished while researchers pursue other areas of project measurement. It is not a coincidence that the risk of cost misestimation is an IT project risk or for short IT risk. We will propose how to quantify IT risks in such a way that proper IT investment management can emerge.

We deal with the risk management of IT-enabled business investment projects. We speak of an IT-enabled business investment project if at least 25% of the project invest-
CHAPTER 3. QUANTIFYING IT ESTIMATION RISKS

ment is spent on IT activities. Our research shows that you can analyze investments in IT-enabled projects through the assessment of the IT risks involved. But what are typical IT risks, and how can you quantify them?

We studied other fields where similar questions were treated and solved. A field with a striking resemblance to the IT world is that of perinatal epidemiology [16, 15, 52, 18]. Perinatal means around the birth; epidemiology refers, e.g., to mortality rates for diseases. Perinatal epidemiology deals with, among other issues, mortality rates around the birth. In perinatal epidemiology one searches for easily detectable indicators that will predict with high accuracy the mortality chances for just born children. Historical information is used in such predictions. It will not be a surprise that prematurely born children have a higher mortality rate than other children. There are also differences between boys and girls in this respect. Suppose the medical records of two just born children are shown to you, and you have to predict the mortality chance. One seems to have a few small problems, and the other is prematurely born. Without more precise quantifications, the first has a higher chance of surviving. Let’s make this problem more complex. A boy, born after 26 weeks, and a girl, born after 25 weeks, are brought into the hospital. They are both prematurely born, and girls are stronger than boys, but the boy is already a week older than the girl. Now what? It is no longer possible to make a prediction without a precise model that quantifies mortality rates based on indicators like gender and the gestational age in weeks, i.e. the number of weeks dating from the first day of the mother’s last menstrual period.

For IT projects similar, but less life-critical, situations exist: most IT projects suffer from misallocated budgets due to the misestimation of the project costs, also called plan inaccuracy [34]. But how does one rank projects by increasing risk, given their status? This is not really possible without decent quantifications. We will show how methods used in perinatal epidemiology transpose to the world of IT to answer such questions. With the quantification of risk, it becomes possible to quantify the expected return of a portfolio. With an ordering of projects by increasing risk, IT investments in the need of management attention will surface and audit attention is optimally allocated. More general, we state that by quantifying the IT risks for an entire IT portfolio it becomes possible to quantify the aggregate expected return of the IT portfolio, and thus it becomes known whether investing in the IT portfolio makes sense. Moreover, it becomes possible to identify the risk drivers and manage new projects based on the right values of the risk drivers with a positive influence on the correct estimation of IT KPIs. We will make the first steps towards achieving these goals by showing how to quantify and find the risk drivers of cost misestimation. Knowing the risk driver misestimation, better estimates are possible in the future, thus supporting investment decisions.

We will focus on quantifying IT risk solely. For quantifying IT value we refer to [119] and to [91], for dealing with IT risks and outsourcing deals we refer to [120], and for aggregating IT investment management to the IT portfolio level we refer to [117]. In an IT governance context IT risks are also of predominant importance. For results concerning IT risks in the light of quantifying IT governance effects we refer to [121], and for connecting IT risk with quantifying IT productivity we refer to [118]. A deep analysis on the risks of requirements volatility is published in [74].

We will deal with risks based on real-life data within a financial services organization.
In particular we will propose a method for coming to grips with a very important risk indicator for IT projects: misestimation of costs. An outcome within a selected boundary around the estimated costs is considered a correct estimate. An actual situated outside this boundary is considered a cost misestimation. This method can be effortlessly used to manage schedule misestimation or solutions underdelivery, if for the latter project size measures, e.g. function point countings, are available. The reason for considering these IT risks is that the problems surrounding those indicators are already astonishing, and getting a handle on those risks alone drastically improves the current situation of endemic capital destruction, for many problems with IT projects are caused by the misestimation of IT KPIs. For example: on an annual basis we invest about 290 billion dollars in aborted IT projects: about 150 billion in the USA and 140 billion in Europe. For comparison, the annual cost of collapsing buildings in the USA is about 4.4 billion dollars. Collapsing IT projects cost a factor of 34 more.

The reason we did not solely consider overruns, but misestimation in general, is because extensive cost underrun will result in unused money, but this money is put in a reserve. Such unused resources could otherwise have been invested profitably, e.g. in other projects.

The cost consequences of misestimations are substantial and change the picture of the business case of the investment proposal dramatically as shown in an earlier paper. The impact of time overrun and the overrun of estimated costs on the outcome of the Net Present Value (NPV) calculation are quite substantial, depending on the business case.

We stress that project failure, costs, schedule and functionality misestimation risks are not just a case of force majeure, but these risks can be influenced if the risk drivers are known by project management. By taking appropriate measures to improve the estimation process, the organization can decrease cost misestimation, duration misestimation, and poor solution delivery and increase quality. However, whether this makes sense clearly depends on the time and costs of taking such measures and the impact that these measures will have. We propose the initial steps by explaining how to diagnose the presumable causes of misestimating important IT KPIs.

In this paper we consider the quality of the estimation process and search for the drivers of the risk of the misestimation of project costs. We will analyze the project data using the logistic modeling technique. Logistic regression is a modeling technique which has been applied successfully, as stated, in medical science to answer similar questions. But logistic regression is also a common statistical tool in other sciences; for instance it is often used in marketing to investigate consumer choice behavior. In marketing the binary variable that is researched represents the buying or not of a certain product. Other areas mentioned in [88] for which logistic regression is used in neural networks are engineering, manufacturing and marketing. In the area of software fault prediction logistic regression is also used (see papers [103, 29]), to predict errors before testing. Article [70] uses logistic regression to identify explanatory variables of fault prone software modules over subsequent releases.

Logistic regression allows us to quantify the effect of a particular risk factor on the
CHAPTER 3. QUANTIFYING IT ESTIMATION RISKS

likelihood that a particular unfavorable outcome, in this paper's context misestimation of IT costs, will occur. Comparable research is conducted in [122], in which schedule estimation is analyzed with logistic regression. But in that paper, the risk drivers are based on surveys; the data we use in this paper as potential risk drivers are not perception data. In paper [122], logistic regression is not elaborately explained, but the risk drivers resulting from their research are that project managers need to be involved in schedule negotiations and adequate requirements information needs to be available at the time of estimation and staff need not to be added late to meet aggressive schedules. The latter two both imply a not unreasonable amount of requirements creep during development. A method for quantifying requirements creep effects is presented in [74], illustrated with examples from different industries. In [133] neural networks and logistic regression are compared as early warning systems for predicting project escalation, but it also depends on perception data from surveys. Besides understanding the requirements, also planning, monitoring and controlling the project resulted as significant variables in [133]. Another paper [92] also uses perception data to identify software development success factors by using logistic regression, but also uses a rather small data set of 40 projects. Our research does not use perception data, but real-world project information such as the estimated budgets and the actual costs.

The reason for considering the particular risk of cost misestimation is not merely a random choice, but inspired by many daily news headlines concerning IT projects. Often huge budget overruns are mentioned, demanding explanations from the responsible manager or government. Almost as often the root causes of these overruns cannot be pinpointed.

Therefore, risk models addressing these risks are useful and needed. Given our quantifications, organizations can focus on the drivers of risk misestimation and move into a position to trade-off whether or not costs are reasonable for mitigating or eliminating the influence of risk drivers and thus increasing the expected yield [91].

3.1.1 Organization of the paper

The rest of this paper is organized as follows.

Risk drivers and their levels We start in Section 3.2 with a discussion of the risk drivers that influence the likelihood of misestimation of project costs. From the literature a large number of IT risk drivers are known. However, which risk drivers should be considered in a specific case heavily depends on the data available within an organization. We describe the process of selecting the risk drivers. Not all project variables need to be considered as risk drivers—for instance, if causality between the binary outcome variable and explanatory variable is lacking. In other organizations different data may have been captured over time leading to the selection of other risk drivers, and also in that case our approach of selecting the risk drivers is applicable. We also discuss the measure level of the risk drivers. With respect to their measure level the risk drivers range from nominal, categorical, ordinal and interval variables to ratios.

Real-life data Readers who are interested in the structure of the data set of our real-life case study are directed to Section 3.3. Of course, all the data have been made anonymous for confidentiality. We show in detail how in general the plausibility and quality of the
available data are assessed. The analysis on the reliability and completeness of the data is an essential step in the modeling process, which should always be carried out. Subsequently, we show in Section 3.3 how we assessed the homogeneity of the data and we pay attention to the test on dependencies between variables. We demonstrate how to select a subset that combines different business units and project types, but still contains enough data from all different types. This set will be used in our subsequent analysis.

**Logistic regression explained** In Section 3.4 we explain the basic ideas behind logistic regression to IT portfolio managers, who do not have any particular statistical and mathematical background. We discuss the striking resemblance of answering questions in the field of clinical research to IT investment decision making. Both phenomena are expressed categorically as dichotomous situations, e.g. yes or no; dead or alive; misestimation or no misestimation. After that, we dive into a more thorough mathematical treatise on logistic regression and show how the parameters of the logistic regression equation are estimated by applying the maximum likelihood principle.

**Risk model building** In Section 3.5 we apply logistic regression to model IT misestimation risks. This results in models based on different misestimation intervals. We discuss the stability of the models as well as the interpretation of the risk drivers found in the models. We discover that the ratio of external developers needs to be kept small and overstaffing must be avoided to improve the quality of cost estimates. Moreover, the statistical analysis displays a maturity mismatch. In IT departments with CMM levels higher than 1, the expected improvement in cost estimation is not realized because the business department remains at a lower maturity level.

**Validity of the model found** In Section 3.6 we compare the goodness of fit of the models found for the risk of misestimation and we discuss the predictive value of the risk models that show the highest goodness of fit test scores. We also show by randomly partitioning the data set in a simulation what the distribution of the lift factor of the misestimation model is and compare it with the lift factor of the model we have found.

**Digging into the data** Section 3.7 shows the results if we analyze the causes of the risk of overrun and the risk of underrun—combined into the risk of misestimation—separately. In this section a new risk driver emerges which has opposite effects on underrun and overrun and therefore did not show up previously.

**Conclusions** Finally, in Section 3.8, we conclude the paper.

### 3.2 Misestimation risk and its risk drivers

In this section we define the risk of cost misestimation and the concept of quantifiable IT risk drivers.

#### 3.2.1 Perception of risk

Many IT risk studies concentrate around the perception of risk factors. In the paper [102] three Delphi surveys were deployed to identify a ranked list of project risk factors. These surveys were conducted in three different countries: Hong Kong, Finland, and the United States. The three panels, one for each country, consisted of experienced project managers. The Delphi survey process in the aforementioned study consisted of three phases. In the
CHAPTER 3. QUANTIFYING IT ESTIMATION RISKS

first phase of brainstorming each panelist made a list of potential risk factors, with at least six factors. The collected risk factors were compared with each other and combined into an extensive list of risk factors (without exact doubles and similar risk factors). In the next step, each panel, independently of the others, narrowed this list down to a more manageable list. Each panelist chose the most important factors, at least ten, from a random set of risk factors based on the previous extensive list. Each factor that was picked by at least 50% of the participants was taken into account in the final phase. The initial list of more than 150 items was now reduced into three smaller lists: Hong Kong ended up with 15 items, Finland with 23, and the USA compiled 17 risk factors. In the last stage each panelist ranked the remaining risk factors for their panel in order of importance, which resulted in an overall project risk-ranking list per separate panel. In order to get an international ranking, a composite ranking was made. All eleven risk factors that were present in all three ranked lists were ordered by their average relative ranks.

While this gives a good idea of what risks factors are perceived by experienced project managers, it rules out risk factors that they do not perceive. One risk factor that was left out of the composite ranking was the lack of effective project management skills. We expect that this is a typical consequence of perception research: they asked experienced project managers. In two countries, this factor was mentioned: Finland gave it the highest rank 1, and the USA ranked it 5. Due to the consensus step in the last phase of establishing the composite ranking, this important factor dropped out. With the methods that we demonstrate in this paper all risk factors for which we have data are considered to investigate whether they are significant or not. So, we prevent that important factors are missing.

A number of success and risk factors are provided by various authors and consulting companies—for instance, the top 10 success factors mentioned by Standish Group [43, 44, 45] or Capers Jones’ twelve characteristics of successful IT projects [59]. We summarized these risk drivers elsewhere [120]. Other factors often mentioned are whether new technologies are being used in a project, whether the project managers are experienced and for instance how many management layers are present in a department. What these success and risk factors share is that some of them are not easy to measure, especially not in the dawn of an IT project. And some of them adhere to the perception of risk, rather than the actual risk potential. Let us give a few examples. Jones formulates a success factor effective communications. While we all agree that this will surely help in turning any project into a success, it is difficult, but not impossible, to quantify it. When are communications effective? You will only know when a project failed due to ineffective communications. Moreover, such project data, when obtained, are subjective, since such data are usually collected through surveys with project managers. But it is plausible that even subjective data consisting of roughly quantified project information can lead to interesting research results, and when available, should be considered for inclusion in risk analyses. Proper care must be exercised if bold conclusions are drawn that are induced by subjective factors.

Standish Group formulates as the most important project success factor: executive support. Again, without doubt, it helps if important people make things happen that otherwise take much more effort. But it is hard to quantify this support, and its effectiveness is hardly predictable. In one company, a student carried out an analysis to figure out
whether success and risk factors known in the literature did correlate with actual success and failure. It turned out that none of the factors correlated with the actual situation. For obvious reasons, the data for this case study are confidential, but the result gave us another boost to dive deeper into the problems of how to design reliable models that do predict IT risks. One of the problems of the thesis was that it was hard to quantify some of the factors in an objective manner, leading to arbitrary results.

An obstacle with most risk factors that are mentioned in the literature is that they refer to software development projects. The projects under study in this article are not pure IT projects, but IT-enabled business projects. In an IT-enabled business project, the IT part measured in terms of cost is at least 25%, so the IT component does play an important role. Therefore, most of the IT risk factors are useful, but not all.

3.2.2 Problem definition

This paper proposes a method for finding the factors that determine the disparity between a cost estimation, the financial investment in a project, and its actual. All projects that are considered in this research have project costs that are determined by an estimate at the start of a project and register actual costs at the end of a project. Before we continue, we present some important definitions concerning misestimations.

Definition 1 A cost estimation $e$ is defined as being misestimated if the actual project costs belonging to that estimate fall outside the interval $[e - y\%, e + x\%]$, where $y$ and $x$ are real numbers $> 0$ and $y < 100$. The percentages $y$ and $x$ are determined or chosen by the decision maker.

Definition 2 An estimation method is any method that yields an estimate of the actual. These estimates are anything from amounts that are just guessed, to amounts based on the gut feeling of experienced managers or to amounts forecast from sophisticated models.

Definition 3 An estimation method is defined to be good, or acceptable, at level $\alpha$ if the percentage of misestimates is lower than $\alpha\%$, where $\alpha$ is some number greater than zero and less than 100. A common value of $\alpha$ is 5.

The above definitions do not require nor imply any knowledge of the estimation method. All that matters is the discrepancy between the estimate, induced by the estimation method, and the actual.

The percentages $x$ and $y$ will usually be chosen in such a way that the interval $[e - y\%, e + x\%]$ is not too large. When $y > x$, then the attitude of the decision maker is characterized as risk avoiding. If $x > y$, the attitude of the decision maker is characterized as risk seeking. In some cases $x$ and $y$ will be chosen equal. In that latter case the loss of underestimating, and reserving money that could have been used in other projects, is judged to be equal to the cost of overestimating and thus creating the need to reserve additional money.

The simplest estimator of the probability of misestimation is obtained by counting how many projects in the historical database displayed cost misestimation and dividing the result by the total number of projects.
3.2.3 IT risk factors

A very simple example of an exogenous factor in ice cream selling is the weather. It is not possible to change the weather to increase sales, but most probably sales stay lower during overcast and rainy weather than on a bright and sunny day. An example of an exogenous factor which potentially influences the risk of making a misestimation of the costs is the project category to which an investment proposal belongs. In our case study there are four categories: transactional projects, informational projects, strategic projects and infrastructure projects. Clearly, the decision maker cannot change the type of an investment project. Nevertheless, the non-influential factors, like the weather for the ice cream seller, need to be taken into account in decision making. It is important to know for IT decision makers whether or not the risk of misestimating the costs is higher for a certain investment category and how much higher that risk is.

If the quality of the estimation method is improved by changing the value of a risk driver, the chance or probability of making a misestimate will decrease as a consequence. Needless to say, the highest potentially attainable goal is to reduce the chance of misestimation to zero. But this will never be accomplished as some amount of uncertainty will always remain. However, if the quality of the estimation method improves, an excellent job has been done. Although decision makers cannot change the value of an exogenous factor it is very important for them to know the influence of such a factor.

The factors that influence the risk of making misestimates are called risk factors. These factors are called risk factors as they influence, negatively or positively, the risk of making a misestimation. We make a strict distinction between risk factors of which the values can be changed by the decision maker, the risk drivers, and the risk factors of which the values cannot be changed. The factors of the first category will be referred to as endogenous factors, but they can also be found called controllable factors, influential factors, or risk drivers. We will refer to the factors of the latter category, which cannot be changed, as exogenous factors. These can also be named uncontrollable factors, non-influential factors or extraneous factors.

There is an abundance of literature that provides us with checklists for addressing and recognizing risks in IT. First and foremost there is the qualitative approach of McFarlan [83], who published in the 1980s his first version of a still used extensive questionnaire for addressing risks in information systems. The problem with such questionnaires is that they measure IT risk perception, not the actual risk itself. Of course, they do help to address actual risks since a checklist forces you to think of aspects that would otherwise go unnoticed until the risk materializes.

3.2.4 Quantifiable risk drivers

Our focus in this paper is on potential quantifiable risk drivers for which we have data available. The following factors are examples of influential and quantifiable risk drivers of IT-enabled business investments.

- Size of the pure IT part of the project, measured in some objective manner, like using function points [3, 4, 39], lines of code, number of applications, etc.
IT Risks in Measure and Number

- Staffing essentials, like the number of FTEs working on the project, ideally broken up into activities.
- Project constraints, like a fixed date or a fixed price contract, internal, external, etc.
- Is a formal development or project management methodology being used? Notice that this is a dichotomous yes/no variable.
- The capability maturity level of the software development department or organization, which is generally known as the CMM or CMMI level.
- Scope aspects of the project, e.g. the number of milestones in the project or the number of changes in the user requirements, or the volatility of the scope expressed as a percentage of change per month [74].
- The experience level of the project team, measured by the total years of experience or by their historical percentage of successful projects.
- The experience level of the project manager, measured by the percentage of successful projects or total years of experience. We can categorize this information into two or three time slots. Think of zero to five years of experience, five to ten, or more. Again, this is a discrete variable.
- The experience level of the user, e.g. the percentage of successful projects completed for a certain user.

Of course, the aforementioned risk drivers are hunches and we do not know whether they are truly influencing the risk that we wish to get a grip on: the risk of IT cost misestimation. And of course, the resulting risk drivers depend on the quality of the data available within an organization. For instance, if there is no information on the experience levels of team members, we cannot examine this risk factor for whether or not it has any influence. For instance, the shoe size of all IT employees is an easily retrievable measure—but, as common sense suggests, not a potential explanatory variable.

In this study we investigate which potential risk drivers, for which data are available, turn out to be real risk factors, given a certain organization and its data. Subsequently, for new projects, it is possible to collect quantifiable data, and feed the data into the established models, and a quantitative impression of the risks is obtained. If the resulting calculated risks are unacceptably high, measures to mitigate the risks need to be taken. Of course, when you address risks, money is involved, and ruling out every risk will lead to exploding project costs. So given the residual risks that are left in the project, it becomes possible to carry out a risk-adjusted appraisal of the value creation of the project. In this paper we will not carry out such risk-adjusted calculations. For elaborate appraisal examples for IT investments we refer the reader to [91, 119].

3.3 The available data

In many organizations there are almost no data available for analyzing any aspect regarding information technology. For those organizations, our paper serves as an example of
what is accomplishable if only data were routinely collected. At the same time, such organizations are also in dire need of tools and techniques for coming to grips with their IT function. In such cases we advise basing the quantitative dimension for IT decision making on models that take industrial averages into account. Those models—also for certain important IT risks—are treated in papers directed towards organizations without proper data collection [74, 91, 117, 116, 120, 119].

In this paper we treat the situation in which data are available. In such a situation there is usually a structured process of data collection, analysis, and feedback. This process is fully integrated within the organization and often there is a special corporate IT department in which data are gathered, combined, scrutinized, analyzed and translated into strategic information for executive decision makers. The organization from which we obtained data is divided into reporting units from which individual and aggregate information is collected.

But also in this case there are limitations and constraints. In our study we used data from finished projects. Finished projects provide, next to their estimates, information on actual performance with respect to actual costs, actual project duration and delivered functionality. Using such historical data, plus a number of potential explanatory variables, we analyzed which variables correlated with which IT risks, if at all.

Next to project specific data we also have access to data describing the context in which the projects were carried out. In Table 3.1, we give an overview of the most prominent generic information that is often easily available at the reporting unit level. This information is also available in our data set that we will analyze later on.

<table>
<thead>
<tr>
<th>Generic information</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reporting unit (RU)</td>
<td>Logical business or regional part</td>
</tr>
<tr>
<td>Executive center</td>
<td>Logical collection of RUs</td>
</tr>
<tr>
<td>Line of business</td>
<td>Type of business for a RU</td>
</tr>
<tr>
<td>Total IT costs</td>
<td>Total IT related costs per RU</td>
</tr>
<tr>
<td>Total IT staff (TIS)</td>
<td>Total IT staff per RU</td>
</tr>
<tr>
<td>Internal IT staff</td>
<td>Breakdown of TIS</td>
</tr>
<tr>
<td>External IT staff</td>
<td>Breakdown of TIS</td>
</tr>
<tr>
<td>Management methodology</td>
<td>Is a management methodology used in a RU?</td>
</tr>
<tr>
<td>IT maturity level</td>
<td>Capability Maturity Model level of a RU</td>
</tr>
</tbody>
</table>

Table 3.1: Generic variables giving information on the environment in which projects are carried out.

Let us explain Table 3.1. A reporting unit is some logical part of an organization, either in terms of business, geographical dispersion, cross-cutting concern (e.g., security) or otherwise. An executive center is a collection of such reporting units, e.g., the Ministry of Homeland Security. Typical names for such collections are well-known. EMEA, an acronym for Europe, Middle-East and Africa, is a typical reporting unit that is found in many organizations. Regional or logical collections of reporting units are not necessarily organized accordingly.

A line of business is also a well-known term, easy to identify, and characteristic for a reporting unit. Think of MRI scanners, mobile phones, private banking, pension administration, etc. as lines of business within organizations. Reporting units spend money,
and contain staff members. This organization routinely collects a number of such aggre- 
gates. Total IT costs is an aggregate that comprises the entire annual IT activities within 
a reporting unit. Think of the IT budget of the FBI, or the IT budget of the Depart-
ment of Housing and Urban Development. Total IT staff is the pendant of total IT costs: it 
aggregates all staff members of a reporting unit that carry out IT related activities. Fur-
thermore, some breakdowns for staff are shown: internal versus external, to mention an 
obvious one. Finally two other kinds of variables reflecting quality of IT management 
and IT craftsmanship are listed: whether or not a project management tool is used and the 
CMM level of the reporting unit. If no assessment has been done on the maturity of the 
IT process, or it is not more than ad hoc, the CMM defaults to level 1.

Apart from these generic characteristics at the reporting unit level, there is also indi-
vidual project information. In Table 3.2 we summarize the most important project specific 
variables.

<table>
<thead>
<tr>
<th>Specific information</th>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reporting unit</td>
<td>ru</td>
<td>Business owner of the project, to link generic information to the project</td>
</tr>
<tr>
<td>Project category</td>
<td>pc</td>
<td>Type of investment project</td>
</tr>
<tr>
<td>Estimated Costs</td>
<td>ec</td>
<td>Estimated costs of the project</td>
</tr>
<tr>
<td>Actual Costs</td>
<td>ac</td>
<td>Actual costs of the project</td>
</tr>
<tr>
<td>Estimated Duration</td>
<td>ed</td>
<td>Expected duration of the project</td>
</tr>
<tr>
<td>Actual Duration</td>
<td>ad</td>
<td>Actual duration of the project</td>
</tr>
<tr>
<td>Estimated Project Power</td>
<td>epp</td>
<td>Size of the project in terms of average investment per month</td>
</tr>
<tr>
<td>Actual Project Power</td>
<td>app</td>
<td>Size of the project in terms of average investment per month</td>
</tr>
<tr>
<td>In-house/Outsourced</td>
<td>io</td>
<td>Either done largely in-house or outsourced</td>
</tr>
<tr>
<td>Functionality delivered</td>
<td>fd</td>
<td>Percentage of delivered functionality</td>
</tr>
</tbody>
</table>

Table 3.2: Project specific variables giving information on project performance within the 
responsible reporting unit.

We elaborate on Table 3.2. We already encountered the reporting unit (RU) in Ta-
ble 3.1, but in this context it is the business owner under whose authority the project is 
being carried out and it is denoted as ru. It is used to link the generic reporting unit in-
formation to a project. The project category pc refers to the type of project investment. 
We distinguish four categories: transactional, informational, strategic, and infrastructure 
projects. It is often easily determined whether a project fits within one of the above cate-
gories, a list that we adopted from [125]. We summarize these four types below.

- **Transactional investments**: Transactional investments provide the technology to 
  process the basic, repetitive transactions of the business, e.g., transaction process-
  ing, accounting, account management etc. The main purpose of this type of IT 
  investment is to improve efficiency and to reduce costs.
- **Informational investments**: Informational technology provides the technology for 
  managing and controlling the organization. Systems in this category typically in-
  clude systems for management and financial control, decision making, planning, 
  communication and accounting.
• Strategic investments: Strategic investments will usually be designed to add real value to the business by increasing competitive advantage, enabling the entry into new markets, or otherwise increasing or enhancing revenue streams. Examples are a system for supporting an Internet-enabled business initiative, cable TV-enabled marketing channels, etc.

• Infrastructure investments: Investments in infrastructure are often of long duration and costly, but may not, in themselves, generate any directly quantifiable financial benefits, although the business applications that depend upon the infrastructure can do so. Therefore, the investments in and maintenance of infrastructure are essential, and not always immediately profitable. This is an essential notion, already mentioned by Adam Smith in 1776 [105, book V, chapter 1, article 1]. Examples of infrastructure investments are the implementation of a new or upgraded systems management product (e.g., Unicenter or Tivoli), the implementation of a new operating system (e.g., Linux), or the roll-out of a new, private, network. Most such investments are likely to be non-discretionary.

So, each project is categorized into one of these four categories. More mundane variables are the variable $ec$ which is the expected cost, in millions of dollars, of the financial investment for the project until completion and delivery. The actual financial investment in a project is denoted by $ac$, also measured in millions of dollars to ease comparison with the estimated costs. The next variable in Table 3.2 is the expected duration $ed$, in calendar months, of the project until completion deployment. Such variables in combination with the actual values provide a rich potential for developing predictive models for schedule misestimation. The actual duration, in calendar months, of a project until completion and delivery, is denoted by $ad$. The estimated and actual project powers, which are the sizes of the project in terms of average investment per month, are denoted by respectively $epp$ and $app$. Furthermore, it is fairly easy to determine whether a project is done largely in-house or is outsourced; this information is summarized in the variable $io$. The $fd$ variable represents the percentage of required functionality that is delivered within a project.

3.3.1 The research data

The aforementioned variables are classified into response variables and explanatory variables. Each response variable is a project risk, and the explanatory variables are ideally influential, i.e. the risk drivers. As Tables 3.1 and 3.2 indicated already, the risk drivers are separated into generic aggregates at the reporting unit level and characteristics at the individual project level.

In Table 3.3 we define the outcome of the project risk of cost misestimation. In the following sections we will analyze the risk of misestimation of project costs. The analysis of the other risks such as duration misestimation is analogous to the method we describe in this paper.

The quality of the estimated costs, $cm$, is a crude dichotomous metric. In the case where the estimation of the costs was significantly misestimated, then $cm = 1$. The variable $cm$ is set to 0 if this was not the case. So, $cm = 1$ indicates that costs were
misestimated. On the other hand, \( cm = 0 \) means that the estimation was within the predefined bounds.

We denote the risk of cost misestimation, which can have any value in the range \([0, 1]\), as the variable: \( p_{cm} \). As stated in Table 3.3, we define a project having a good estimate if the actual costs have a value in the interval \([e - y\%, e + x\%]\) for a certain \( x \) and \( y \). For example, let \( x = 2.5 \) and \( y = 5 \) and let us assume that 70% of the projects were misestimated. We now want to know which factors were influential on the performance of the estimation process. Which factors can in the future positively influence the outcome of a cost estimate? Does the accuracy of the estimate depend on having a mature organization? Or on using certain project management tooling? The research presented here studies the quality of estimates after the estimates have been produced by some estimating technique. It does not consider the estimation method itself.

We are using an interval to indicate misestimation and since we do encounter underruns and overruns for our chosen interval, the method that will be used to find causes of misestimation, i.e. the method of logistic regression, will be able to find useful results. As in perinatal epidemiology, we will not research the amount of survived days, or in our case the amount of overrun or underrun, but the research interest lies in the survival itself or in our case misestimation. Because we use a bandwidth around the estimate, the model treats actuals close or equal to an estimate in the same way.

We will focus on the risk of cost misestimation in this paper. Our goal is to explain how to calculate such risks. Of course the method is applicable to the other two aforementioned risks as well.

The measure level in Table 3.3 refers to the kinds of scales and levels of measurement for each variable. As an intermezzo we explain this for the uninitiated. We already alluded to the distinction between discrete and continuous variables. Discrete, or categorical, variables are variables in which there are no intermediate values possible. Continuous variables can theoretically take any value in between two points. The estimation quality is a discrete variable: it is either zero or one. Nothing in between is possible in the world that we defined.

The measure levels for a variable refer to either one of the four basic levels: nominal, ordinal used for categorical variables and interval, or ratio, used for continuous variables. We explain the four levels for completeness.

- A variable measured on a nominal scale is a variable that does not really have any evaluative distinction. One value is really not any greater than another. A good example of a nominal variable that we encountered earlier on in this paper is gender: boy or girl. Information about nominal scales is usually coded with numbers. We used the number zero for misestimation and one if this project outcome did not materialize for the project. Of course, this choice is arbitrary and one value is not larger
CHAPTER 3. QUANTIFYING IT ESTIMATION RISKS

or better than another choice. As illustrated we use these—arbitrary—encodings in all kinds of formulas or programs to do calculations with nominal variables. The main idea is that with nominal variables there is a qualitative difference between values, not a quantitative one.

- A variable that is measured on an ordinal scale does have an evaluative connotation. One value is also in reality greater or larger or better than the other. An example is the earlier mentioned perception of risks. You can rate risks on a scale from 1 to 10, with 10 representing no risk, and 1 very high risk. With ordinal scales, we only know that higher is better than lower, only we do not know by how much. Also the scale is not constant, in the sense that the distance between 1 and 3 could vary from the distance between 7 and 10. A well-known ordinal scale is the Likert scale, often used as a five-point scale in questionnaires.

- A variable measured on an interval scale gives the same information as ordinal scales do, but interval variables have in addition an equal distance between values. The percentage of cost overrun is a good example: the difference between 10 and 20% cost overrun is the same as the difference between 60 and 70% cost overrun.

- Variables measured on a ratio scale have the same properties as ones on an interval scale have, but in addition, there is an absolute zero point.

This concludes our little discussion of project measure levels.

For the project outcome duration misestimation (dm), we have a similar dichotomous distribution: duration misestimation is present if the project experienced more than \( x\% \) overrun or less than \( y\% \) time underrun, and not if this did not occur. For functionality failure we also have a dichotomous definition, but this is a little different than the others. We consider a project to have failed in functionality delivery if less than 95\% of the functionality is delivered than was targeted for, or if the requirements creep was so high that more than 110\% is delivered. If this is the case we assign 1 to \( ff \), and otherwise 0. To ease comparison and calculations we recommend measuring functionality in function points.

Furthermore, we have a continuous variable representing the amount of overrun, or for that matter underrun. Note that the latter is possible if people estimate much too high costs, which happens for instance for political reasons. Estimated project costs can also contain biases, for instance if salami tactics are being used. For an extensive paper on forecasting quality see [34]. This variable is measured on an interval scale and it is the percentage of cost overrun. It is easily calculated by dividing the estimated costs by the difference of the actual costs minus the estimate. The project risk schedule misestimation is analogous to cost misestimation. It considers the overrun or underrun of the project schedule. Functionality misestimation considers either solutions underdelivery, or in the case of overrun, requirements creep if all functionality plus all added requirements have been delivered.

Next, we discuss the non-influential classification factors that we used. Risk factors are called classification factors if they cannot be influenced, as is the case with the weather for the earlier mentioned ice cream seller. We summarized the classification factors in Table 3.4. We already met the factor project category. We encode the four categories
IT Risks in Measure and Number

<table>
<thead>
<tr>
<th>Classification factors</th>
<th>Abbr.</th>
<th>Definition</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project category</td>
<td>pc</td>
<td>Type of project investment category</td>
<td>Nominal</td>
</tr>
<tr>
<td>Line of business</td>
<td>lob</td>
<td>Type of business for which the project is carried out</td>
<td>Nominal</td>
</tr>
<tr>
<td>Executive center</td>
<td>ec</td>
<td>The executive center in which the project is carried out</td>
<td>Nominal</td>
</tr>
</tbody>
</table>

Table 3.4: Classification factors for projects.

with the numbers 1, . . . , 4 as follows: transactional is 1, informational 2, strategic 3 and infrastructure is assigned 4. Since 1 is not better or different than 3 this is a nominal scale. The other two classification factors were already discussed before.

<table>
<thead>
<tr>
<th>Reporting Unit specific risk factors</th>
<th>Abbr.</th>
<th>Definition</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT maturity</td>
<td>cmm</td>
<td>RU’s CMM level in which the project is done</td>
<td>Ordinal</td>
</tr>
<tr>
<td>Reporting Unit Size</td>
<td>rs</td>
<td>Size of RU in terms of total IT costs</td>
<td>Ratio</td>
</tr>
<tr>
<td>Project management tool</td>
<td>pmt</td>
<td>Project management tool used or not</td>
<td>Nominal</td>
</tr>
<tr>
<td>Reporting quality of financial information</td>
<td>rgf</td>
<td>Reporting has been done or not</td>
<td>Ordinal</td>
</tr>
<tr>
<td>Development department size</td>
<td>dds</td>
<td>% of development and enhancement staff in total IT staff</td>
<td>Ratio</td>
</tr>
<tr>
<td>Internal development staff size</td>
<td>ids</td>
<td>Breakdown of DDS</td>
<td>Ratio</td>
</tr>
</tbody>
</table>

Table 3.5: Generic risk factors at the reporting unit level.

This brings us at the other risk factors: the generic risk factors that are playing a role at the reporting unit level, and the project specific risk factors. We start with the generic ones in Table 3.5. We used the reporting unit’s maturity level: the CMM level, the size of the reporting unit in terms of its total IT budget, and whether or not an overall project management tool was used within a reporting unit. Instead of using the CMM levels verbatim in the analyses, we transformed this variable. If a reporting unit has CMM level 1, the variable $cmm_{>1}$ has the value 0; if the reporting unit has level 2 or 3, the variable $cmm_{>1}$ has the value 1. We do this because there are very few instances of level 3 in our data set. Too few instances of a level can easily lead to uninterpretable or wrong conclusions later on. Furthermore, we used a rating on the quality of the reporting unit on financial information. Often this is known from internal or external audits done by accountants, and it is complemented with other indicators, like reporting to the corporate IT department about the financial state of the IT-enabled business investments themselves. Furthermore, we used the size of a reporting unit in terms of total IT staff dealing with functionality change: new and enhancements. We used a breakdown of that generic risk driver and calculated the percentage of such IT staff who were internal.

The quality of the project management is an important risk driver according to the literature [43, 44, 45, 62]. In the aforementioned references an experienced project manager was ranked high. Actual data for such risk drivers are usually not available or difficult to measure. An indication based on running project data is available in a direct manner.
In our field study, the reporting units needed to report on the expected financial performance of running projects. Think of reporting a Net Present Value, an Internal Rate of Return, Economic Value Added, Risk-adjusted Return on Capital, and other economic measures. See [119, 91] for such analyses on IT-intensive investments. The presence of these financial performance indicators implies the existence of a proper business case.

The $rqf$ risk driver depicts whether a reporting unit has been reporting on financial indicators or not. We distinguish two categories for the variable $rqf$. The levels of this variable are *no reports* for units who have not reported on their expected financial returns for any of their projects and *reports available* for those with a capability of reporting on risk and return data. To give an indication, on the running project data, we encountered about 27% of the reporting units that we assigned *no reports* to, so the majority did report on at least a few projects. Normally, the largest projects obtain more management attention than the smaller ones. So, even if only a few projects have financial reports, it will most probably add up to a large percentage of the total amount of IT investments. That is why we chose to flip the variable to true as soon as there was some financial reporting.

From Table 3.2 we selected the variables for analyses that are shown in Table 3.6; variables in Table 3.2 that refer to actuals, such as actual costs or delivered functionality are left out, since they cannot influence the quality of an estimate, as the sold amount of ice creams cannot influence the estimated amount of sold ice creams. In the next section we will assess the quality of our data set.

<table>
<thead>
<tr>
<th>Project specific risk factors</th>
<th>Abbr.</th>
<th>Definition</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimated Costs</td>
<td>ec</td>
<td>Size of the project in terms of total estimated costs</td>
<td>ratio</td>
</tr>
<tr>
<td>Estimated Duration</td>
<td>ed</td>
<td>Size of the project in terms of total estimated duration</td>
<td>ratio</td>
</tr>
<tr>
<td>Project Power</td>
<td>epp</td>
<td>Estimated size of the project in terms of average costs per month</td>
<td>ratio</td>
</tr>
<tr>
<td>In-house or Outsourced</td>
<td>io</td>
<td>Project either developed in-house or outsourced</td>
<td>nominal</td>
</tr>
<tr>
<td>Project category</td>
<td>pc</td>
<td>Transactional, infrastructure, strategic or informational</td>
<td>nominal</td>
</tr>
</tbody>
</table>

Table 3.6: Risk factors at the project level.

### 3.3.2 Data quality

The project database contained information on 221 finalized projects of a large organization totaling to a financial investment of at least $435 million—for 33 projects cost figures were missing. We narrowed the data set down to 165 projects with estimated costs of $370 million by taking the following criteria into consideration.

- Projects with missing data were not included in the research set.
- Double entries of projects were removed from the data set.

The remaining 165 projects have no missing data regarding potential project risk drivers. In a few cases, we completed missing data regarding potential generic risk drivers.
by copying these data from other projects in the same reporting unit. For two reporting units the CMM level was missing. An experienced IT auditor estimated their CMM levels. The reasons for leaving projects out of the 165-projects data set were one or more missing data fields. Almost always success criteria were missing. Missing data on the duration of a project was ranked second as the reason for removing a project from the data set. Furthermore, we noticed cost data missing, functionality data missing, or mistakes like actual costs of zero dollar. We removed three double projects. These were three relatively small projects, and they accounted for only 0.12% of the original estimated costs of $435 million of the 165 projects. Notice that in a situation when you start up this kind of IT portfolio management practice, you can save millions of dollars just by removing truly redundant projects from the portfolio [11, 117]. In this case, no projects were done twice for real, they were just reported on more than once.

**Data correctness**

The collected data are reasonable, correct and reliable. Namely, in this organization there is a data intake procedure, where unlikely values are detected and currency issues are checked and corrected. We can weed out these errors easily. Methods for detecting such values are based on comparisons with industry benchmarks. Let us explain with an example project that is estimated with $60 million project costs and whose estimated duration is about 6 months. We use a formula from [117] that is as follows:

$$tcd(d) = \frac{ru}{1800} \cdot d^{3.564}.$$  \hspace{1cm} (3.1)

In Formula 3.1 \( w \) is the number of working days in a year, \( r \) is the daily burdened compensation rate, and \( d \) is the duration of the project in calendar months. So, for a given duration we can calculate with Formula 3.1 its total cost of development \( tcd(d) \) according to industry benchmarks. In the above example, we just take \( d \) equal to 6 months, and for the specific reporting unit we use the generic data for daily rates and working days per annum to calculate the total cost of development. As an example, we take 200 working days per year and a daily burdened rate of $1,000. This results in \( tcd(6) = 65,930.83 \) dollar. What happened in the example project is that the local reporting system works in thousands of dollars, and the project that was estimated to cost $60,000 was keyed in as such. Adding an additional erroneous three zeros to the project in the reporting system led to the unlikely high cost of $60 million. In the same way currency problems and staff problems are detected, and other extreme outliers. Of course, there are many kinds of projects and for other kinds of projects other benchmark formulas were necessary. For an overview of a number of such benchmark formulas and how to create them, we refer the interested reader to [117], in which about fifty formulas are found based on industry benchmarks.

**Data plausibility**

To have a quick overview of the data we conducted the following visual inspection. In Figure 3.1 we display a histogram, a box plot, a density plot, and a Q-Q plot of the estimated costs in the data set.
A histogram displays the frequency of a variable in certain value classes, which gives us a rough indication of the distribution. A box plot represents a graphical sketch of the numerical statistics. The solid box depicts the data between the first and the third quartile, the inter-quartile range, displaying 50% of the data, which is a rather small box in our case. The line within the box represents the median, which cuts the data in half. The so-called whiskers embody the boundaries of the box plot representing the bulk of the data; data points outside these limits are often considered as outliers. A density plot depicts a smooth estimate of the distribution or density. This estimate is based on subparts of the values of variables. Our Q-Q plot, or Quantile-Quantile plot tests the data against the log-normal distribution with unit rate. If this plot is more or less a straight line, then

---

**Figure 3.1:** Visual insight into the empirical distribution of the estimated costs (indexed for confidentiality).
we have an indication that the probability density function belongs to the family of log-normal distributions. In our experience, this is the normal pattern if data have not been manipulated.

Let us discuss the four plots in this figure. First of all we plot a histogram. This one shows that the majority of the estimated financial investments are in the partition with the lowest investments. We immediately see that there are outliers, and they can be pretty large. The next plot is the box-and-whiskers plot. Fifty per cent of the data are cluttered around the median. The horizontal line segments outside the whiskers, the horizontal square brackets, are the potential outliers. There are many potential outliers, all high values. Furthermore, in the lower left plot of Figure 3.1 we estimated an empirical probability distribution of the estimated durations in the data set. We notice that this distribution has a spike around the third quartile, and a long tail to the right. The Q-Q plot is roughly a straight line, giving us further visual evidence that the data has a log-normal distribution. The above analysis has been carried out for other variables also: actual costs and estimated and actual duration. These analyses showed the same result that these KPIs have a log-normal distribution. IT KPIs often display long tails in their distribution and a log-normal distribution has a long tail. So, the data are in line with characteristics that we usually encounter.

Comparing the distributional behavior of the estimated durations with the estimated costs, we expect somehow a correlation. As elaborately treated in [118], these correlations are almost always not representing a one-to-one correspondence, since a lot of stochastic effects are in place when constructing IT-enabled business projects. An exception to this general rule is the case of the largest outliers. And also in this case, the single outlier above 60 months and the duration of the largest project are from the same project. Another interesting observation is that there are somewhat fewer outliers of the estimated duration, and they are more evenly scattered, than for the estimated investments. This is an indication that the style of IT governance in this organization is more directed towards managing on costs than on time. In an organization that manages uniformly on time, you would expect to see clouds of data around certain time frames, like 12 months. Also these aspects are useful in the analysis of distributional behavior of important KPIs of IT-intensive projects; such patterns are elaborately discussed elsewhere [121].

**Overperfect data detection**

So far, we have shown that the data in our data set of 165 projects are plausible in the sense that the KPIs display plausible characteristic distributional behavior. But we have to take overperfect data [121] into account. Overperfect data are data that are too good to be true. Suppose you plan for certain KPIs, say durations, or costs. Then it is important to know how good the predictions are. You can do this by comparing estimated and actual KPIs. Sometimes the similarities are so striking that the chances that the estimated data are retrofitted to the actual data are very large. In order to spot such effects we compare the estimated KPIs against their corresponding actuals. We do so by calculating the correlation coefficient of the estimated and actual costs, $r^2$, which has a value of 0.98. If the correlation was 1, than all actuals would have been equal to the estimates or transformed by a constant value, a case of overperfection.
In order to obtain more confidence about this, we overlaid their cumulative distribution functions, which are the integrals of the probability density functions. We notice that the two lines almost coincide. So, the hunch that the distributions are potentially similar turned out to be valid. Now is this a case of over perfect data, or is this a case of very good planning? From the data we cannot tell the difference. So additional qualitative insight is needed here. In all cases where data show such similarities it is wise to dive into the reasons. In this case it turned out that the estimated data cannot be retrofitted once the data are reported to the corporate IT department and the data are checked for unlikely values. Additionally, since the data collection is quite mature, the data producers gained a lot of experience in estimating the costs correctly. Since the costs in the data set were rounded off to thousands or millions of dollars this also led to actuals being equal to estimates. Although this data set is very suitable for the analysis presented here, more exact, not rounded off, numbers are always preferable for data analysis.

We learned from all these visual statistical plots that the KPIs under investigation are reasonable and plausible. A method for measuring the quality of an estimation method, and analyzing for political influences or other biases, is presented by Eveleens and Verhoef [34]. The method uses the Estimating Quality Factor (EQF) and they present benchmarks for the EQF. The median EQF of the estimated costs for our data set is 9.4. This is a rather good result since in their paper a median EQF of 8.5 is considered good estimation practice. Although the quality of the estimation practice in our case study is not bad at all, there is still room for improvement. By defining a cost misestimation as actual project costs falling outside the interval $[-5\%, 2.5\%]$, a considerate amount of cost estimates are classified as misestimates. Therefore, it still makes sense to search for the risk drivers of the misestimations.

3.3.3 Pooling or splitting up?

Now that we are confident that the useful data are also plausible, we address another question: can we pool all projects into one data set to analyze for the influence of risk factors?

As a first check we created Figure 3.2 to inspect the distribution of the disparity between actual costs and estimated costs. This figure displays that cost overrun as well as underrun occurred and that there are no multiple peaks in the distribution. Multiple peaks are an indicator that the data set should be split up in two or more sets each containing one peak. Each peak has its own causes and needs to be analyzed separately. Since this is not the case we continue with the entire data set.

Our data set is a so-called pooled data set: the original data are collected from various different samples; in this case from different reporting units. We must know whether the pooled data set is homogeneous with respect to the phenomenon which we want to explain: the variation in the risk of project cost misestimation. If the data set can be subdivided into subsets that are not comparable in kind of nature with respect to the explained variable, the data set is called heterogeneous. Let us explain. In our case study the data set can be subdivided in a number of ways. For example it can be subdivided along the axis of the executive centers. We then obtain three subsets of 25, 29, and 111 projects respectively. For each subset a specific influence on the risk of project cost misestimation.
can play a role which is specific to that subset which does not apply to the other subsets. Let us label that factor culture. If this culture effect exists, the three subsets are not homogeneous with respect to the phenomenon that we want to explain.

There are two ways to deal with this problem. First, we can pool the data of the three subsets and include in our model a categorical variable which can take one of the values 1, 2 or 3 and include interaction variables that allow for the combined influence of this categorical variable and other variables. If a specific EC-culture influence exists, the categorical variable and/or some interaction variables will show up in the logistic regression equation with a significant coefficient. Second, we can estimate the logistic regression equation for each of the three subsets separately. In that case we do not have to include a categorical variable and interaction variables in the three models. The result
CHAPTER 3. QUANTIFYING IT ESTIMATION RISKS

will be three different logistic regression equations for the three subsets. However, in both approaches the number of observations in each subset must be sufficiently large to carry out meaningful statistical analyses. If different causes for project cost misestimation exist across the executive centers, and if we want to search for these differences, we need enough projects of each subcategory. If a subset contains not enough data points in relation to the number of explanatory variables, it is statistically not possible to test on heterogeneity. Or, in other words if a subset does not contain enough projects it is not possible to determine whether there is a culture effect which must be taken into account in explaining the variation in the risk of misestimating project costs.

Of course, we can also subdivide the data set along the axes of lines of business (LOB), and project category (PC) to investigate the presence of LOB and PC effects. Potential subsets based on these two classification factors are:

- Subsets by line of business: In our case there are three major lines of business, with 88, 60, and 17 projects.
- Subsets by project category: In our data set there exist four categories: 51 transactional projects, 14 informational projects, 52 strategic projects and 48 infrastructure projects.

In Figure 3.3 we depict simple bar plots of the three classification factors that are present in our sample set. In the figure we include the amounts of misestimated projects for each category.

![Figure 3.3: Distribution of misestimation among potential risk factors.](image)

We notice that the bar plots display similar distributional shapes for well estimated and misestimated projects. Only slight differences for the categories with small sets of data are displayed, which differences are very likely to be induced by the small size of these data sets. But the estimation process can have differences between executive centers and also in project category or line of business.

However, we are interested in the possible combined EC/LOB/PC effects on the risk of project cost misestimation. If we subdivide the data set along the axes of the ECs, LOBs and PCs we obtain $3 \cdot 3 \cdot 4 = 36$ subsets. Each subset differs from the other subsets.
by the influence of a specific EC/LOB/PC combination, that holds for that subset and
does not apply to the other subsets. In Table 3.7 we present the number of projects and
the percentage of misestimation per subset.

It is questionable whether all subsets do contain enough data to estimate the logistic
regression equation for that influence of that subset. Table 3.7 shows that certain lines of
business do not occur in certain executive centers and that the three executive centers and
the three line of business show differences in the misestimation percentage. Therefore, it
will be hard to determine statistically whether or not the specific EC/LOB/PC combination
has an influence on the risk of project cost misestimation. We will not dive into the sample
size issue, but restrict our attention to the largest subset which contains enough projects
of the different subcategories that are left. This results in the subset defined by executive
center C and line of business 2, shaded in Table 3.7.

<table>
<thead>
<tr>
<th>Project Category</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>estimate % mis</td>
<td>n</td>
<td>100%</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0%</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>75%</td>
<td>4</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>50%</td>
<td>8</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>100%</td>
<td>1</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>62%</td>
<td>29</td>
<td>75%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>40%</td>
<td>5</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 3.7: Three-way table of the exogenous variables and the percentage of misestima-
tion for each class.

We see similar misestimation percentages in all four project categories and these cate-
gories contain sufficient projects, except for project category 2. The projects in this execu-
tive center and line of business still have a substantial amount of data points: 79 projects.
Therefore, we will continue our search for the influential factors of misestimation with
the data set of projects in executive center C, line of business 2 and project categories 1,
2 and 4. This is a data set that contains enough data points in each project category class
for searching for influences of the potential risk factor under consideration.

Note, that there are more classification variables than ec, lob, and pc among our risk
factors. We refer to the variables io: project either developed in-house or outsourced,
pmt: project management tool used or not, rrf: reporting on financial information or not, and cmm>1: CMM level higher than 1 or not in the project’s reporting unit. These
variables are dichotomous variables that can just take two values: 1 or 0. Since we have
enough projects in our subset of 79 projects of each type, we do not have to split up the
data set further. The logistic regression technique includes dummy variables in our logistic
regression model that allow for the potential influence of io, pmt, rrf, and cmm>1. If
some dichotomous variable has an influence the variable and/or some of its interaction
variables will show up in the logistic regression equation with a coefficient that differs
from zero in a statistically significant way.
As stated, a way to further split up the data is to leave out all the outsourced projects \((io)\). Outsourced projects must be left out if the risk of misestimation depends on the characteristics of the company to which the system development is outsourced, as opposed to a dependency on the company that outsources the labor. In the case of dependency on the company to which the development is outsourced, we need information about the CMM level of that company, their project management tools, and so on. For our data set, this is not the case since the estimates were made in-house and not by the outsourcing party. Therefore, estimates are dependent on the maturity level and the project management tools of the company itself. Later, we will reconsider this issue of pooling in-house and outsourced projects. As stated, if a difference in misestimation for in-house and outsourced projects exists, the variable \(io\) will show up in the logistic regression equation as a risk driver.

### 3.3.4 Dependencies between variables

Next, we investigate whether the variables in our remaining data set of 79 projects display mutual dependent behavior. This is important, because when having strongly dependent variables, one of them could show up in the analysis, and the other not. In that case we have to check which of the two is the explanatory variable. We distinguish three kinds of dependencies, because our set consists of continuous and categorical variables.

- Dependencies between pairs of continuous variables.
- Dependencies between pairs of continuous and categorical variables.
- Dependencies between pairs of categorical variables.

<table>
<thead>
<tr>
<th></th>
<th>(ec)</th>
<th>(ed)</th>
<th>(epp)</th>
<th>(rs)</th>
<th>(dds)</th>
<th>(ids)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ec)</td>
<td>1.00</td>
<td>0.75</td>
<td>0.55</td>
<td>0.26</td>
<td>-0.2</td>
<td></td>
</tr>
<tr>
<td>(ed)</td>
<td>0.75</td>
<td>1.00</td>
<td>0.31</td>
<td>0.18</td>
<td>-0.29</td>
<td></td>
</tr>
<tr>
<td>(epp)</td>
<td>0.55</td>
<td>0.31</td>
<td>1.00</td>
<td>0.51</td>
<td>-0.19</td>
<td></td>
</tr>
<tr>
<td>(rs)</td>
<td>0.26</td>
<td>0.18</td>
<td>0.51</td>
<td>1.00</td>
<td>-0.42</td>
<td>-0.07</td>
</tr>
<tr>
<td>(dds)</td>
<td>0.26</td>
<td>0.18</td>
<td>0.51</td>
<td>0.42</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>(ids)</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.42</td>
<td>-0.07</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 3.8: Correlation matrix between continuous risk drivers.

We start to inspect the interdependencies of the continuous risk drivers. To that end we calculated the correlations between continuous variables. A perfect correlation is displayed by the number 1, a perfect negative correlation is displayed by -1, and no correlation by 0. Numbers in between indicate low or high correlation depending on the displayed number. The lower left half of Table 3.8 is left empty for readability, but can of course be filled with a mirrored copy of the upper right half of the table. We spot immediately that not all risk drivers are independent. We discuss the dependencies. The estimated investment, \(ec\), correlates with the estimated duration, \(ed\), this is displayed by the value 0.75 in Table 3.8. This is what you hope to be the case, because projects with long durations usually have higher costs than short projects, where the latter display lower costs.
The ratio of development and enhancement staff to the total IT staff, $dds$, does not correlate with the ratio of internal development staff $ids$, as is displayed by the value $-0.07$. If in the logistic regression analysis two risk drivers emerge that are strongly dependent, we need to consider whether one of the two variables can be left out.

We now consider tests of independence for pairs of categorical variables, for which we use contingency tables. A contingency table represents the combined counts of the levels of two categorical variables. With the well-known statistical $\chi^2$-test we then infer whether two variables are dependent.

We illustrate the use of this $\chi^2$-test with an example. We investigated the potential dependency between the dichotomous variable indicating a higher maturity level than level 1, $cmm > 1$, and whether the reporting unit uses a project management tool. Table 3.9 depicts the observed amount of projects for each combination of the maturity level $cmm > 1$ and the variable $pmt$ indicating whether a project management tool is in use.

<table>
<thead>
<tr>
<th>$cmm &gt; 1$</th>
<th>$pmt_{not\ used}$</th>
<th>$pmt_{used}$</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>26</td>
<td>27</td>
<td>53</td>
</tr>
<tr>
<td>$&gt; 1$</td>
<td>6</td>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>total</td>
<td>32</td>
<td>47</td>
<td>79</td>
</tr>
</tbody>
</table>

Table 3.9: Contingency table with actual number of IT-enabled business investments grouped on maturity level and whether or not a project management tool is in use.

To establish whether the two factors are independent, we calculate the expected amounts of projects in each category based on Table 3.9. There are in total 79 projects of which 53 projects reside in a reporting unit with CMM level 1, so the chance of having CMM level 1 is $53/79 = 0.67$. Since there are in total 32 projects that are in a reporting unit without a project management tool in use, the frequency of $pmt_{not\ used}$ projects with CMM level 1 should be $32 \times 0.67 = 21.5$. In this manner, the expected cell counts are estimated as the products of the observed marginal totals divided by the table total. In this way we can fill another contingency table based on the basis of the assumption that both variables are independent. Table 3.10 contains all the expected amounts.

<table>
<thead>
<tr>
<th>$cmm &gt; 1$</th>
<th>$pmt_{not\ used}$</th>
<th>$pmt_{used}$</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21.5</td>
<td>31.5</td>
<td>53</td>
</tr>
<tr>
<td>$&gt; 1$</td>
<td>10.5</td>
<td>15.5</td>
<td>26</td>
</tr>
<tr>
<td>total</td>
<td>32</td>
<td>47</td>
<td>79</td>
</tr>
</tbody>
</table>

Table 3.10: Contingency table with the expected number of IT-enabled business investments grouped by maturity level and whether or not the reporting unit that creates the estimates uses a project management tool.

The Cochran conditions [22, 23] are used as a rule of thumb for whether the $\chi^2$-test can be used to test the dependency of two variables. These conditions state that 80% of the expected values in tables calculated like Table 3.10 need to be higher than 5 and all
values need to be higher than 0. For Table 3.10 we use these Cochran conditions to check whether a \( \chi^2 \)-test should be used at all: none of the cells has an expected amount of zero, and more than 80% of the cells have a value higher than 5. Therefore, it makes sense to carry out a \( \chi^2 \)-test. The test gives a \( \chi^2 \)-statistic of 3.9 and a \( p \)-value of 0.049. This calculated \( p \)-value is small, so at a 5% confidence interval we can reject the hypothesis that the maturity level is independent of whether a project resides in a reporting unit in which a project management tool is in use. We thus have to assume that the two variables are dependent on each other, a conclusion that makes sense, since, whenever the maturity level of a reporting unit increases, the usage of a project management tool will be required.

It will not always be the case that the Cochran conditions are satisfied for a contingency table with expected values. In that case, the Fisher’s Exact Test is used to determine whether there are nonrandom associations between two categorical variables. Fisher’s Exact Test calculates a \( p \)-value of 0.035 in the case of \textit{cmm} \( >1 \) and \textit{pmt}, leading to the same conclusion as before.

In this section we have analyzed the available data. We have assessed the quality and homogeneity of the data and dependencies between variables. Before we start with our search for risk drivers for misestimation, we explain the basics of logistic regression in the next section.

### 3.4 Logistic Regression

In this section we dive into the technicalities of IT risk quantification through logistic regression modeling. First, it is worthwhile to give some background on the mathematical methods underlying such analyses.

#### 3.4.1 Introduction

Logistic regression is a specialized form of regression that is designed to predict and explain a binary categorical variable rather than a metric dependent measure—for instance, what factors or combinations of factors are useful in predicting heart failure, or are significant in explaining buying, or not buying, behavior. Similar in form to regular regression, it can and must be used when the basic assumptions for normal regression, particularly normality of the independent variables, are not met. Usually regression analysis relies on strictly meeting the assumptions of multivariate normality and equal variance-covariance matrices across groups. Logistic regression does not need these strict assumptions.

The binary nature of the dependent variable means that the error term has a binomial distribution instead of a normal distribution, and it thus invalidates all testing based on the assumption of normality. The variance of the dichotomous variable is not constant, creating instances of heteroscedasticity as well. Neither of these can be remedied through ordinary transformations of the dependent or independent variables. Logistic regression was developed to specifically deal with this issue. To that end it makes use of the so called logit transformation, which will be explained further in this section, a special case of what Kendall Atkinson [8] calls “the family of folded power transformations”, in which the natural logarithm is used to create “normality by proxy”.

96
Now, compare the quest for IT risk drivers with perinatal epidemiology: one wants to predict neonatal mortality of a child within the first four weeks for live born infants with a birth weight less than 1500 grams, given the gender of the child and the gestational age of the child. In perinatal epidemiology, the dichotomous variable neonatal mortality is the outcome variable of interest, and infant gender and gestational age are the predictors of this outcome. It is relatively easy to know the gestational age or pregnancy time in weeks, and it is trivial to detect the dichotomous gender variable. Usually, some kind of regression analysis based on continuous variables would be used to fit their interrelations, if any. But if the dependent variable, neonatal mortality, can only have two values, this is no longer obvious. One therefore does not model the mean of the outcome variable itself, but the probability that the outcome variable has one of two possible values. This modeling technique is known as logistic regression. In logistic regression you directly estimate the probability of an event occurring. So, one wants to predict the probability of neonatal mortality of children, given their gender and the gestational age. Now, notice that the gender variable has a binary value: boy or girl, and the gestational age variable at birth has a limited range of different values in neonatal mortality: 24 weeks, . . . , 29 weeks. Now we want to turn this discrete input into continuous output, such as a 27.6% chance of mortality for a child with certain indicators.

Because we are just interested in probabilities between zero and one, we have to do a smart transformation to fit these discrete numbers to a value between zero and one. For that, we use the so-called logit transformation, which is crucial for the form of regression that we need for later quantifying IT risks of estimating.

Estimating uses restating probabilities as odds to calculate the logit values. Instead of using ordinary least squares, logistic regression uses the maximum likelihood method by comparing an estimated null model as baseline for a model fit with a proposed model containing the independent variables that are the potential risk drivers.

First, we define the logit transformation and then we will explain its properties.

\[
\text{logit}(p) = \log (\text{odds}) = \log \left( \frac{p}{1 - p} \right)
\]  

Let \( p \) be the probability function of the phenomenon that we are searching for, so it is a function ranging between zero and one. The range of the odds of function \( p \) is the ratio of the probability of \( p \) to that of its alternative \( 1 - p \), which is \( p/(1 - p) \) in a formula. The odds of \( p \) ranges between zero and infinity. Now, if we take the logarithm of a function ranging between zero and infinity, we end up with a function that ranges between minus infinity and plus infinity. We do so by taking the logarithm of the odds of \( p \), and hence the above expression, Formula 3.2. So the logit transforms a range between zero and one into the real numbers. The idea behind this transformation is that if we find some trend with a range we can use the inverse of the logit to bring that range back to the range of the probability function that we are searching for. The trick is thus not to model the probability of a phenomenon itself which potentially predicts values that are theoretically impossible, but to model the logit of the phenomenon, and when a relation is found by statistical means, to convert back to probabilities using the inverse of the logit.

The most well-known modeling technique that is based on this roughly sketched idea is called logistic regression analysis. We will apply logistic regression to the data of our
case study and use that to model certain IT risks, exactly like how you would model the mortality of prematurely born children.

### 3.4.2 The logistic regression model

After having explained the general idea behind logistic regression, we will now explain the necessary mathematics so that we can apply logistic regression modeling in our field study. We explain for the uninitiated how to directly estimate the probability of an event occurring using a logistic modeling technique and how the parameters of the resulting logistic regression equation can be estimated.

Let \( y_i \) denote the outcome of the event *cost misestimation* or *no cost misestimation*, which equals \( c_{mi} \), for project \( i \) with the following possible values for the actual costs \( ac \) and estimate costs \( ec \):

\[
\begin{align*}
    y_i &= c_{mi} = \\
    &\begin{cases} 
        1 & \text{if } ac < (ec - 5\%) \text{ or } ac > (ec + 2.5\%) \\
        0 & \text{if } (ec - 5\%) \leq ac \leq (ec + 2.5\%)
    \end{cases}
\end{align*}
\]

(3.3)

In our research we used the above boundary values for misestimation, but you are free to choose them otherwise. Let \( p_i \) denote the probability of cost misestimation of project \( i \). Let \( X_1, X_2, \ldots, X_n \) denote the risk factors on which the outcome of \( y_i \) depends. Let \( \beta_j \) denote the impact of risk factor \( X_j \) on \( p_i \). So \( \beta_j \) is the weight that can be given to factor \( X_j \), even if its effect is confounded by the presence of other risk factors that influence the outcome of \( y_i \). From each project of our sample we know whether or not cost misestimation occurred and we know the corresponding observations on the risk factors. So we have a sequence of observations at our disposal:

\[
(y_1, x_{11}, x_{12}, \ldots, x_{1n}),
\]

\[
(y_2, x_{21}, x_{22}, \ldots, x_{2n}),
\]

\[
\ldots
\]

\[
(y_m, x_{m1}, x_{m2}, \ldots, x_{mn})
\]

Here \( x_{ik} \) denotes the \( i \)-th observation of the explanatory variable \( X_k \). A well-known method for predicting a dependent variable from a set of independent variables is multiple-regression analysis. As stated before, multiple linear regression is not suited for modeling binary data. Let us consider the following equation for multiple linear regression

\[
p_i = \beta_0 + \beta_1 \cdot x_{i1} + \ldots + \beta_n \cdot x_{in} + u_i.
\]

(3.4)

In Equation 3.4 the disturbance term is denoted by \( u_i \). It is assumed that all the \( u_i \) are variables with zero expectation and have equal variance for all \( i \). The explanatory variables are allowed to be all kinds of variables—ordinal, categorical, continuous variables—but the dependent variable needs to be a continuous variable. The coefficients \( \beta_0, \beta_1, \ldots, \beta_n \) and the parameters of the distribution of \( u_i \) are unknown, and the problem is to obtain estimates of these unknowns. In linear regression the parameters \( \beta_0, \beta_1, \ldots, \beta_n \) of the model are estimated using the method of least squares. Doing so in this case, the difficulty is
clearly that not all possible values of \( p_i \) predicted from the model can be interpreted as probabilities as they are not constrained to fall in the interval between 0 and 1. So, the linear model can predict values which are theoretically impossible: below 0 or above 1. Moreover, the assumptions necessary for testing hypotheses in regression analysis are violated as it is unreasonable to assume the distribution of the errors to be normal if the dependent variable can only have two values. Also, other well-known multivariate statistical techniques such as linear discriminant analysis to predict the membership of a group are also not to be considered, as the assumptions necessary for applying the method are not fulfilled, i.e. multivariate normality of the explanatory variables and equal variance-covariance matrices in the two groups.

Of course, it is also possible to try to create a linear regression model which does not model the chance of project cost misestimation, but the amount of misestimated costs. Such practice is also possible in the perinatal epidemiology case. Modeling the survived number of days of prematurely born infants with a light birth weight, instead of the chance of not surviving, creates a linear regression model as opposed to the common logistic model. But the logistic model, indicating survival of the first 28 days, is of greater interest than the exact number of days that an infant has survived. For reasons similar to those that are apparent in epidemiology, we choose in the underlying case of IT risk also logistic modeling: we are mostly interested in the presence or absence of cost misestimation, and the factors that influence that outcome rather than the actual amount of misestimation.

Since it is inconvenient to model the probability directly, we apply the logistic regression model. In logistic regression we do not assume that \( p_i \) depends on the set of explanatory variables through a linear combination of these variables, but assume that the logistic transformation of \( p_i \), which we denote by logit\((p_i)\), depends on a linear combination of the explanatory variables, i.e. the risk drivers. This dependency is shown in Equation 3.5.

\[
\text{logit}(p_i) = \log \left( \frac{p_i}{1-p_i} \right) = \beta_0 + \beta_1 \cdot x_{i1} + \beta_2 \cdot x_{i2} + \ldots + \beta_n \cdot x_{in} \tag{3.5}
\]

The logarithms used in this paper are always logarithms with base \( e \), the natural logarithm. Clearly, applying a logarithm to a number is the same as measuring a number on a different scale. Remember that \( x_{ik} \) denotes the \( i \)-th observation for the explanatory variable \( X_k \) with respect to project \( i \).

In matrix notation the logit becomes

\[
\text{logit}(p_i) = \log \left( \frac{p_i}{1-p_i} \right) = \mathbf{X}_i\beta. \tag{3.6}
\]

In Equation 3.6 \( \mathbf{X}_i \) denotes the vector of the observations of the explanatory variables with respect to project \( i \): \( \mathbf{X}_i = (1, x_{i1}, \ldots, x_{in}) \) and \( \beta \) is the vector of the coefficients to be estimated: \( \beta = (\beta_0, \beta_1, \ldots, \beta_n) \). The choice of the logit transformation of \( p_i \) has two advantages. First of all, as stated before, it maps the range \([0, 1]\) onto the range \( (-\infty, \infty) \). Second, \( p_i/(1-p_i) \) can be interpreted as the odds of investment misestimation of project \( i \), which makes a direct interpretation of the regression coefficients possible. A logistic
coefficient can be interpreted as the change in the logarithmic value of the odds associated with a one-unit change in the explanatory variable. When we solve $p_i$ from Formula 3.6 we obtain the logistic regression model:

$$p_i = \frac{e^{X_i\beta}}{1 + e^{X_i\beta}}. \tag{3.7}$$

To obtain estimates of $\beta_0, \beta_1, \ldots, \beta_n$ we use the maximum likelihood method. According to this method we determine the values of $p_i$ for $i = 1, 2, \ldots, n$, which make the observed outcomes $y_1, y_2, \ldots, y_n$ most likely. The $y_i$ are a sequence of zeros and ones. Or, in other words, those estimates $\beta_i$ are selected that make the observed results $p_i$ as likely as possible.

We assume that the risk of cost misestimation of project $i$ is independent of the risk of cost misestimation of project $j$ for all $i \neq j$. In that case the likelihood function has the form presented in Equation 3.8.

$$L(\beta) = \prod_{i=1}^{n} p_i^{y_i}(1 - p_i)^{1-y_i}. \tag{3.8}$$

If we substitute $p_i$ in the right-hand side of Formula 3.8 with the $p_i$ from Formula 3.7 and take logarithms of both sides of Formula 3.8 we obtain the following function of $\beta$:

$$\log(L(\beta)) = \sum_{i=1}^{n} [y_i X_i \beta - \log(1 + e^{X_i \beta})]. \tag{3.9}$$

Formula 3.9 shows the general likelihood function. If we multiply the log-likelihood function by $-2$ we obtain the so-called deviance of the model by definition. The deviance is just a scaled log-likelihood, and both terms are often used in logistic model evaluation. To find the value of $\beta$ that maximizes Formula 3.9, denoted by $\hat{\beta}$, we set the derivative of $\log(L(\beta))$ with respect to $\beta$ equal to 0 and solve $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_n$ from the resulting system of normal equations. Because the normal equations are nonlinear in the unknowns to be solved, an iterative procedure is applied to obtain the estimates. The estimates can be obtained by the Newton-Raphson procedure if all regularity conditions are fulfilled that are needed to obtain the usual asymptotic properties of the maximum likelihood estimators. We do not go into those mathematical details, and refer the interested reader to the literature on this subject [128, 18, 85, 97]. We assume the regularity conditions to be fulfilled in our case. We have used the logistic regression procedure of the statistical package R [95] to carry out the necessary computations.

The logistic regression procedure of R yields not only $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_n$, but also the standard errors for the estimated parameters. If the sample size is sufficiently large, each regression coefficient is normally distributed by approximation. The standard error can subsequently be used to statistically test whether the estimated regression coefficient is significantly different from 0. If the coefficient does not appear significantly different from 0, using some chosen significance level (e.g. $\alpha = 5\%$), then the corresponding
explanatory variable—the risk driver—does not have an influence on the outcome of Y in a statistical sense.

The logistic regression procedure has several methods available for model selection. By model selection we mean that the procedure determines which explanatory variables from the list of available variables, IT risk factors in our case, are meaningful for including into the logistic regression equation. Coefficients are meaningful if they differ significantly from zero, given some chosen significance level (e.g., 5%). Moreover, it should not be possible to increase the likelihood function significantly by entering one of the non-selected explanatory variables into the regression equation. Basically, there are two approaches for model selection. Both methods are based on the idea of stepwise modeling. One can start with a model that only contains a constant, and at each step the explanatory variable with the highest contribution to the increase of the likelihood function is entered into the model. This method is called the forward stepwise selection method. The backward elimination method starts with entering all explanatory variables on the list into the model. Then, in a number of steps, variables are evaluated for entry or removal. To select variables for removal the likelihood ratio statistic is used.

The deviance of a logistic regression model is defined as $-2 \times$ the log-likelihood of that model and has a $\chi^2$ distribution with $N - k$ degrees of freedom, where $N$ equals the number of data points and $k$ the number of parameters in the model [49]. The null hypothesis of the deviance statistic is that the fitted model is not significantly different from a perfect model. A saturated, or perfect, model is a model that explains all variability, or in other words, it contains a binary variable for every data point in the data set. The difference in deviance between two models fitted on the same data set and similar parameters has a $\chi^2$ distribution with $k$ degrees of freedom, with $k$ the difference in number of variables between the two models. Testing for a significant difference in deviance is called the likelihood ratio test and is used to test for inclusion or exclusion of parameters in a model.

For more information on logistic regression we refer the reader to any textbook on the subject, e.g., [49]. A particularly readable introduction to logistic regression with easily understood examples is [16, 15]. In the next section we will apply logistic regression to our homogeneous data set.

### 3.5 Modeling cost misestimation risks through logistic regression

In this section we show how to find a regression formula for the risk of cost misestimation of an IT-enabled investment project using the logistic modeling technique. Formulas for the risk of schedule misestimates, functionality underdelivery and overall project failure are obtained in a similar way, but not considered in this paper. This article demonstrates the usage of logistic regression in the analysis of IT risks of misestimation with real-world project data with a focus on the risk of cost misestimation.

As in perinatal epidemiology we will model a binary variable indicating whether or not cost misestimation occurred. First, we consider the simplest case, in which there are no risk drivers considered that influence the probability of cost estimation. Following that,
we consider the case in which there is one risk driver allowed in the model. And finally, we allow all risk drivers to be considered as variables in the model.

3.5.1 The case of no risk drivers

If there are no risk drivers that influence the outcome of \( y_i \), then the probability of cost misestimation is equal for all 79 projects that we selected for analysis. In that case Formula 3.5 boils down to a very simple expression for all projects:

\[
\logit(p_i) = \log \left( \frac{p_i}{1 - p_i} \right) = \beta_0 \quad \text{for all } i. 
\]

(3.10)

Of course we still have to estimate \( \beta_0 \). We do this using the maximum likelihood method. According to this method we determine the value of all \( p_i \) that make the observed results \( y_1, y_2, \ldots, y_n \), a sequence of ones and zeros, most likely. If we want to maximize the log-likelihood expression, Function 3.9, for \( \beta_0 \), the following equation represents the model with no risk drivers:

\[
\log(L(\beta_0)) = \sum_{i=1}^{n} (y_i \beta_0 - \log(1 + e^{\beta_0}))
\]

(3.11)

The latter formula equals:

\[
\log(L(\beta_0)) = \beta_0 \cdot \sum_{i=1}^{n} y_i - n \cdot \log(1 + e^{\beta_0})
\]

(3.12)

To find \( \hat{\beta}_0 \) that maximizes Equation 3.12 we take the derivative of Equation 3.12, set it equal to zero, and solve the result for \( \beta_0 \).

\[
\frac{\partial \log L}{\partial \beta_0} = \sum_{i=1}^{n} y_i - n \cdot \frac{1}{1 + e^{\beta_0}} \cdot e^{\beta_0} = 0
\]

(3.13)

\[
n \cdot \frac{e^{\beta_0}}{1 + e^{\beta_0}} = \sum_{i=1}^{n} y_i
\]

(3.14)

\[
\frac{e^{\beta_0}}{1 + e^{\beta_0}} = \hat{p}_i = \frac{\sum_{i=1}^{n} y_i}{n} = \bar{y} = 0.6962
\]

(3.15)

In Equation 3.15, \( \bar{y} \) stands for the mean of \( y_i \) for all \( i \). In that equation we obtain \( \hat{p}_i \) directly. Now we can calculate \( \hat{\beta}_0 \) by using Equation 3.7.
\[
\Rightarrow \hat{\beta}_0 = 0.8293. \quad (3.16)
\]

So, the simplest estimator of the probability of failure is the number of displayed cost misestimations divided by the total number of projects. In our case study we defined misestimation as more than 5% underrun or more than 2.5% overrun. For our subset of 79 projects we obtain \( \bar{y} = 55/79 = 69.62\% \) misestimation, which is a very simple constant model for cost misestimation risk: given a new project, the chance of cost misestimation is 69.62%.

By substituting \( p = 55/79 \) in Formula 3.16 we obtain \( \hat{\beta}_0 = 0.8293 \) as the estimator of \( \beta_0 \) when using the logistic regression model. This is the intercept of a model where the misestimation risk is associated with a constant model, also called the null model. Indeed, if we fit a model using logistic regression with a statistical package we find an intercept of 0.8293, and a log-likelihood of \(-48.51\) and a deviance of \(97.02\). The deviance of the null model is used as a yardstick to judge whether or not other models that are found later on are an improvement. We recall that when we have found \( \hat{\beta} \) we have to transform the model using the inverse of the logit function to find the chance of misestimation. But in this simple case of no risk drivers, we already know that value: 69.62%. With Formula 3.7 we find that the model for cost misestimation risk equals to

\[
\frac{e^{0.8293}}{1 + e^{0.8293}} = \frac{55}{79} = 0.6962. \quad (3.17)
\]

In Figure 3.4, we depicted both the model from Equation 3.17 and the distribution of the presence and absence of cost misestimation. Indeed, as we can see, the model is just a constant over all the projects. The right-hand side plot is a bar plot showing the frequency of the cost misestimation risk data.

![null model diagram](image.png)

Figure 3.4: Plots of the null model and the cost misestimation risk data.
3.5.2 The case of one risk driver

With the null model we illustrated the basics of applying logistic regression. Let’s continue with making the model a little more complex. Suppose we want to know whether conducting a project in-house or outsourcing influences the misestimation risk. To that end we model the misestimation risk for the 79 projects with one categorical variable: whether a project was done in-house or outsourced. This leads to the following logistic regression model:

\[
\logit(p_i) = \log \left( \frac{p_i}{1 - p_i} \right) = \beta_0 + \beta_1 \cdot io_{1i}. \tag{3.18}
\]

In Equation 3.18 \(io_{1i}\) is a dummy variable that stands for the categorical variable representing in-house projects, \(io_{1i} = -1\), or outsourced projects, \(io_{1i} = 1\). This so-called Helmert contrast will be explained later. The probability of misestimation now depends on the variable \(io_{1i}\), whether a project was done in-house or outsourced, and therefore is not equal for all projects. If there is just one explanatory variable, the vector \(\beta\) consists of two elements: \(\beta_0\) and \(\beta_1\). Using the logistic regression procedure of the statistical package R we obtain Formula 3.19.

\[
\logit(p_i) = 0.746 - 0.235 \cdot io_{1i} \quad io_{1i} = \begin{cases} -1 & \text{if in-house} \\ 1 & \text{if outsourced} \end{cases} \tag{3.19}
\]

Therefore, \(\hat{\beta}_0 = 0.746\) and \(\hat{\beta}_1 = -0.235\). The standard error of the coefficients is 0.260 for both variables.

Figure 3.5: The in-house and outsourced aware model and the distribution of the misestimation risk.
In Figure 3.5 we depict the ensuing model with open circles for the different projects. The left-hand plot contains the new model. As we can see, it is a model with a dichotomous explanatory variable, as is logical given the two possible values for our single variable \( io_1 \). To compare the null model with the new model we also depicted the null model with the straight line. The in-house versus outsourced model is a slight improvement over the null model. Indeed, the deviance, \(-2\) times the log-likelihood, is still large, albeit slightly smaller than that of the null model: 96.21, instead of 97.02 for the null model. The purpose of the last model is that if a new project comes in, we predict the cost misestimation risk a little bit more accurately given the extra information of whether the project was done in-house or outsourced. This is only true if this is a significantly better model; we will shortly see that this is not the case. Assuming it is better, then, if a project is going to be outsourced, the risk of cost misestimation changes from

\[
p_{cm}(\text{in-house}) = \frac{e^{0.746-(0.235-1)}}{1 + e^{0.746-(0.235-1)}} = 0.727
\]

to

\[
p_{cm}(\text{outsourced}) = \frac{e^{0.746-(0.235-1)}}{1 + e^{0.746-(0.235-1)}} = 0.625
\]

Of course, these chances are equal to the proportions of the height of the black boxes in the right-hand side plot of Figure 3.5 to the width of the gray boxes. The right-hand side plot shows the distribution of the in-house and outsourced projects, in which the projects with a misestimation displayed as the black bars.

Note that a statistical analysis indicated that the added value of the variable \( io \) is not significant. The likelihood ratio test, that we showed in the end of Section 3.4, for the new model comparing to the null model equals 97.02 \(-\) 96.21 = 0.81. The \( p \)-value for this test can be calculated with the \( \chi^2 \) distribution with \( k \) degrees of freedom with \( k \) the difference in number of variables. In this case \( k = 1 \), which is the difference between a model with only an intercept and a model with an intercept and the variable \( io \). This results in a \( p \)-value of \( P\left(\chi^2(1) > 0.81\right) = 0.36 \), indicating that we cannot reject the null hypothesis that the coefficient of the variable \( io \) equals zero. Therefore, we conclude that the coefficient is most probably zero and an insignificant improvement of the model is found, and we will continue our search for a model that is more significant than the null model.

### 3.5.3 Creating a full model

Let us now consider the case in which all risk factors that potentially influence the outcome of the event of cost misestimation are taken into account. In Table 3.11 we summarize the factors that are propagated from the previous Tables 3.5 and 3.6. Because we have eliminated two classification factors to create a homogeneous set, these factors are no longer present in Table 3.11.

In Table 3.11 we give a summary of the risk drivers and factors that potentially influence the outcome of cost misestimation, \( cm \), and that will be used in our further analysis.
Since there are many explanatory variables, and many possible ways to combine them, it is a time-consuming effort to check all the possibilities. In fact, we have ten potential risk drivers, one potential risk factor which cannot be influenced (pc) and all potential interactions between each of the risk drivers and the risk factor, so a lot of possibilities to check. This procedure is automated in the statistical package R [95] that we used in this research. The idea is to start with the simplest model, i.e., the null model, and to end up with the model that contains all variables and potential interactions between variables. This occurs if two variables have, besides an individual influence also a combined influence on the misestimation probability. In the modeling process the step-by-step approach first adds variables, and if necessary later also interaction variables. A variable is added if the decrease in deviance of the model is statistically significant. In principle, statistical packages automate the process described, of adding variables, by using the likelihood ratio test based on the difference in deviance of different models. Peduzzi et al [90] state that at least 10 projects are needed for each explanatory variable in the final model. Since we are adding variables one at a time and have 79 projects, there is no problem for our analysis. Only if we end up with a model with more than eight explanatory variables do we have to reject the resulting model, and remove some of the variables before we start the analysis. The command for the analysis for the statistical package R is the following:

```r
step(nullmodel,
     scope=c(lower = ~1, upper = ~.^2 ),
     direction = "both"
)
```

Let us explain the above code snippet, since it illustrates the idea of stepwise modeling.
The function `step()` is a generic function that builds models in a stepwise fashion. The first argument `nullmodel` is an object that represents a model of the appropriate class. It does not need to be a good model, as long as it is a feasible model. So, our null model that we depicted in Figure 3.4 suffices. The next argument is a `scope` defining the range of models that needs to be examined in the stepwise search for the best model. The notation `lower = "1` means that the crudest model is only having an intercept, which is the null model. The notation `upper = "2` means that the most sophisticated model includes all variables, as well as potential interactions between variables in the model. We start with the lower model and step to the upper models until the likelihood ratio test tells us that adding variables is not going to improve the model. We can also start with a full model and remove unnecessary variables. To tell the package to try both directions, forward and backward stepwise modeling, we define the `direction` of the `step()` function to be both. The statistical package will then try forward and backward searching and show the best model. Separate forward and backward searches can also be done to check whether the two best models coincide.

Recall that we can step from lower to upper model and vice versa. There is no guarantee that we will end up with the same model. So forward and backward modeling can lead to different models. This is not too strange, since each model is characterized by its deviance, or likelihood, on the one hand, and its complexity in terms of number of variables, on the other hand. Depending on the value of the test statistic, and its corresponding $p$-value, we decide in a structured way which model is best. A start model that contains variables that are already considered risk drivers can also be helpful in finding the optimal logistic regression model. If one variable shows up in forward and another in backward modeling, it is important to check whether one of the two variables causes the effect of the other variable.

The influence of the different risk drivers is assessed by examining the coefficients of the variables in the various regression equations. The reason for assessing the regression equations is that we can detect both positive and negative influences of certain factors. Such knowledge gives you more grip on the IT function and is an instrument in management based on measured facts. In the following we will discuss different models.

### 3.5.4 Helmert treatment

It is not possible to estimate a coefficient for each level of a categorical variable, as the model becomes overparameterized with binary variables for each possible level. Therefore, categorical variables are replaced by sets of binary variables also called dummy variables, as we have already seen in the model with one variable. A particular set of dummy variables is called a set of contrasts. In statistical modeling tools it is possible to use the so-called Helmert contrasts to determine the coefficients of the dummy variables used; see Table 3.12.

The statistical package we use creates by default so-called `treatment` contrasts. In the case of treatment contrasts (see Table 3.13), one level of each categorical variable is left out and the dummy variable represents the difference between that level and the left-out level. In the case of our categorical variables `cmm>1`, `pmt` and `rqf` we leave out the lowest level, because we are interested in the change of risk when reaching a higher level. This
CHAPTER 3. QUANTIFYING IT ESTIMATION RISKS

is indeed more in line with our intuition that all projects have at least CMM level 1.

Note that the dummy variable \( cmm > 1 \) = 1 stands for CMM level 2 or 3, and that for CMM level 1 the dummy variable has the value 0. This is more natural to interpret and gives us more insight into the influence of going from CMM level 1 to a higher level. We have a similar situation with the variable \( rqf \) representing a good financial reporting capability and \( pmt \) representing the usage of a project management tool. For the variable in-house versus outsourced it is not the case that one variable is necessarily better than the other; that is why we use the Helmert treatment for that variable. The most important difference between models with Helmert contrasts and treatment contrasts is the significance of the intercept.

3.5.5 Search for a model without allowing interaction variables

We now present the model that we found using all data without any interaction variables and applying treatment contrasts to \( cmm > 1 \) and \( rqf \) in Equation 3.20. We used both forward and backward searching.

\[
\logit(p_{cm}) = 1.690 + 4.261 \cdot dds - 3.090 \cdot ids
\]

(3.20)

The deviance of this model is 84.488. If we test the difference from the null model, we will
see that it is significant: $97.020 - 84.488 = 12.53$. Then, $P(\chi^2(2) > 12.53) = 0.00019$, a significant improvement at the $\alpha = 0.001$ level when comparing to the null model. To apply Formula 3.20 for other projects, we need to know the values of the variables that are present in the formula. They are $dds$, the percentage of IT staff devoted to development and enhancement activities, and $ids$, the percentage of development and enhancement staff who are internal. With such generic information we can already gain more insight into the misestimation risk of the costs in a project.

As stated, if the sample is sufficiently large, each regression coefficient is normally distributed by approximation. This implies that we can easily assess by inspection whether the value zero falls into the 95% confidence interval of the estimated coefficient. The 95% interval is defined by the mean plus and minus 1.96 times the standard error of the estimated coefficient. The 95% and the 90% confidence interval are shown in Table 3.14.

<table>
<thead>
<tr>
<th></th>
<th>coefficient</th>
<th>95% confidence</th>
<th>90% confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1.69</td>
<td>(-0.976, 4.356)</td>
<td>(-0.547, 3.927)</td>
</tr>
<tr>
<td>$dds$</td>
<td>4.261</td>
<td>(0.475, 8.046)</td>
<td>(1.084, 7.438)</td>
</tr>
<tr>
<td>$ids$</td>
<td>-3.09</td>
<td>(-5.989, -0.191)</td>
<td>(-5.523, -0.657)</td>
</tr>
</tbody>
</table>

Table 3.14: Confidence intervals of the coefficients of the misestimation risk model [-2.5%, 1.25%].

The outcome of a logistic regression equation can be turned into a probability by the inverse logit transformation. Let $p_{cm}$ be the chance of cost misestimation risk $cm$. Then Formula 3.21 shows how the cost misestimation risk is turned into a probability.

$$ p_{cm} = \frac{e^{\text{logit}(cm)}}{1 + e^{\text{logit}(cm)}} \quad (3.21) $$

The outcome of the logistic regression Equation 3.21 is transformed into a risk probability with Equation 3.22:

$$ p_{cm} = \frac{e^{1.690+4.261 \cdot dds - 3.090 \cdot ids}}{1 + e^{1.690+4.261 \cdot dds - 3.090 \cdot ids}} \quad (3.22) $$

To provide more intuition for this equation, let us calculate the predicted maximum and minimum values of $p_{cm}$. Our data set contains a project with an $ids$ of 0 a $dds$ of 0.294. Almost a third of the IT staff were developers, and they were all external. This project yields, using the above formula, a risk of 95% for a disparity between the estimate and its actual. Another project contains an $ids$ of 1 and a $dds$ of 0.143. 14.3% of the IT staff were developers, and they were all internal. This project predicts a misestimation chance of 31%. As we see, the coefficients of the regression equation are scaled to the values of its corresponding variable. To have a more sophisticated way to assess the quality of the regression coefficients we observe upper and lower bounds of the coefficients provided by the confidence intervals. Each coefficient has a value within these boundaries with 95% and 90% reliability; see Table 3.14.

The model that we have found contains a constant term and two explanatory variables, risk drivers, of which the regression coefficients differed significantly from zero. The most
important risk driver is the percentage of the internal development staff ($ids$) with a negative regression coefficient and the second one is the percentage of development staff in the total IT staff ($dds$) with a positive regression coefficient. Increasing the percentage of internal developers means increasing the specific knowledge within the development team of the core business of the company and its specific culture and this will certainly help in making better estimates. The estimates in this company were all made by internal project managers. This conclusion was supported by the organization that supplied the data set. The internal developers make better estimates as they are much more familiar with testing environments, have better knowledge of the complexity of systems and the infrastructure, and will better judge whether potentially additional requirements are necessary.

Also the regression coefficient of +4.261 of the risk driver $dds$ can be interpreted meaningfully. It tells us that efficiency is important for good judgment. An overstaffed development project increases the amount of communication which is often not considered in the cost estimates. Conte et al. [24] discuss the relationship between the size of a team and the productivity which was confirmed by the company that provided the data. With a database of 187 projects they show that the average productivity per person drops when the team size increases. They explain this effect in terms of an increased number of communication paths, citing also the seminal work of Brooks [17]. Brooks states that an increased team size leads to a greater need to coordinate the activities of the group, thus increasing overhead at the expense of production work. As we see in our data set, this effect also influences the quality of estimates. More recent publications on the topic of complex dynamic systems [13, 14] further underline this notion of increased and more complex communication in larger development teams.

![Box plots for $ids$ and $dds$](image)

Figure 3.6: Box plots for $ids$ and $dds$ for the data set with 79 projects.

To give an idea of the common values of the variables $ids$ and $dds$, Figure 3.6 shows the box plots of these variables for the data set analyzed. Some projects have no internal staff at all, but most projects have development staff for which three quarters are internals.
The median value of $dds$ is 0.36; the highest value of the ratio of development staff to the total staff is 0.63.

In order to investigate potential interaction between variables we also carried out analyses where interactions of variables are potential risk drivers, besides the risk drivers $ids$ and $dds$. This analysis produced the same result as Equation 3.20. Apparently, on the basis of our available data, it appears that there is no interaction between the variables $dds$ and $ids$.

3.5.6 Loosening the misestimation risk definition

At this point we have shown the regression equation that we found by applying logistic regression to the available data. We have also shown how to interpret the explanatory variables in the regression equation round. Now, we will consider the stability of the regression equations. Therefore, the bandwidth that indicates budget misestimation is widened. We change the interval from $[-5\%, +2.5\%]$ to $[-10\%, +5\%]$; this results in 48 projects with a cost misestimation. The enlargement of the interval results in fewer cases of misestimation and yields Equation 3.23 and the confidence intervals in Table 3.15.

$$\logit(p_{cm}) = 1.706 - 1.914 \cdot ids$$  \hspace{1cm} (3.23)

<table>
<thead>
<tr>
<th></th>
<th>coefficient</th>
<th>95% confidence</th>
<th>90% confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1.706</td>
<td>(0.034, 3.378)</td>
<td>(0.303, 3.109)</td>
</tr>
<tr>
<td>$ids$</td>
<td>-1.914</td>
<td>(-4.111, 0.283)</td>
<td>(-3.758, -0.07)</td>
</tr>
</tbody>
</table>

Table 3.15: Confidence intervals of the coefficients of the risk model for misestimation outside $[-10\%, 5\%]$.

By changing the definition we found a new model that only incorporates $ids$ as a significant variable with $\alpha = 0, 1$ besides the intercept. We recall that $ids$ is the percentage of internal development and enhancement staff. The effect of increasing the internal staff is the same, a lower risk of misestimation as in the previous model. Although this is a simpler model, we will later see that the goodness of fit of this model is worse than for the previous model containing both $ids$ and $dds$ based on the smaller interval of good estimates.

If a variable definition of the outcome variable, like the misestimation interval, changes, be sure to check that the variable that is being explained does not contain too few true or false cases. For example, if only 2 out of 79 projects are considered to have been misestimated, this will lead to improper conclusions, because the calculations in the logistic regression process need more discriminating data for the outcome variable. A statistical rule of thumb is that we need at least about 20 to 30 values of each possible outcome of the binary variable which needs to be explained.

3.5.7 Overdispersion

In logistic regression, overdispersion indicates the presence of a larger variability in a data set than is expected from the statistical model underlying logistic regression. Overdisper-
tion is caused by small data sets or small subsets that are induced by different categorical variables as we have seen before. To prevent overdispersion we removed the categories with a small amount of projects. Also a definition that creates an outcome variable with either very few zeros or very few ones will not lead to a proper analysis. Overdispersion can be detected by dividing the deviance of the logistic model by the degrees of freedom of the logistic model found; this ratio needs to be around 1. If this ratio is improper, larger data sets are needed or a different definition of the variable explained that yields a more discriminating set of trues and false. Also, a scale factor can be introduced to reduce the variability; this scale factor induces wider confidence intervals, but retains the model found. In our case we have properly mitigated the risk of overdispersion as we have shown in previous sections by taking a subset with enough data points for each classification variable and a proper definition of the outcome variable misestimation.

3.5.8 CMM insignificant

In the misestimation formula that we found the influence of the CMM level turned out to be insignificant. We suspect that the skills needed for estimation are independent of the CMM level which is geared towards process maturity. Our data set contains a dichotomous variable that yields 0 in the case of CMM level 1 and in the case of CMM level 2 or 3 the variable has the value 1. This turned out to be a statistically insignificant variable, which was recognized by the company that supplied the data set, which we explain below. In Section 3.7, we will see that also digging deeper in the data set does not reveal an influence of the CMM level on the correct estimation of costs of an IT-intensive project.

At first sight, this outcome contradicts the definitions of the second level of the CMM model. Level 2 of the CMM model [89], labeled Repeatable, dictates that: ”It is characteristic of processes at this level that some processes are repeatable, possibly with consistent results”. But it also states that ”there could still be a significant risk of exceeding cost and time estimates”. So, correct estimation is desirable at level 2 and higher, but in the underlying case, in which CMM audits were conducted for the various reporting units, the higher CMM levels did not significantly improve the cost estimation practice. According to the definition of CMM level 2 a probability of exceeding costs still exists. Apparently this probability of cost misestimation is factual for a considerate amount of the projects in our data set. The CMM audits were done by the organization itself and not by a certified third party. Therefore the qualifications of CMM level 2 are possibly optimistic. But it is more likely that the absence of improved estimates is explained by the following. If a higher CMM level is reached, the requirements process within the IT department improves, but the business department falls behind. For instance, the business does not correctly consider testing phase efforts, leading to incorrect estimates, despite the higher CMM level. This is more widely known as a maturity mismatch [21]. A discrepancy in maturity level between the business and IT or supplier and client nullifies the expected benefits of the higher CMM level.
3.5.9 Pooling in-house and outsourced

In this section we return to the issue of pooling the in-house and outsourced projects. The data set that we have been analyzing did show significant variables, $ids$ and $dds$, with an acceptable goodness of fit. But it did not show an influence of the variable $io$, whether or not a project is outsourced. This result justifies the pooling of the data.

The same holds for the categorical variables $pmt$, $rqf$ and $cmm_{>1}$. Since these variables did not show up in the logistic regression equation there was no specific influence of one of the separate values of these variables.

3.6 Goodness of fit and predictive power

We have inferred several cost misestimation models, and now we will assess their goodness of fit and predictive power. Note that you cannot tacitly assume predictive value with such models, since they are built for analytical purposes. We need to assess the goodness of fit first.

3.6.1 Goodness of fit

In the following sections we start with the assessment of the goodness of fit of the models found. We will inspect the graphical goodness of fit as well as the calculated goodness of fit.

Graphical goodness of fit inspection

We will use the cost misestimation risk model with the interval $[-5\%, 2.5\%]$ as an example to illustrate two graphical quality plots. In the next figure we illustrate the calculated probabilities of misestimation of the model.

Figure 3.7 depicts the risk probabilities of cost misestimation versus the actual misestimations, measured after completion of the projects. To obtain more spread on the vertical axis the disparity percentages are placed on a logarithmic scale. Therefore, the projects that are close to the misestimation interval of $[-2.5\%, 5\%]$ are more visible. The vertical lines represent the interval that defines the boundaries of good estimates.

It is not possible to count exactly 79 dots in Figure 3.7, because some projects have equal values for the predicted chance of misestimation and the actual misestimation and therefore appear as one dot. Since the explanatory variables in our model are business unit specific variables, Figure 3.7 displays vertical lines of projects with equal chances, but different actuals.

We highlighted some projects in the figure, for instance project number 5. These projects realized a large saving in the estimated investment. The model predicted high chances, higher than 80%, of cost misestimation, which is a very good prediction of the model. Upon inspection of the project data, it turned out that project 5 was a combined project where software and business processes were reengineered. The high predicted risk of cost misestimation is in line with the risks of large business/software reengineering projects. The actual 40% cost savings of this project were due to proper risk mitigation.
efforts by the organization. Project 37 on the other hand displays an overrun of costs.

From the plot it seems that the larger the chance of cost misestimation, the more likely it is that there will indeed be a misestimation. But when the probability is low it seems that there is less predictive power. To make this more precise, we will plot the chance that a cost misestimation occurs given a minimal predictive value from our model.

For instance, if the model calculates a chance of between 30% and 40% of misestimation, what is the chance that an actual misestimation occurs? We measure the amount of actual misestimations that indeed emerged and took the fraction of the total number of projects where our model predicts between 30% and 40% chance of cost misestimation. In this case, there are nine projects with that chance and four of them have an actual mis-

Figure 3.7: Quality plots of individual risk probabilities of the cost misestimation risk model $[-5\%, 2.5\%]$.
IT Risks in Measure and Number

Figure 3.8: The chance of actual misestimation given a prediction from the cost misestimation model.

So, when we predict a chance of 30–40% of misestimation, there is a 44% chance that there is going to be an actual misestimation. In Figure 3.8 we depict this relation for all decimals. Indeed for higher predicted misestimation chances the chance that an actual misestimation will occur increases. In the next section we calculate the goodness of fit more formally.

Calculating goodness of fit metrics

With the deviance of our models we tested whether the misestimation models do not perform significantly worse than a perfect, saturated model. If we perform this likelihood ratio test [49] for the [−5%, 2.5%] null model, the $p$-value equals 0.07, and we need to reject the hypothesis that the model performs as well as a perfect, saturated, model. The $p$-value for this test for our [−5%, 2.5%] model, with $ids$ and $dds$ as risk drivers, is as follows: 0.236. In this case, we cannot reject the null hypothesis for any common $\alpha$ level, and conclude that we have a model that does not perform significantly worse than a perfect
For the model with the $[-10\%, 5\%]$ interval we do reject this null hypothesis for the null model and the fitted model with ids as a risk driver. As in linear regression, there exist goodness of fit metrics for logistic regression. The test of the unweighted sum of squares has a $p$-value of 0.98 for the $[-5\%, 2.5\%]$ model, indicating a good fit. The Hosmer-Lemeshow goodness of fit statistic $\hat{C}$ is calculated by ordering all projects by their predicted chance of misestimation and subsequently partitioning the projects over $k$ groups, usually $k = 10$ [49], for which $n_k$ indicates the number of projects in group $k$. For each partition the number of actual misestimations, $o_k$, and the number of expected misestimations, $e_k$, based on the chances in group $k$, are calculated. The statistic $\hat{C}$ is then calculated as follows:

$$\hat{C} = \sum_{k=1}^{10} \frac{(o_k - e_k)^2}{e_k \cdot \left(1 - \frac{e_k}{n_k}\right)}.$$  (3.24)

The Hosmer-Lemeshow goodness of fit test compares the observed and estimated expected frequencies for the $k$ groups. The value of $\hat{C}$ for the $[-5\%, 2.5\%]$ misestimation model equals 10.11246, it has a $\chi^2$ distribution with eight, $k - 2$, degrees of freedom and therefore a $p$-value of 0.257, indicating a well fitted model.

For the $[-10\%, 5\%]$ model the $\hat{C}$-statistic has a value of 6.83, with a $p$-value of 0.555; in other words: both are well fitted models.

**Banana lifting**

A tabular way to evaluate a model’s goodness of fit for classification purposes consists of simply classifying successes and errors of the model. We distinguish four kinds of successes and errors: true positives and false positives, and true negatives and false negatives. We can put this in a classification matrix, also called a confusion matrix [132, 54].

<table>
<thead>
<tr>
<th>prediction</th>
<th>actual</th>
<th>misestimate</th>
<th>correct estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>misestimate</td>
<td>true positive</td>
<td>false positive</td>
<td></td>
</tr>
<tr>
<td>correct estimate</td>
<td>false negative</td>
<td>true negative</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.16: A confusion matrix for a two-class case.

Table 3.16 illustrates the idea. True positives are projects that had an actual misestimation and are also predicted to have a misestimation. False positives are projects that have no significant disparity between the estimate and the actual, but are predicted to have a misestimation. The other half of the table can be read in a similar way. Now we must choose a probability for distinguishing a risky from a non-risky project. This probability enables us to provide values for the confusion matrix. The crux is how to choose this probability. An ideal graphical support technique for selecting this probability is the lift chart, sometimes called a banana chart because of its visual appearance [132, 54]. The lift chart shows how a response variable behaves when a prediction model is used. The chart displays three lines, a baseline that represents a random choice of projects, a perfect
prediction line and the lift curve itself induced by the model. The lift curve displays, one hopes, an increase in response rate, which is called the lift. A lift chart indicates which subset of the data contains the largest possible proportion of responses, in our case misestimations. The further the lift curve is away from the baseline, the better the performance of the model. So, in fact the lift shows how much better the model is than a random pick. To create a lift chart, instances in a data set are sorted in descending order of their probability of a misestimation. Plotting the sorted data creates a graphical depiction of the various probabilities. A lift chart is thus ideal for giving an overview of the classification power of a model.

Figure 3.9: Lift chart for the cost misestimation model.

Figure 3.9 depicts a lift chart for our cost misestimation model for the data set of 79 projects. The horizontal axis represents the inspected projects, ranked by decreasing predicted misestimation risk as predicted by our misestimation risk model. On the vertical axis we ranked the projects with an observed misestimation. The solid staircase-shaped curve represents the lift chart of our predictive model. This line is formed by sorting the predicted risks from high to low and comparing them with the observed misestimation, which is whether a project did suffer from a significant disparity between the estimate and
its actual or not. Each time an actual misestimation is detected, the line is lifted.

The dashed line with an angle of 45° is the predictive power of a random prediction: detecting 50% of the projects with misestimated costs is achieved by inspecting 50% of the projects at random. The random model is just the one where all projects have a risk probability of 50%. The dashed line segments above the prediction represent the perfect prediction: every project is predicted correctly. This means that the 55 projects inspected first are all the projects with a cost misestimation. This line turns horizontal after the 55 projects inspected. Our lift chart is enclosed within the region created by a random and a perfect pick: it is better than random, but not as good as a perfect prediction. The ideal place to be in the lift chart is near the upper left-hand corner: the speed for detecting risks is optimal, since all actual risky projects are detected first due to the ordering by misestimation chance. The lift chart of the cost misestimation model for the interval \([-5\%, 2.5\%]\) on the overall data set is reasonable. It is well above the line of random predictions for half of the data set, but overall not much better than a random pick. This is not strange, because the model is based on the same data set as it is predicting the chances for. Later we will apply this method in simulation of random subsets of our data sets.

The leftmost labeled point in the lift chart indicates that inspecting 35% of the projects ranked according to our model leads to a detection of 49% of the projects with an actual misestimation. Our model thus started reasonably with a lift of 49% when inspecting 35% of the projects, yielding a lift factor of \(49/35 = 1.4\). From that point on the lift factor diminishes: the line moves away from the ideal prediction. The next labeled point in the chart has a lift factor of only 1.13 (76% found by inspecting 67% of the projects). The lift chart tells us that if the risks of misestimation are lower, the predictive power of the model is also somewhat lower. The lift factor of a random prediction can be calculated as follows: the last inspected project is the last misestimated project to be found. This equals a lift factor of \(100\% / 100\% = 1\). If after an inspection of 50% of the projects only 25% of the misestimates are found, the lift factor for this subset equals \(25/50 = 0.5\).

We noted that ratios like \(49/35\) were instructive for assessing the quality of our lift chart. To make this more formal we introduce definitions taken from [132]. They are the recall and the precision of a lift chart. The recall is the ratio of the true positives, the predicted misestimates that are also actual misestimates in our case, divided by the sum of the true positives and the false negatives. The precision is the ratio of the true positives divided by the sum of true and false positives. We calculate the recall and precision for a lift chart by inspecting the confusion matrix. Suppose that we inspected 25% of the projects ordered with a decreasing cost misestimation risk; then Table 3.17 gives us the correct amount of true and false positives and negatives.

The recall at 25 inspected projects is \(24/(24 + 31) = 0.45\), or 45%, with a corresponding precision at 25 inspected projects of \(24/(24 + 1) = 0.96\). A summary measure for recall and precision is the so-called three-point average precision or the eleven-point average precision. This is the average precision at certain recall levels. For the three-point summary this is the average precision at 20%, 50% and 80%, and for the eleven-point average precision this is the average precision at 0%, 10%, . . . , 100%. The summary measures for this model are 84.75% and 75.10% respectively. So, the average precision of our cost misestimation model for interval \([-5\%, 2.5\%]\) is about 80%. We expected to detect \(55/79 = 0.70\) or 70% of the projects with a cost misestimation, which implies
an average lift factor of about 1.14 (80/70) in the lift chart. This average lift factor can be seen as an overall quality of lift charts. These are relative measures and are used to compare between different models for the same risk. We have computed the average lift factors on the basis of the earlier defined subsets in order to determine the most stable classification model. We depict these lift factors for the cost misestimation risk model in Table 3.18.

<table>
<thead>
<tr>
<th>average lift factor</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>risk for misestimation outside [-5%,2.5%]</td>
<td>1.148</td>
</tr>
<tr>
<td>risk for misestimation outside [-10%,5%]</td>
<td>1.088</td>
</tr>
</tbody>
</table>

Table 3.18: Average lift factors of the misestimation risk.

From Table 3.18 we observe that our misestimation models have average lift factors of 1.15 and 1.09. Our cost misestimation risk model performs in a stable manner with respect to the classification performance of the data set on which the model was built. The lift charts and metrics used so far indicate that our misestimation risk model classifies the projects decently, but it does not help us in determining the ideal amount of projects to control. To that end, we calculate the so-called $F$-measure that represents the information retrieval quality of the model [132]. This measure is defined as follows:

$$F\text{-measure} = \frac{2 \cdot recall \cdot precision}{recall + precision}$$

We compute the $F$-measure for each possible inspection amount of projects with the above formula. In Figure 3.10 we plot the number of projects inspected against their corresponding $F$-measures. A high risk detection quality is expressed by high values of the $F$-measure.

Figure 3.10 shows us that we need to inspect at least 40 projects to detect most actual misestimated projects; after 30 the $F$-measure still rises but not as fast as before the first 40 projects inspected.

With the $F$-measure we were able to obtain an indication of the ideal control amount of projects based on the risk model for cost misestimation. This indication is based on theoretical measures and past performance since the models were assessed with the historical data on which the models were built.
3.6.2 Predictive power of the cost misestimation risk models

Having a model does not imply that you can use it for anything you like. Suppose we want to use the model to predict misestimation risks of individual projects; we have to judge its predictive power first, which we will do in the following. In this section we compare the predictive quality of the models found earlier on the basis of the estimation intervals $[-5\%, 2.5\%]$ and $[-10\%, 5\%]$. The predictive quality of a model is measured by the so-called Mean Minus Log-Likelihood ($MML$), which calculates the deviation of the predictions from the original response variable [52, 18]. If the original data set is used to calculate the $MML$, it equals minus the log-likelihood divided by the size of the this
data set, which explains its name: minus the average, or mean, of the log-likelihood. The $MML$ is presented in Formula 3.25.

$$MML = - \frac{1}{N} \sum_{i} \left[ Y_i \log \hat{p}_{(-i)}(X_i) + (1 - Y_i) \log(1 - \hat{p}_{(-i)}(X_i)) \right]$$ (3.25)

In Formula 3.25 $p_{(-i)}(X_i)$ stands for the predicted chance of misestimation for project $i$ based on a misestimation model created with all data except project $i$; hence the subscripted minus $i$ in brackets. The perfect prediction of a data set has an $MML$ of zero. The worst value is $+\infty$. We consider a model as having an acceptable predictive power if its $MML$ is between zero and 0.5.

<table>
<thead>
<tr>
<th>Interval</th>
<th>MML</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[-5%, 2.5%]$</td>
<td>0.588</td>
</tr>
<tr>
<td>$[-10%, 5%]$</td>
<td>0.742</td>
</tr>
</tbody>
</table>

Table 3.19: Performance of overall models on the different subsets.

Zooming in on the models 3.19, we notice that the cost misestimation model with the small interval is performing better than the model with the larger interval. The model with the $[-5\%, 2.5\%]$ interval has an $MML$ that is almost acceptable. A larger data set is needed to draw more definite conclusions on the predictive value of the model on the individual project level. Given the available data set though, it is possible to better understand the predictive value of the models. We investigate this in more detail in the following sections.

Note that the calculation of the $MML$ through Formula 3.25 is needed if no test set is available and we resort to the original data. If a test set of projects is available those projects are used to calculate a mean minus log-likelihood. In that case the logistic regression model found on the basis of the research data set is used to predict the chance of the misestimation of the project costs of the test set.

**Simulation**

Since the $MML$ did not provide enough evidence of the predictive power for classification purposes for our model, we run a simulation. In this simulation we create random subdivisions of our 79-project data set. Note that the ideal way to assess the predictive power of a model is to use it to predict the risk of cost misestimation for a larger number of new projects and to evaluate the successes and errors. However, such a set is often not available. In that case a simulation as described in this section is an alternative. With a random subset with a size of 80% of the 79 projects we create a cost misestimation model. With this model based on about 63 projects we calculate the average of the three- and eleven-point lift factors of the remaining 20% of the data set, i.e. 16 projects. This process is repeated 10,000 times. From the 10,000 lift factors we calculate the probability density function, which is displayed in Figure 3.11. In this way we have simulated the
situations in which the research data set for constructing the model consists of 63 projects and the test set consists of 16 projects.

![lift factor probability density plot in simulation](image)

Figure 3.11: Distribution of lift factors obtained from 10,000 simulations of random subset division for which the 80% subset served to create a model and the 20% complement to calculate the lift factor.

In Figure 3.11 we have plotted the lift factor of the original model as well as the median lift factor of the randomly subdivided sets. Given a random subdivision of the data set of 79 projects, a similar lift factor is to be expected. Since we had no separate test set for testing the predictive power of our [−5%, 2.5%] model, we created random partitions to check whether the lift factor of the [−5%, 2.5%] model is an exceptional lift factor or not. As Figure 3.11 shows, the lift factor based on the entire research set is almost equal to the median of the 10,000 simulations. Apparently, the earlier calculated
lift factor of 1.15 already gave a good indication of the predictive power of the model. At this point we are in a position to conclude that our models are useful for finding the risk drivers of misestimation, because of the goodness of fit. The predictive power of the model for individual projects is rather poor as the MML displays, but for the classification of portfolios of projects it is useful as we have seen with the lift models.

3.7 Separating misestimation into overrun and underrun

In Section 3.5 we have seen that the explanatory variables for misestimation are $dds$, the ratio of enhancement and development staff to the complete IT staff, and $ids$, the ratio of internal development and enhancement staff. Now we will dig deeper into the data set and study the influences on overrun and underrun separately, which we previously combined into a misestimation that incorporates both overrun and underrun. In Figure 3.12 the differences of the definition of the risk materialization are illustrated.

![Figure 3.12: Illustration of different risk materialization definitions.](image)

The first part of Figure 3.12 illustrates the model of the misestimation already examined. In that case an actual value that is outside predefined boundaries of the estimate, indicated by the thick line, counts as a misestimation. The second part of Figure 3.12 represents the underrun situation: an actual that is located on the thin line counts as an underrun. Overrun is defined by an actual value that is located on the thin line of the third part of Figure 3.12. In all three definitions the comparison of an actual with an estimate results in a dichotomous variable fit to be used in logistic regression. The first case of logistic regression, using an interval, was extensively explained in the previous sections. As we have our data set up for logistic analyses in the previous sections, it is close to effortless to analyze alternative scenarios.

First, we examine the alternative of underrun. Recall that underrun is defined as an actual that turned out to be lower than the estimation. Using the same techniques as we elaborately discussed in this paper, we obtain the following equation by applying logistic
regression for the chance of underrun, denoted by $p_{ur}$:

$$\text{logit}(p_{ur}) = 2.148 + 0.714 \cdot r_{yf_{yes}} - 3.531 \cdot ids. \quad (3.26)$$

The standard errors for the variables in Equation 3.26 are respectively 0.948, 0.502 and 1.301. The data set for the underrun model contains 38 misestimations. Second, we calculate the logistic regression equation for the chance of overrun, denoted as $p_{or}$. Overrun is present if an actual value is higher than the estimate. This model contains 20 misestimations. The resulting equation from the logistic regression method is as follows.

$$\text{logit}(p_{or}) = -2.879 - 1.202 \cdot r_{yf_{yes}} + 6.111 \cdot dds. \quad (3.27)$$

The standard errors for the variables in Equation 3.27 are respectively 0.868, 0.664 and 2.403. What strikes one about Equations 3.26 and 3.27 is that the variables $ids$ and $dds$ each only appear in one equation. But they have the same effect on the misestimation if an interval is considered, as was elaborately discussed in Section 3.5 with Equation 3.20. Increasing the ratio of development staff increases the risk of overrun and the risk of misestimation, when considering an interval. Increasing the ratio of internal staff decreases the risk of underrun and the risk of misestimation if an interval is considered. Apparently if the overrun and underrun effects are combined, the separate causes $ids$ and $dds$ remain risk drivers in the resulting logistic regression equation.

Another striking point of Equations 3.26 and 3.27 is the appearance of the variable $r_{yf_{yes}}$. This variable has the value 1 if financial reporting is practised and 0 if this is not done. In the equation for underrun there is a higher chance of underrun if financial reporting is present. In the equation of overrun on the other hand, its presence displays a decrease in the chance on overrun. So, if financial reporting is practised, there is both a decrease in the chance of overrun and an increase in the chance of underrun. Since no other variables regarding for instance the costs or duration appear in these equations, the two samples are of a similar nature as regards size. These equations support the notion that the reporting variable is not visible in the equation for misestimation if an interval around the estimate is inspected. Apparently, if financial reporting is conducted, projects do not become compliant to target, but drop significantly below their estimate, and become underruns below the boundaries of the inspected interval. Estimates for projects with financial reporting tend to be too high, where their opposites tend to be too low. If no financial reporting is conducted, financial estimates tend to be risk seeking, and if the reporting is conducted, the estimates tend to be risk mitigating.

### 3.8 Conclusions

In this paper we have shown that logistic regression is a powerful modeling technique for investigating what risk factors influence the risk of a significant disparity between a cost estimate and its actual, for short, misestimations. The definition of significant in this context is defined by the bounds of an interval around the estimate. In this paper we mainly researched an interval of $[-5\%, +2.5\%]$ around the estimate; an actual that is situated outside this interval is considered to be misestimated. If the project costs or schedule are incorrectly estimated, wrong decisions are made at the start of the project as
regards allocation of money and staff capacity. Moreover, calculations of the Return on Investment (ROI), the Net Present Value (NPV) and Pay Back Time (PBT) are based on wrong figures and this may lead to the acceptance of unsound investment proposals. It therefore makes sense to analyze your set of IT projects and investigate for risk drivers that cause misestimations.

We have focused on the risk of falsely estimating the costs, being one of the most critical KPIs of an investment project. The authors are comfortable that the method developed for cost misestimation is applicable for investigating the significant risk drivers for project schedule misestimation and the risk of functionality underdelivery if the necessary data are available. However, we have not carried out that exercise, as the focus of the organization that provided the data is on cost management.

For our case study we found that the differences between the development environments of the reporting units explain the variety in the chances of project cost misestimation. According to the logistic regression equation found, the risk of misestimation varies within the range of 0.31 to 0.95 for the various reporting units. The chance of misestimation is independent of project specific characteristics. It only depends on the characteristics of the development environment. To be more precise, it varies with the percentage of developers in the total IT staff, the metric \( dds \), and the percentage of the development staff who are internal, the metric \( ids \). For management this information is extremely useful. It tells management that the focus needs to be on \( dds \) and \( ids \) in order to improve the estimation quality. Of course the coefficients of the regression equations that we have found are specific to the data set that we have researched, but the risk drivers found are more generally applicable.

Our most important learning experiences are listed below.

- It is very important to inspect the data carefully before applying the logistic modeling technique. It does not make any sense to use an entire data set of projects if the set is not homogeneous. In that case a regression equation will be found that does not apply to any of the homogeneous subsets of which the total data set consists. In our case we detected one subset of 79 projects, out of a total set consisting of 165 projects, which satisfied the condition of being homogeneous with respect to the research questions under consideration and consisting of sufficient data.

- The regression equation found must have a logical interpretation. In our case we found a constant term and two risk drivers for which the regression coefficients differed significantly from zero. The most important risk driver was the percentage of the internal development staff, \( ids \), and turned out to have a negative regression coefficient, and the second one was the percentage of development staff in the total IT staff (\( dds \)) with a positive regression coefficient. The question is whether one can put a meaningful interpretation on the risk drivers found and the signs of their regression coefficients. In our case the answer is yes, and the statistical conclusions were supported by the company providing the data set. Increasing the percentage of internal developers means increasing the specific knowledge within the development team of the core business of the company and its specific culture, complexity, infrastructure and requirements process. Apparently this aids in making better estimates. Also the regression coefficient of +4.448 of the risk driver \( dds \) can be
CHAPTER 3. QUANTIFYING IT ESTIMATION RISKS

interpreted meaningfully. This tells us that efficiency is important for good judgment. Projects that are overstaffed increase the amount of communication paths which is often not considered in the cost estimates. This results in a misestimation of the project costs. It is remarkable that the CMM level turned out not to be significant in the regression equations as one of the risk drivers. A significant improvement of estimations is expected on the basis of the definitions of the capability maturity model. The CMM audits were done by the company itself, and not by certified CMM auditors, making the level 2 qualifications probably optimistic. But the absence of improved estimates in higher CMM levels is more likely induced by a maturity mismatch. The higher CMM level improves requirements processes, but the business is not aware of efforts needed for testing, leading to incorrect estimates. The variable that indicates reporting on financial information turned out to have a decreasing effect on the risk of estimate overrun, and also an increasing effect on the risk of estimate underrun. When considering the more general notions of misestimation, both underrun and overrun, the effects of financial reporting cancelled each other out for the separate cases. The underlying systematics are probably best described as follows. In reporting units without reporting, projects tend to be risk seeking, and in reporting units with reporting, they are more risk avoiding.

- The model has been very useful for identifying the important and less important risk drivers in the collected data as the goodness of fit showed. To our surprise only two risk drivers turned out to have a significant influence on the estimation quality. For management this is valuable and useful information. It tells management that it must first of all focus on these two risk drivers to improve the estimation quality and management can neglect the other potential risk factors for the time being, for instance an effort to go from CMM level 1 to CMM level 2 to improve the estimation practice.

- The predictive power of the model for individual projects is not acceptable as the \textit{MML} displays, but for the classification of portfolios of projects it is useful as we have seen with the lift models. To reduce the cost of auditing, our models provide nonrandom selections of projects that have the highest chance of misestimation problems. This aids in focusing attention first on the most risky projects in terms of the largest chance of misestimated project costs or project duration.

We stress that the coefficients of the formulas presented in this paper cannot be used verbatim by other organizations and are specific to the organization providing the data. However, the risk drivers found are in our opinion points of interest and attention in IT governance in other organizations, especially those in the financial services industry. The conclusions presented are useful as guidelines even in the absence of the data necessary for constructing logistic regression models. Moreover, the methods explained in this paper for identifying the drivers for IT risks are universally applicable for obtaining your own IT risk models.

Capturing misestimation risks implies, among other things, that given a set of easily retrieved indicators as regards an IT project and the environment in which it is carried out, a prediction can be made as to how large the IT risks will be, so that proper measures can be taken to mitigate them.
IT risico’s in maat en getal

Uit analyses van de huidige kredietcrisis blijkt dat het nemen van risico’s, zonder te weten hoe groot de kans is op aanzienlijke schade of verlies, kan leiden tot een globale crisis. Banken die niet goed op de hoogte waren van de onzekere waarde van hun hypotheekportefeuilles moesten na het omvallen van de huizenmarkt in de Verenigde Staten grote bedragen afschrijven op hun balans. Dit vergrootte de algehele onzekerheid op de financiële markten, leidend tot de huidige kredietcrisis.

Het nemen van risico’s is een logisch onderdeel van ondernemen. Bij een duurzaam ondernemersbeleid worden doorgaans gecalculeerde risico’s genomen in plaats van lukrake beslissingen gebaseerd op onderbuikgevoelens. Er wordt nagegaan op welke manier de productiefactoren zo optimaal mogelijk ingezet kunnen worden, of kansen op verlies verkleind kunnen worden. In de traditionele economie zijn arbeid, kapitaal en natuurlijke hulpbronnen de productiefactoren. Met behulp van historische data, zoals bijvoorbeeld gegevens over ziekteverzuim, levensduur van een machine, aantal millimeters regen kan een werkgever met enige zekerheid de verwachte inzet door werknemers voorspellen, wanneer een machine vervangen moet worden en hoeveel water bij het waterbedrijf ingekocht moet worden om een veld met kroppen sla voldoende groot te laten worden.

Vandaag de dag is software een belangrijke productiefactor, maar historische gegevens of beschrijvende of voorspellende modellen ontbreken doorgaans. Met een grote regelmaat worden grote IT projecten voorpagina nieuws als blijkt dat kosten ruimschoots overschreden worden en de projecten zelf onbeheersbaar blijken doordat de specificaties maar blijven wijzigen.

Het EQUITY project beoogt door middel van haar onderzoek meer rationele beslissingen te bewerkstelligen daar waar IT een factor speelt en onderbuikgevoelens te vermijden. Dit proefschrift doet verslag van onderzoek naar het kwantificeren van IT risico’s. Daarbij ligt in hoofdstuk 2 de focus op de mate waarin instabiele projecteisen een project onbeheersbaar maken en in hoofdstuk 3 wordt geanalyseerd welke factoren de discrepantie tussen verwachte en daadwerkelijke IT kosten verklaren.

In veel bedrijven en overheidsinstellingen zijn geen analyses mogelijk door het ontbreken van gegevensverzamelingen over IT projecten. De gegevensverzamelingen die
gebruikt zijn in dit proefschrift zijn allemaal uit de praktijk afkomstig van verschillende overheden, en bedrijven uit diverse industrieën. Door gebruik te maken van deze praktijkdata zijn de uit dit onderzoek resulterende modellen direct toepasbaar en resulterende benchmarks beschikbaar, ook voor bedrijven die gegevensverzamelingen ontberen.

Instabiele eisen

Een belangrijke oorzaak voor mislukking en verspilling bij grote projecten is de indiening van aanvullende functionaliteitseisen tijdens de projectuitvoering. Doorgaans worden bij een IT-project aan het begin de eisen van het op te leveren product vastgesteld in het programma van eisen. Deze eisen heten bevroren te zijn tot oplevering, totdat door voortschrijdend inzicht, wijziging in wetgeving of andere exogene factoren, het slot van de kluis met vastgestelde eisen smelt en er toch achteraf wijzigingen nodig blijken. Dit is een herkenbaar moment voor iedere ITer en zeker goed verdedigbaar: op een product dat voldoet aan vervallen wetgeving, maar niet aan de nieuwe gewijzigde wetgeving zit immers niemand te wachten. Maar hoe vaak, en hoeveel nieuwe eisen, ook wel requirements genoemd, en wijzigingen van deze requirements kun je toestaan voordat een project onderschrijvingsregels raakt door continue wijzigingen van het op te leveren product? Projecten met een onbeheerste groei van specificaties leiden gemakkelijk tot het falen van een project, doordat het onduidelijk is en blijft wat er uiteindelijk bij de eindstreep verwacht wordt: de ontwikkeling van de software wordt onbeheersbaar.

Op welk moment moet er dan besloten worden geen nieuwe eisen of wijzigingen van de eisen van het af te leveren product meer toe te staan? Het rente-op-rente model biedt soelaas. Om te bepalen hoeveel groei tijdens een project nog gezond is grijpen we naar een eenvoudig model uit de economie: samengestelde interest of rente-op-rente. Als je 100 euro op een spaarrekening zet, keert de bank na een jaar 3 euro rente uit en verhoogt daarmee het bedrag op je spaarrekening. Het jaar daarop krijg je 3% rente over de 103 euro en keert de bank je 3 euro en 9 cent uit. 9 cent meer rente dan in het eerste jaar. Een veelgebruikte rekenregel in de economie om te weten hoeveel jaar het duurt om een spaarbedrag te laten verdubbelen bij een bepaald rentepercentage is de 72-regel. De formule voor deze rekenregel is terug te vinden in geschriften uit de 15e eeuw [87] en luidt als volgt:

\[ t = \frac{72}{r} \]  

Deel 72 door het jaarlijkse rentepercentage \( r \) en de formule geeft bij benadering het aantal jaren dat je geld moet achterblijven bij de bank om het te verdubbelen bij een gegeven rentepercentage \( r \). De 100 euro is bij een rentepercentage van 3% dus na 24 jaar verdubbeld.

Naar analogie van het samengestelde interestmodel introduceerde Capers Jones [60] een model om de groei van de hoeveelheid werk uit te drukken in maandelijkse percentages. Door bij oplevering van een project, of tussentijds, de omvang te meten en dit te delen door de omvang aan het begin van een project, dat is nadat initieel het pakket van eisen bevroren is, kan aan de hand van het aantal tussenliggende maanden berekend worden
wat de groei per maand was. Hiermee kunnen de groeipercentages van projecten onderling vergeleken worden. In aanvulling op de door Jones vermelde groeipercentages per maand, laten we zien in hoofdstuk 2 hoe je kunt berekenen bij welke percentages je in de gevarenzone terecht komt en er alleen nog maar extra werk bij komt en nauwelijks werk wordt opgeleverd zonder uitzicht op afronding van het project. Als voorbeeld laten we in Figuur 2.17 de groeipercentages zien van een overheidsproject dat uiteindelijk mislukt is.

Figuur 2.17 laat van één project en van een subproject de metingen zien op verschillende momenten; aan het begin, na een paar maanden en na 8 maanden. De horizontale as geeft het tijdsverloop van het project aan. De verticale as geeft de omvang van het project in functiepunten weer. Een functiepunt is een veel gebruikte maat voor de omvang van IT-projecten. Zoals men kan zien is het project in 8 maanden tijd gegroeid van ruim 1000 functiepunten naar bijna 1800 functiepunten. Dit betekent dat de projectomvang per maand 6,5% is gegroeid, acht maanden lang. Als we alleen naar de eerste twee maandelijks maand bekijken, groeide het project zelfs met 11,9% per maand. De voornaamste veroorzaaker van deze groei was het getoonde subproject, dat in die periode van 8 maanden groeide van 131 naar 823 functiepunten. Dit is maandelijks groei van maar liefst 25,6% voor het subproject! In de eerste twee maandelijks maand groeide het subproject zelfs 65,5% per maand.

Na de eerste tussentijdse meting had men eigenlijk de functionele eisen van het project moeten fixeren. Echter de groei van requirements heeft uiteindelijk verder doorgevoerd, wat geleid heeft tot een onbeheersbaar en mislukkend project. Met de formules uit hoofdstuk 2 kan voor iedere doorlooptijd van een project een gevarenzone berekend worden. Deze gevarenzone ligt voor kortdurende projecten bij een hoger maandelijks groeipercentage verder weg dan bij langdurende projecten. Een project dat 10 maanden lang 10% groei per maand ondervindt, heeft daar namelijk meer last van dan een project dat 2 maanden lang 10% groei per maand heeft. Door te kijken hoe dicht de waargenomen maandelijks groei bij die gevarenzone is gekomen, kunnen projecten van verschillende duur beter met elkaar vergeleken worden. Ook wordt in de formules rekening gehouden met de omvang van het project. Voor een project van grote omvang geldt dat ook bij een relatief laag maandelijks groeipercentage de gevarenzone al snel wordt bereikt.

De initiële eisen van bijvoorbeeld het OV-chipkaart project zijn onderhevig aan veel veranderingen. Helaas hebben wij niet de nodige informatie om vast te stellen of dit project daadwerkelijk in de gevarenzone ligt, maar het staat buiten kijf dat de veranderende requirements zorgen voor grote problemen. Door de technieken en benchmarks uit hoofdstuk 2 in de toekomst toe te passen bij tussentijdse audits kan beoordeeld worden of op dezelfde voet met gelijkblijvende groei doorgewerkt kan worden, of dat er ingegrepen moet worden. Dit kan gedaan worden door de groei te beheugen of zelfs aan te sturen op het laten inkrimpen van het project. Naast het hierboven beschreven gouvernementele project wordt in hoofdstuk 2 in twee andere bedrijfsssectoren de requirements volatiliteit geanalyseerd. Ten eerste wordt de projectportfolio van een grote financiële instelling geanalyseerd en ten tweede analyseren we de groei van specificaties van embedded software van een software product line.

De technieken en data uit hoofdstuk 2 kunnen ook gebruikt worden om over afgeronde projecten inzicht te krijgen hoeveel groei nog toelaatbaar was en geabsorbeerd kon worden binnen de organisatie. Indien de audits waren uitgevoerd bij de grootschalige
overheidsprojecten waar we de afgelopen jaren kennis van hebben kunnen nemen, dan hadden op basis van deze *early warnings* tijdig maatregelen genomen kunnen worden om verspillingen te voorkomen.

**Schatten van projectkosten**

In hoofdstuk 3 wordt uit de doeken gedaan hoe met logistische regressie de bepalende factoren voor misschatten van projectkosten gevonden kunnen worden. De statistische techniek logistische regressie wordt in andere vakgebieden veel toegepast. Zo wordt in de geneeskunde deze methode gebruikt om na te gaan welke factoren van invloed zijn op de sterfte van pasgeboren baby’s. In de logistische regressie is er één te verklaren variabele met twee mogelijke waarden, 0 of 1, en een aantal potentieel verklarende variabelen met een verschillende bereik. In de perinatale epidemiologie is de te verklaren variabele het wel of niet overleven van de eerste vier weken na de geboorte. De mogelijk verklarende variabelen in dat onderzoeksgebied zijn de leeftijd in weken sinds de laatste menstruatie van de moeder, het geslacht van de baby en het gewicht van de baby. Met behulp van een historische gegevensverzameling van de te verklaren en verklarende variabelen en logistische regressie is het mogelijk de bepalende factoren te achterhalen.

In het beschreven onderzoek is gekeken welke factoren van invloed zijn op een misschating van de projectkosten. Een schatting van de kosten is daarbij gedefinieerd als een misschating indien de actuele kosten de geschatte kosten met meer dan 2,5% overschrijden of indien ze meer dan 5% lager uitvallen dan geschat. Kosten die uiteindelijk hoger blijken dan geschat zijn problematisch voor de financiële situatie van een bedrijf, maar indien de kosten lager uitvallen dan geschat dan zijn ten onrechte te veel financiële middelen gealloceerd die voor andere projecten gebruikt hadden kunnen worden. Daarom is het belangrijk te analyseren welke factoren bepalend zijn voor een misschating.

Een gegevensverzameling van 79 projecten met een IT component van een grote financiële instelling vormde de basis van het onderzoek. De factoren waarvan gegevens beschikbaar waren om te bepalen of ze van invloed zijn op het ontstaan van misschattingen waren onder andere de geschatte kosten, de geschatte doorlooptijd, het volwassenheidsniveau van de business unit op IT gebied uitgedrukt in het Capability Maturity Model (CMM) niveau, of er gerapporteerd wordt over financiële gegevens, het percentage ontwikkelaars op de IT afdeling en het percentage interne ontwikkelaars. De laatste twee factoren bleken bepalend voor een misschating van de kosten. Bij een hoger percentage interne ontwikkelaars neemt de kans op misschattingen af. De bedrijfsspecifieke kennis neemt daardoor toe, de kennis over cultuur, complexiteit, infrastructuur en het requirements proces. Dit helpt blijkbaar in het maken van betere schattingen en daarmee minder misschattingen.

Een lager percentage ontwikkelaars op een IT afdeling verlaagt de kans op misschattingen. Dit duidt er op dat efficiëntie belangrijk is, overbezette afdelingen hebben meer communicatielijnen. Deze extra communicatielijnen worden vergeten bij het schatten van kosten en resulteren in misschattingen.

Opmerkelijk genoeg bleek het volwassenheidsniveau van een business unit, het CMM niveau, niet van invloed op de kans op misschating. Een verbetering van schattingen werd
wel verwacht gezien de definitie van het CMM model. Een verklaring kan zijn dat de CMM audits zijn uitgevoerd door het betreffende bedrijf zelf en niet door gecertificeerde auditors. Daardoor zijn de CMM niveau’s mogelijk optimistisch ingeschat. Veel waarschijnlijker is echter dat de verbetering niet werd behaald wegens een zogeheten maturity mismatch. Aan de IT kant worden bij een hoger CMM niveau onder andere verbeteringen ingevoerd in het requirements management proces. Indien de business echter nalaat de daardoor noodzakelijke verbeteringen door te voeren bij het testproces, dan wordt het effect van de verbeteringen aan de IT-kant teniet gedaan. Deze maturity mismatch leidt tot het ontbreken van een verbetering in het percentage misschattingen bij hogere CMM niveaus.

De variabele die aangaf of er gerapporteerd wordt over financiële informatie bleek in eerste instantie geen invloed te hebben op misschattingen wanneer tegelijkertijd over- en onderschattingen worden bestudeerd. Bij nadere inspectie bleek deze variabele wel een effect te hebben indien alleen overschattingen of alleen onderschattingen werden geanalyseerd. In het geval van het ontbreken van financiële rapportages was er vaker sprake van onderschattingen dan indien ze wel aanwezig waren. Als er wel financiële rapportages waren, dan waren er juist meer overschattingen. Blijkbaar veranderen projecten bij het invoeren van financiële rapportages van risicozoekende naar risicomijdende projecten.


IT Risks in Measure and Number


135


138


IT Risks in Measure and Number


