Science of Computer Programming 72 (2008) 136–175

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

# Quantifying requirements volatility effects

G.P. Kulk*, C. Verhoef

*VU University Amsterdam, Department of Computer Science, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands*

ARTICLE INFO

*Article history:*
Received 31 August 2007
Received in revised form 29 April 2008
Accepted 30 April 2008
Available online 13 May 2008

*Keywords:*
Requirements volatility
IT portfolio management
Quantitative IT portfolio management
Volatility benchmark
IT dashboard
Requirements metric
Requirements creep
Scope creep
Requirements scrap
Requirements churn
Compound monthly growth rate
Volatility tolerance factor
$\pi$-ratio
$\rho$-ratio
Requirements volatility dashboard

ABSTRACT

In an organization operating in the bancassurance sector we identified a low-risk IT subportfolio of 84 IT projects comprising together 16,500 function points, each project varying in size and duration, for which we were able to quantify its requirements volatility. This representative portfolio stems from a much larger portfolio of IT projects. We calculated the volatility from the function point countings that were available to us. These figures were aggregated into a requirements volatility benchmark. We found that maximum requirements volatility rates depend on size and duration, which refutes currently known industrial averages. For instance, a monthly growth rate of 5% is considered a critical failure factor, but in our low-risk portfolio we found more than 21% of successful projects with a volatility larger than 5%. We proposed a mathematical model taking size and duration into account that provides a maximum healthy volatility rate that is more in line with the reality of low-risk IT portfolios. Based on the model, we proposed a tolerance factor expressing the maximal volatility tolerance for a project or portfolio. For a low-risk portfolio its empirically found tolerance is apparently acceptable, and values exceeding this tolerance are used to trigger IT decision makers. We derived two volatility ratios from this model, the $\pi$-ratio and the $\rho$-ratio. These ratios express how close the volatility of a project has approached the danger zone when requirements volatility reaches a critical failure rate. The volatility data of a governmental IT portfolio were juxtaposed to our bancassurance benchmark, immediately exposing a problematic project, which was corroborated by its actual failure. When function points are less common, e.g. in the embedded industry, we used daily source code size measures and illustrated how to govern the volatility of a software product line of a hardware manufacturer. With the three real-world portfolios we illustrated that our results serve the purpose of an early warning system for projects that are bound to fail due to excessive volatility. Moreover, we developed essential requirements volatility metrics that belong on an IT governance dashboard and presented such a volatility dashboard.



> The creation of software requirements is reminiscent of hiking in a fog that is gradually lifting.
> –T. Capers Jones [39]

## 1. Introduction

Software is ubiquitous, 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

* Corresponding author. Tel.: +31 205987782.
*E-mail addresses:* erald@few.vu.nl (G.P. Kulk), x@cs.vu.nl (C. Verhoef).

0167-6423/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2008.04.003

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 [5], 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: [12,21,29,44,59,60,65,81,83]. Weinberg succinctly states the rationale for requirements [81]:

> Requirements are made for a common purpose: to change vague desires into explicit and unambiguous statements of what the customers want.

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 [81] 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 [35] 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 [55], 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 solving problems. In this paper 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) [9,14,23]. The project concerns modernization of the Air Traffic Control system to:

> meet projected increases in traffic volumes, enhance the system's 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 paper, 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?

Software is a knowledge product, therefore, during the development process either progressive understanding of the product is gained or new technology comes available that needs to be incorporated into the product. Because of the volatile nature of the environment that software needs to abide by, initial requirements are not necessarily the right ones and need to be adapted and bent in the right direction during the process. However, requirements arriving in later project phases seem more difficult to implement than requirements added in earlier stages of the process. We consider requirements volatility healthy if it delivers an end-product based on changed requirements due to progressive understanding of the product and when the end-product based on the changed requirements better suits the customer's needs than an end-product based on the initial, partly or completely, incorrect requirements. The requirements volatility must be within healthy bounds to enable the project to deliver a product at all. Moreover, the volatility must be supported by a sound business case. In all other cases, we are facing intolerable volatility, i.e. when we have either too much volatility or it lacks a sound business case, or both, which is usually the standard. With the metrics proposed in this paper, it is possible to monitor the volatility and create a requirements volatility dashboard as an early-warning system to monitor for projects that are out of control.

*Requirements reality.* We recall Jones who already wrote in 1996 [35] 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 [44, 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 [47] that between 41% [63] and 56% [72] of all defects can be traced back to errors made during the requirements phase. Robert Glass traced in a small empirical study [22] 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* [8]. 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 paper 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 language, and therefore very suitable for cost estimation, comparisons, benchmarking [1–3,13,20] 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 [20]. Function points can be used consistently and with an acceptable degree of accuracy [40,41,75]. There are different techniques known for conducting function point countings [32,52]. The International Function Point Users Group provides an ISO standard for function point analysis [28]. 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. When 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[1] sector we identified a low-risk IT subportfolio for which we were able to quantify its requirements volatility. An IT portfolio 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

[1] Bancassurance: an, originally French, portmanteau of *banking* and *assurance* [84].

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 our paper is a real-world known to be low-risk portfolio. The results presented in this paper 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.

### 1.1. Overview

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

*Taxonomy.* We start with a taxonomy of requirements volatility in Section 2.

*Basics and related work.* In Section 3 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 4. 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 5, 6 and 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 6 analyzes the volatility of high-risk government projects and compares these projects with our bancassurance benchmark created in Section 5. Section 9 discusses the volatility of a software product line.

*In-depth studies.* More in-depth studies of requirements volatility can be found in Sections 7 and 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 8 we perform a root-cause analysis on the bancassurance portfolio to clarify differences in volatility.

*Lines of code.* Section 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 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. 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 [81,80], 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 paper. 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 paper.

*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 [9] 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 amount of requirements for the running project. The amount of requirements decrease: 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 a 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 [9,15]. All DSR installations were eventually completed in 2000 [17].

*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.

## 3. Modeling volatility as compound interest

Jones [35] 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 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 [56,57,76]. 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 the paper 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 in 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 [62] with $r$ being the periodical interest rate that is accredited every interval and $t$ denotes the number of intervals the *StartAmount* stays within the bank, which is usually denoted in years.

$$FutureAmount = StartAmount \cdot (1 + r)^t. \tag{1}$$

By applying some standard algebraic manipulations the aforementioned Formula (1) can be rewritten to the following equivalent Formula (2).

$$r = \sqrt[t]{\frac{FutureAmount}{StartAmount}} - 1. \tag{2}$$

Instead of calculating the future value of a certain amount as can be done with Formula (1), the required interest rate is now calculated. Formula (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) that an annual interest rate of approximately 4.14% would be needed to achieve our goal.

### 3.1. Compound requirements

The CAGR originating from accounting transposes effortlessly into information technology on the subject of requirements engineering, as introduced by Jones [35]. 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 (3) shows the requirements equivalent of Formula (1) and in Formula (4) we show the equivalent of Formula (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.

$$SizeAtEnd = SizeAtStart \cdot \left(1 + \frac{r}{100}\right)^{t} \tag{3}$$

$$r = \left(\sqrt[t]{\frac{SizeAtEnd}{SizeAtStart}} - 1\right) \cdot 100. \tag{4}$$

*Jones' 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 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 paper 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 paper 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 amount 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 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 alarming requirements volatility values. 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.

### 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 (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). Later on in this paper 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 [7, page 40], it is therefore important to know at what moment in a project the requirements have been added. Jones [39] 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.

### 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 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 [75]. In brief, boxplots and Kolmogorov-Smirnov tests were used this 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 [75].

*DSDM.* A method to manage requirements is the MoSCoW principle from the DSDM method [67,66] 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.

### 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 [4,78,43,50,6,19, 42,79,30,82,43,46,58,70,25,24]. However, these 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 [11,61,24]. In a paper by Zowghi and Nurmuliani [85] 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 [49,48] 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.

**Table 1**
Jones' industry averages and maximum rates in various industries

| Software type | Avg $r$ (%) | max $r$ (%) | Out of control (%) |
|---|---|---|---|
| Contract or outsourced software | 1.1 | 3.4 | >5 |
| MIS software | 1.2 | 5.1 | >5 |
| Systems software | 2.0 | 4.6 | >5 |
| Military software | 2.0 | 4.5 | >15 |
| Commercial software | 2.5 | 6 | – |
| Civilian government software | 2.5 | – | – |
| Web-based software | 12 | – | – |

Other papers [31,51,71] 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 our paper 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. [27] 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 paper 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 [27], 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 paper. Stark et al. [69,68] 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. [53]. 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 [4] 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 for military software was done by Henry and Henry [26]. 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 [64] 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 [35,36,39,38] 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 1. Besides the averages of requirements change, Jones states encountered maximum volatility rates in [37, Table 7.4, Table 8.5, Table 9.4, Table 10.3, Table 11.4], summarized in the third column of Table 1 and in [37, 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 1. In Table 1 MIS refers to Management Information Systems. The accompanying failure factor is the event that the requirements creep is out of control.

The figures in Table 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 [39] also an average creep rate of 12% for agile projects. However, Table 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.

## 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 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 10,000 function points. It turns out that twice the original requirements end up – often undocumented – in the final

**Table 2**
Various total requirements increases for a volatility of 2% per month

| Year | Requirements increase (%) |
|---|---|
| 1 | 27 |
| 2 | 61 |
| 3 | 104 |

**Table 3**
Requirements doubling

| Volatility rate (%) | Month when requirements have doubled |
|---|---|
| 2 | 36 |
| 3 | 24 |
| 5 | 15 |
| 10 | 8 |
| 20 | 4 |

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 10,000 function point project this results, when using Formula (3), in $10{,}000 \cdot 2.04 = 20{,}400$ 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 [35] in which he mentions projects with overall changes of 100 and even 270% growth.

Table 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.

Projects having a larger compound monthly growth rate will reach the point of doubled requirements sooner. See Table 3 for some typical examples. From Tables 2 and 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 (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 $e$.

$$t = \frac{\log 2}{\log (1 + r/100)}. \tag{5}$$

A simplification of Function (5) is often referred to as the 72-rule [54, 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 (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 $r$.

$$t = \frac{\log 2}{r}. \tag{6}$$

In Table 3 some values of doubling requirements are shown as an example. The numbers in Table 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 1, but without mentioning 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. Fig. 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 Fig. 1 have a starting value of a hypothetical single function point project. For a project with a different starting value the growth curves in Fig. 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 3 with vertical dot-dash lines.

An interesting characteristic about the growth function $b^t$ of the requirements or workload of a project in Fig. 1 is the point for which the derivative equals 1, or equally, when the time-elasticity of volatility equals 1. Since, before this point

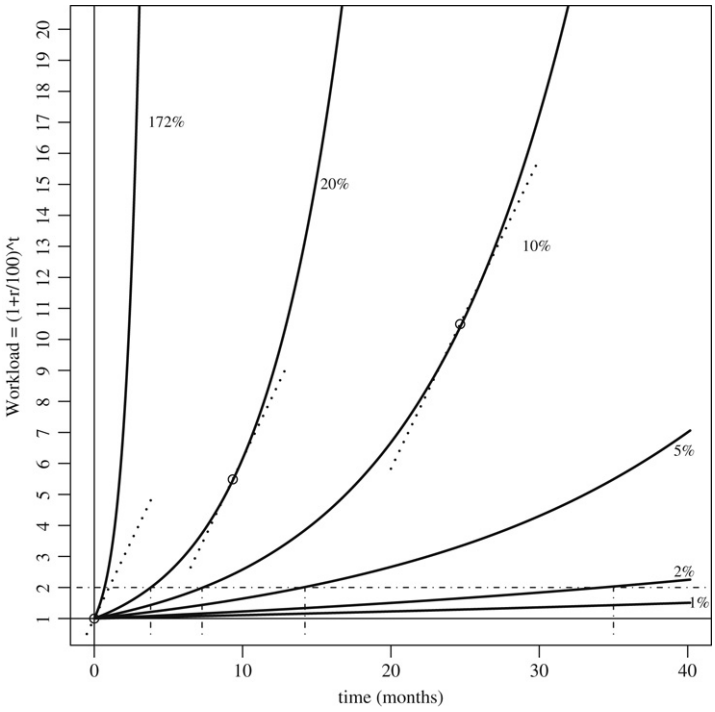


**Fig. 1.** Requirements growth.

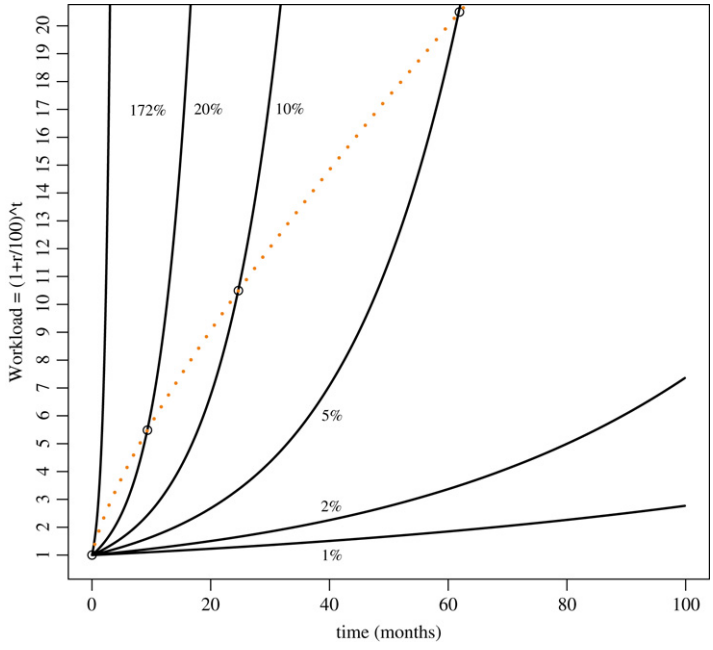


**Fig. 2.** Connecting the tangents of proportional growth.

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 $b^t$ equaling 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:

$$f'(t) = \log(b) \cdot b^t. \tag{7}$$

We need to solve the equation $f'(t) = 1$ to find the point for which the function $b^t$ intersects a tangent $f(t) = t + c$ with a certain constant $c$ dependent on the rate $r$. In Fig. 2 we have redrawn Fig. 1 a little zoomed out, but now with the line

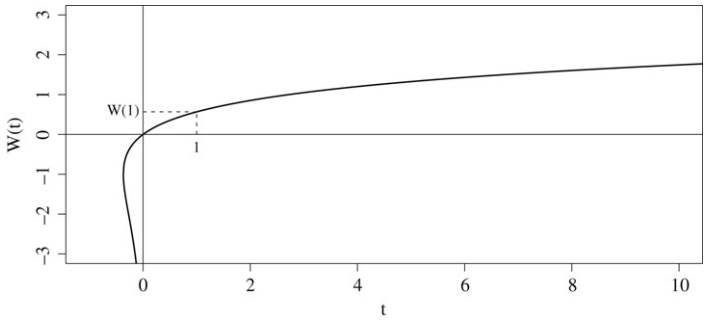


**Fig. 3.** Lambert's $W$ function.

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 every month than were counted at the beginning of the project. So, the line in Fig. 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 Eq. (7) is Formula (8) for a certain multiplier $b$ and is found as follows:

$$\begin{aligned}
&\log(b) \cdot b^t = 1 && \xrightarrow{\text{divide by } \log b}\\
&b^t = \frac{1}{\log b} && \xrightarrow{\text{logarithm of both sides if } b^t > 0}\\
&\log(b^t) = \log\left(\frac{1}{\log(b)}\right) && \xrightarrow{\text{rewrite left- and right-handside}}\\
&t \cdot \log(b) = \log(\log(b)^{-1}) && \xrightarrow{\text{divide left- and right-handside by } \log b}\\
&t = \frac{-\log(\log(b))}{\log(b)} \qquad b^t > 0. && (8)
\end{aligned}$$

### 4.1. Lambert's W function

The solution presented in Eq. (8) can be used to calculate a maximum project duration in months for a given rate $r$. Recall that $b = 1 + \frac{r}{100}$. Fig. 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 (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 (8) is Function (9) when solving for $b$ and Function (10) when solving for $r$. Function (9) results in a multiplier factor $b$, with $b = 1 + \frac{r}{100}$ and $r$ being the compound monthly volatility rate. In Fig. 4 we will draw Function (10). Function (10), resulting in a multiplier $b$ for a project with duration $t$, can be found solving Eq. (8) using Lambert's $W$ function [10,16,45]. Later on we will show how we arrived at this solution.

$$b = e^{\frac{W(t)}{t}} \qquad b^t > 0, b = 1 + (r/100) \tag{9}$$

or, equivalently for rate $r$

$$r_{\max} = (e^{\frac{W(t)}{t}} - 1) \cdot 100 \quad r, t > 0 \tag{10}$$

Lambert's $W$, or Omega function, used in Eq. (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 Eq. (11), the value of $W(1)$.

$$\Omega \cdot e^{\Omega} = 1 \Longrightarrow \Omega \approx 0.567143290409. \tag{11}$$

The Lambert $W$ function is multi-valued in the interval $[-\frac{1}{e}, 0]$ as can be seen in Fig. 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 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 Fig. 3 for the interval $[-\frac{1}{e}, 10]$.

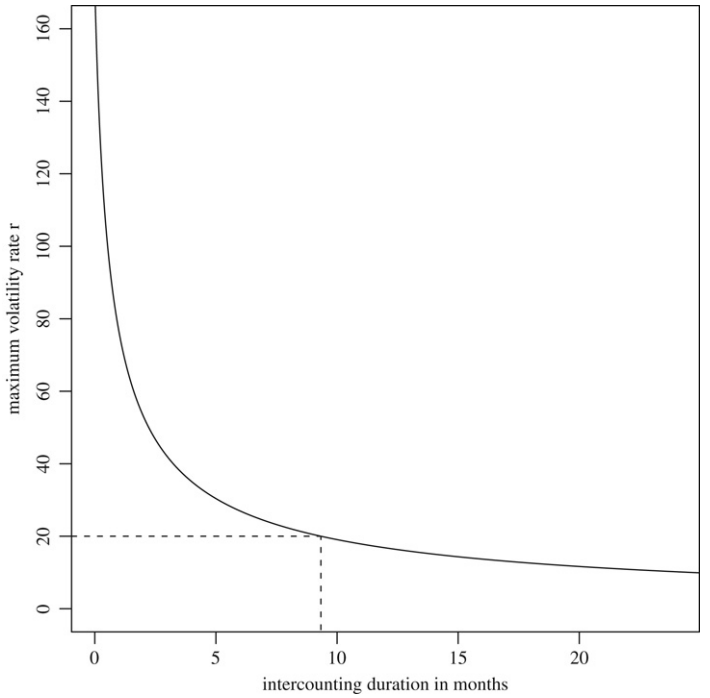


**Fig. 4.** Eq. (10): Maximum monthly requirements growth $r$ for duration $t$ and $p = 1$.

*Solving with Lambert.* This paragraph shows how the $W$ function has been used in solving Eq. (7). Since every function of the form $Y = X \cdot e^X$ can be rewritten to $X = W(Y)$ [10], we will rewrite Eq. (7) in that format:

$$\begin{aligned}
&\log(b) \cdot b^t = 1 && \xrightarrow{\text{if } b^t > 0} \\
&\log(b) \cdot e^{\log(b^t)} = 1 && \xrightarrow{a \cdot \log b = \log b^a} \\
&\log(b) \cdot e^{t \cdot \log(b)} = 1 && \xrightarrow{\text{multiply by } t} \\
&t \cdot \log b \cdot e^{t \cdot \log b} = t && \xrightarrow{X = t \cdot \log b \text{ and } Y = t \text{ and } Y = X \cdot e^X \Rightarrow X = W(Y)} \\
&t \cdot \log b = W(t) && \xrightarrow{\text{divide by } t} \\
&\log b = \frac{W(t)}{t} && \xrightarrow{e^{\text{left- and right-hand side}}} \\
&b = e^{\frac{W(t)}{t}}. &&
\end{aligned} \tag{12}$$

The limitation we have created by solving for variable $b$ is $b^t > 0$. But, since we have only positive durations, $t$ will always be positive and since $b$ expresses the multiplication factor $1 + (r/100)$, this variable that is always larger than 0. A negative $b$ has no physical meaning in our model, since that would imply a project with a negative amount of function points. Therefore, $b^t$ will always be larger than zero in our model. With Eq. (12) we now have a function to calculate the maximum allowable volatility rate $r$, with $b = 1 + (r/100)$ when we have a project duration $t$ available.

*Example of volatility approaching the danger zone.* The resulting volatility function in Eq. (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 (10) can be illustrated with the following example. Let's consider two projects. The first is a two month project initially estimated at a 100 function points and the second a 20 month project initially estimated at 1000 function points. Eq. (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 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. When we observe that both projects underwent a 15% monthly requirements growth, for the second project this is very close to the dictated maximum, then, the first project has 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' industrial averages are inadequate as an early warning system to detect unhealthy volatility.

### 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 (9) also the point for which the derivative equals 0.5 or 2. To create a function similar to Eq. (9) we will need to solve a new equation, see Eq. (13), in which $p$, replacing 1 in Eq. (9) indicates the slope of the tangent of the growth function, $t$ the duration between size estimates and $b$ the multiplicative factor defined by $b = 1 + (r/100)$.

$$p = p(b, t) = \log b \cdot b^t. \tag{13}$$

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 Fig. 4. The actual value of the tolerance factor $p$ for a specific completed project can be calculated with Eq. (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 Eq. (14).

$$P = \max(p_i). \tag{14}$$

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 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 (13) for $b$, $t$ or $r$ results in the Eqs. (15)–(17) with $b$, $t$ and $r$ defined as before. In fact, these equations are similar to the previous Eqs. (9) and (10) with tolerance factor $p$ added to them. When $p$ equals 1 we end up with the previously described formulas.

$$t = \frac{\log p - \log(\log b)}{\log b} \tag{15}$$

$$b = b(p, t) = \mathrm{e}^{\frac{W(p \cdot t)}{t}} \qquad b^t > 0, b = 1 + (r/100) \tag{16}$$

$$r_\pi = r_\pi(p, t) = (\mathrm{e}^{\frac{W(p \cdot t)}{t}} - 1) \cdot 100 \qquad r, t > 0. \tag{17}$$

In Fig. 5 a plot is shown of Eq. (17) for $p = 0.25, 0.5, 1$ and 2. The intercounting duration in Fig. 5 is the duration between two size estimates. Fig. 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.

### 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 projects' actual requirements volatility rate $r_{\mathrm{act}}$ by its maximal healthy volatility rate calculated by Eq. (10), or, when a tolerance factor different from 1 is being used Eq. (17). The calculation of this volatility ratio is illustrated in Fig. 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.

*$\pi$-ratio.* The $p$-proportional volatility ratio $\pi$, presented in Eq. (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 $r_{\mathrm{act}}$. 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

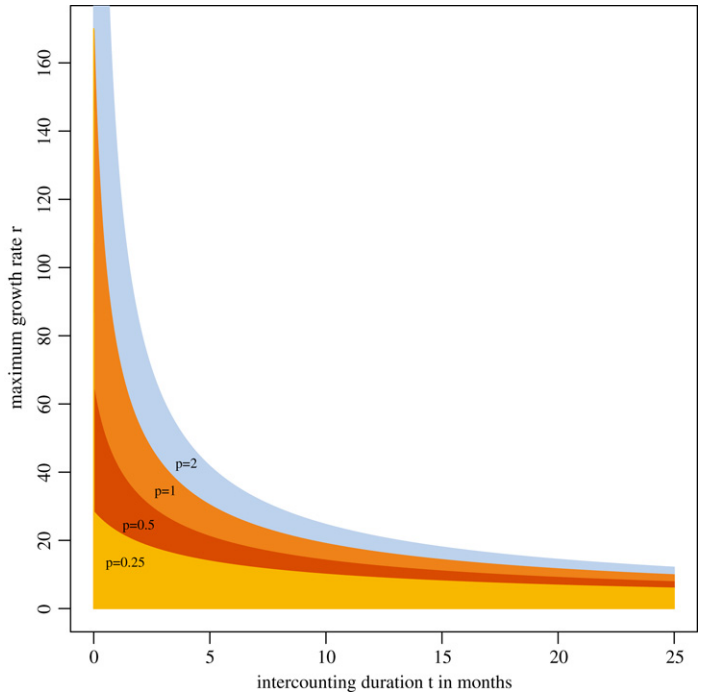


**Fig. 5.** Maximum growth factor $r_\pi$ for intercounting duration $t$ and different tolerance factors $p$.

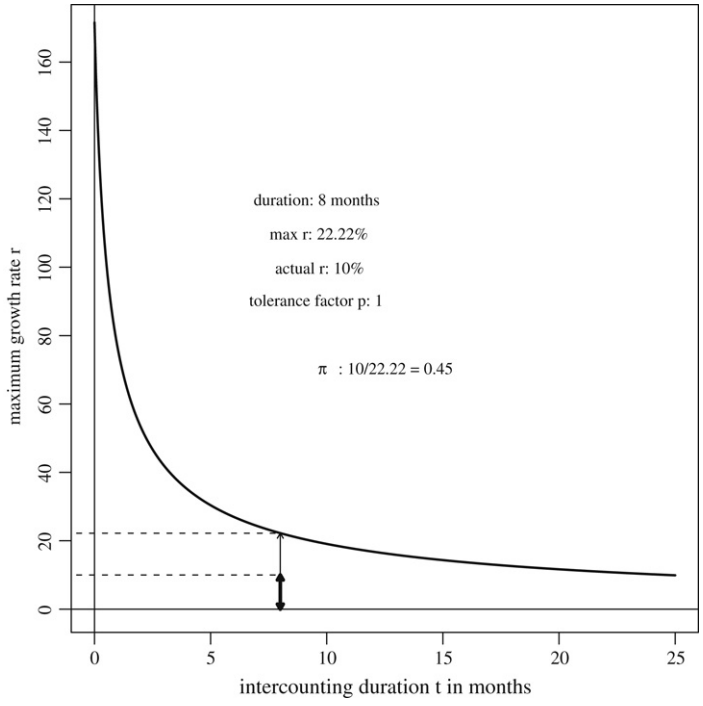


**Fig. 6.** Calculating the $p$-proportional volatility ratio $\pi_1$.

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.

$$\pi_p = \pi_p(r_{\text{act}}, t, p) = \frac{r_{\text{act}}}{r_\pi} = \frac{r_{\text{act}}}{(\mathrm{e}^{\frac{W(p\cdot t)}{t}} - 1)\cdot 100}. \tag{18}$$

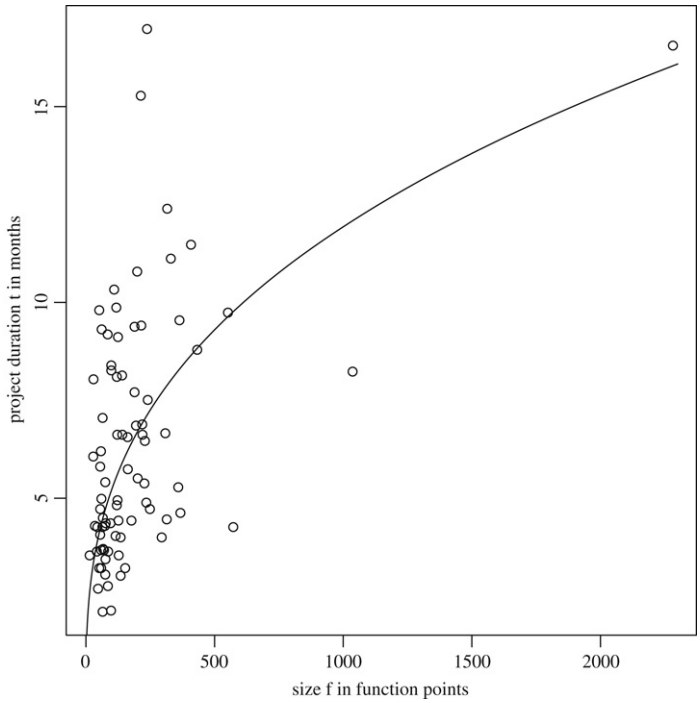


**Fig. 7.** Project size versus project duration combined with estimated function.

### 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 [36, p. 202] a benchmark is presented for the relationship between duration $d$ and size $f$ of IT projects. In Eq. (19) this relationship is presented with the formula's exponent 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 is 0.359, which is similar, but a little bit smaller than the values presented in [36].

$$f^{0.359} = d. \tag{19}$$

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 Fig. 7 is not a result of chance alone. In Fig. 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 can be seen from Fig. 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.

In Eq. (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. 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 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.

$$r_\rho = (\mathrm{e}^{\frac{W\left(\frac{p \cdot t}{\log f}\right)}{t}} - 1) \cdot 100 \qquad (1 + r/100)^t > 0, f > 1. \tag{20}$$

We illustrate the maximum requirements volatility presented in Eq. (20) with the three-dimensional Fig. 8. Fig. 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 $t$.

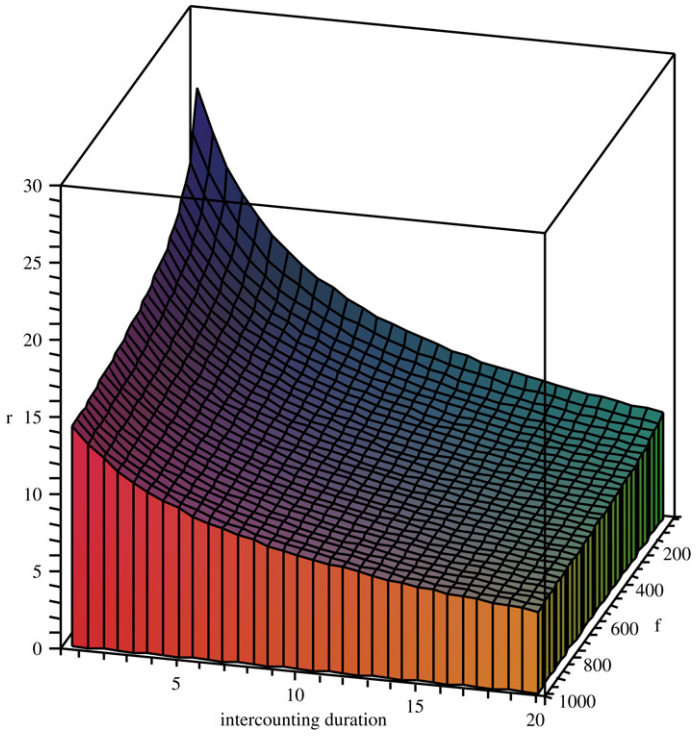


**Fig. 8.** $r_\rho$ for $p = 1$ and intercounting duration $t$ and size $f$.

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 $r_{\text{act}}$, the size of the project $f$, the intercounting duration $t$ and a tolerance factor $p$. This new ratio is defined in Eq. (21) and is used to calculate a duration and size invariant volatility metric.

$$\rho_p = \rho_p(r_{\text{act}}, t, p, f) = \frac{r_{\text{act}}}{r_\rho} = \frac{r_{\text{act}}}{\left(\mathrm{e}^{\frac{W\left(\frac{p \cdot t}{\log f}\right)}{t}} - 1\right) \cdot 100}. \tag{21}$$

*Minimum requirements volatility.* For completeness' sake we state that the lower bound in Fig. 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 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.

## 5. Case study: Measuring requirements volatility

As was observed in [75] 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

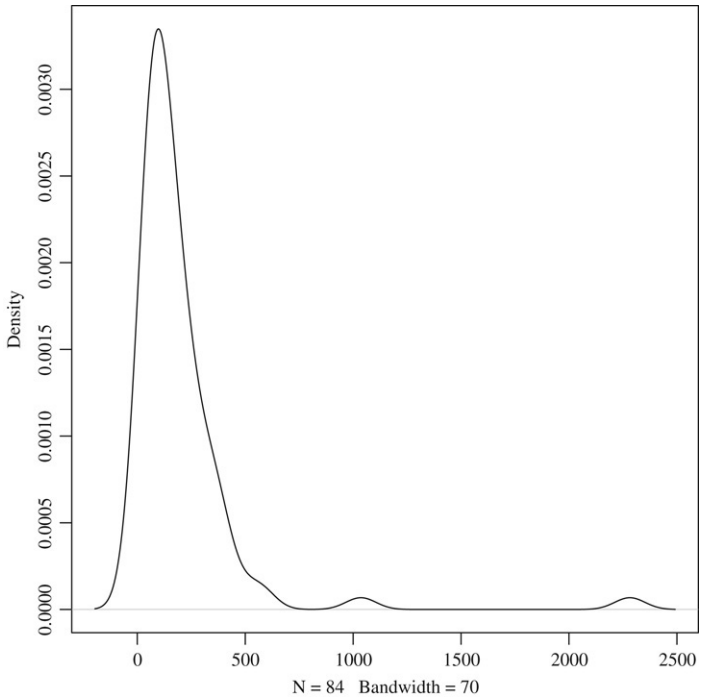


**Fig. 9.** Project size in function points.

**Table 4**
Function point countings characteristics

| Min. | First quartile | Median | Mean | Third quartile | Max. |
|---|---|---|---|---|---|
| 14.0 | 66.8 | 123.5 | 197.1 | 227.5 | 2282.0 |

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 paper are a result from considering the characteristics of the volatility of several portfolios of projects instead of the averages presented in Table 1. All project data from different organizations that are the basis for this paper 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 [75]. 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' failure factors, see Table 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 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. Fig. 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 [75].

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 [75]. 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,

**Table 5**
Compound monthly volatility rate $r$ characteristics for the bancassurance portfolio

| Difference | Min. | First quartile | Median | Mean | Third quartile | Max. |
|---|---|---|---|---|---|---|
| Relative $r$ | −23.660 | −3.553 | 0.000 | 0.903 | 3.708 | 31.500 |
| Absolute $r$ | 0.000 | 1.398 | 3.681 | 5.797 | 6.272 | 31.500 |

an overall growth of about 6%. For some projects after the functional design an extra size estimate of the requirements was done. In Section 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 5.

From Table 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' averages in Table 1. The last row of Table 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 an average of 1.2% requirements volatility and the 20 outsourced projects with an average of 1.1% requirements volatility, this 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' industrial averages. This volatility based on averages is only a 0.27% percent point off our own measured volatility average of 0.903%. We infer from the coincidence of Jones' latest averages with the averages of our detailed information that in itself it is an indication that it is useful to use Jones' 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 paper or use our benchmarks as a surrogate, rather than an average of a presumably strongly asymetric data set.

Jones has measured maximums that are displayed in Table 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 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 size 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 $\pi$ 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 Fig. 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.

### 5.1. Volatility density function

Next, we will analyze the volatility indicator's probability density function and its outliers for the bancassurance portfolio. Fig. 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 [73] and are used to visually depict the five-number summaries as illustrated in Table 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 [75] already hinted, that the probability is leptokurtic and heterogeneous

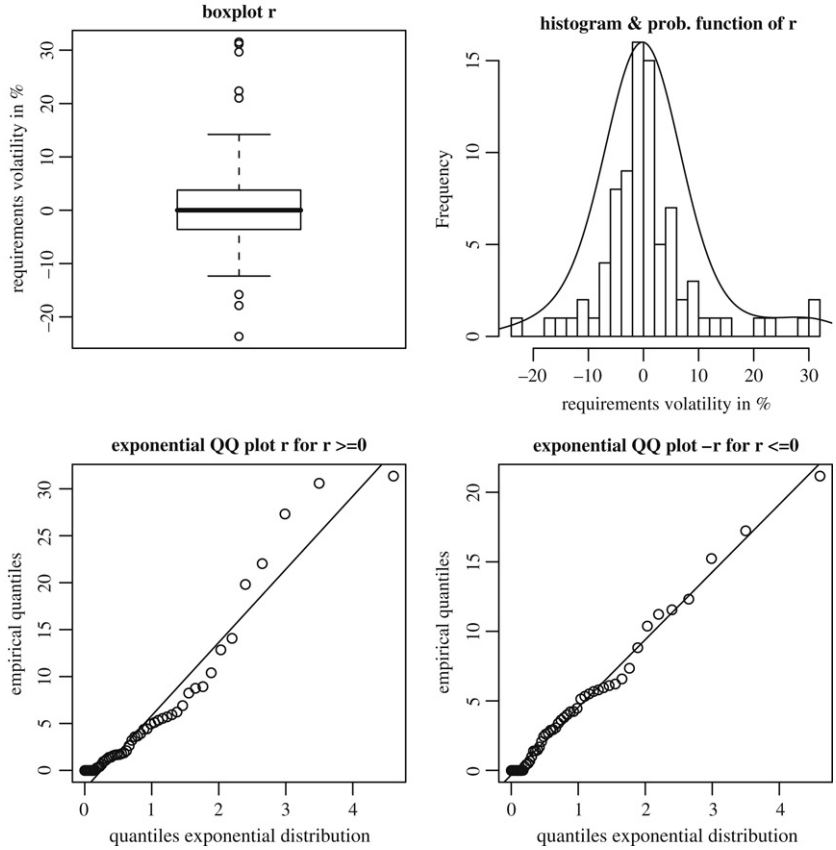


**Fig. 10.** Volatility characteristics.

with heavy tails on both sides. To further illustrate this assumption we provide also Quantile–Quantile plots, or short Q–Q plots in Fig. 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 Fig. 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.

### 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 (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 (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

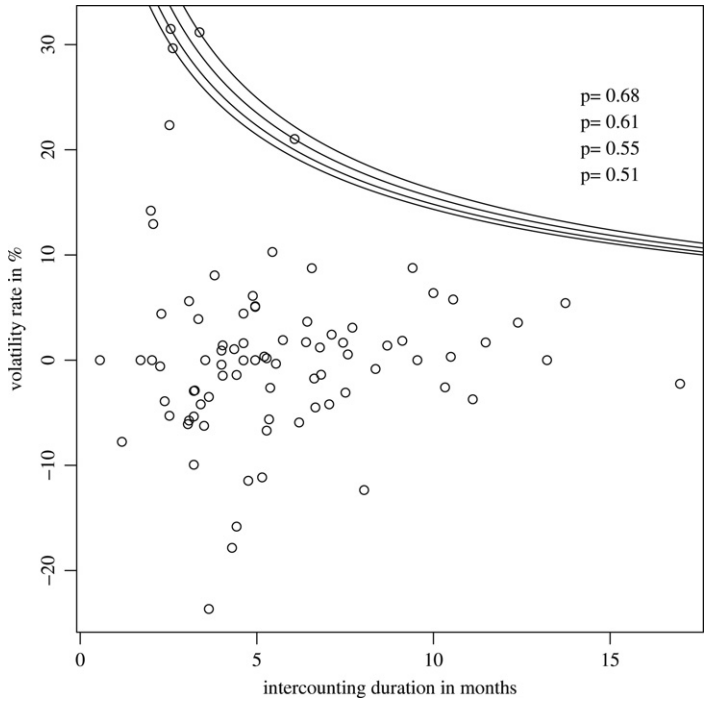


**Fig. 11.** Various tolerance factors for the bancassurance portfolio.

**Table 6**
Kurtosis and skewness of requirements volatility for the bancassurance portfolio

| | |
|---|---|
| Kurtosis | 3.266893 |
| Skewness | 1.088345 |

volatility using Eqs. (22) and (23) are presented in Table 6.

$$Kurtosis = \frac{m_4}{m_2^2} - 3 = \frac{n\sum_{i=1}^{n}(x_i - \bar{x})^4}{(\sum_{i=1}^{n}(x_i - \bar{x})^2)^2} - 3 \tag{22}$$

$$Skewness = \frac{m_3}{m_2^{3/2}} = \frac{\sqrt{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{(\sum_{i=1}^{n}(x_i - \bar{x})^2)^{3/2}}. \tag{23}$$

Table 6 confirms with a kurtosis of 3.267 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.

### 5.3. Actual tolerance factors

For each project's tolerance factor $p$ a function as drawn in Fig. 4 can be made. In Fig. 11 we have drawn a picture with the same axes as Fig. 4 for the bancassurance portfolio. In Fig. 11 we can also see the tolerance factors $p$ for four projects: 0.51, 0.55, 0.61 and the highest tolerance factor 0.68. These factors are, when placing them as a tolerance factor $p$ in Function (17), a point in the maximum volatility function (17), as is shown in Fig. 11 for these four projects.

In Table 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 Eq. (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

**Table 7**
Tolerance factor $p$ characteristics for the bancassurance portfolio

| | Min. | First quartile | Median | Mean | Third quartile | Max. |
|---|---|---|---|---|---|---|
| $p$ | −0.101 | −0.0281 | 0.000 | 0.0397 | 0.0457 | 0.678 |

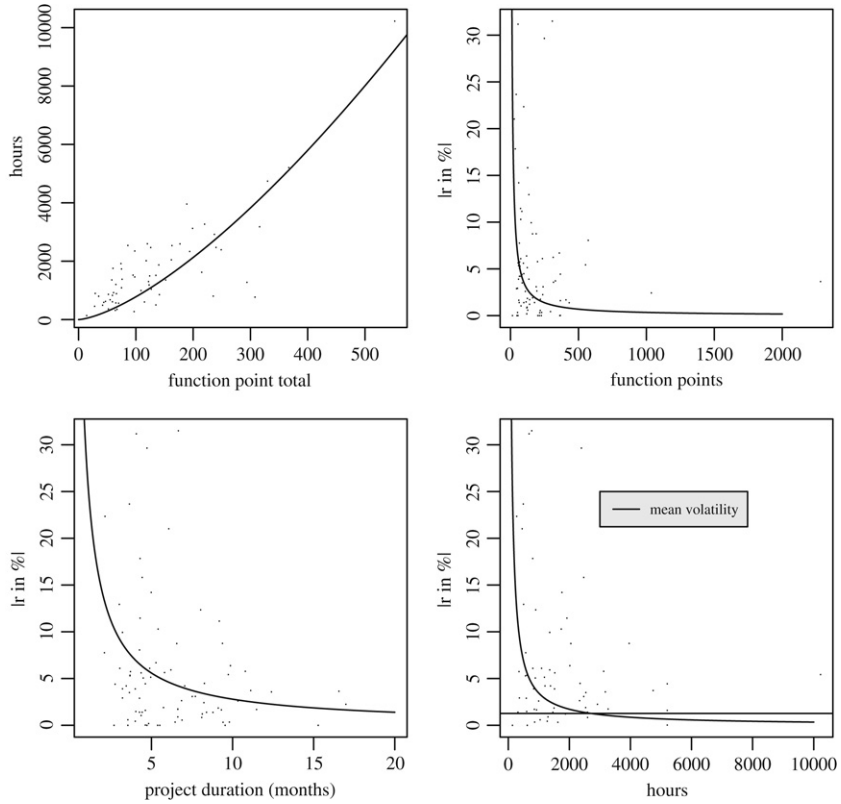


**Fig. 12.** Bancassurance portfolio characteristics.

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 Fig. 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 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. 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 $r$. We know that we are dealing with a low-risk portfolio and we see high volatilities occurring at small projects. Indeed, a further qualitative analysis 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 6 in which we analyze the volatility of a high-risk portfolio.

### *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.

Fig. 13 juxtaposes different boxplots of the project portfolio to show the differences between twice and thrice counted projects. The first two boxplots in Fig. 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

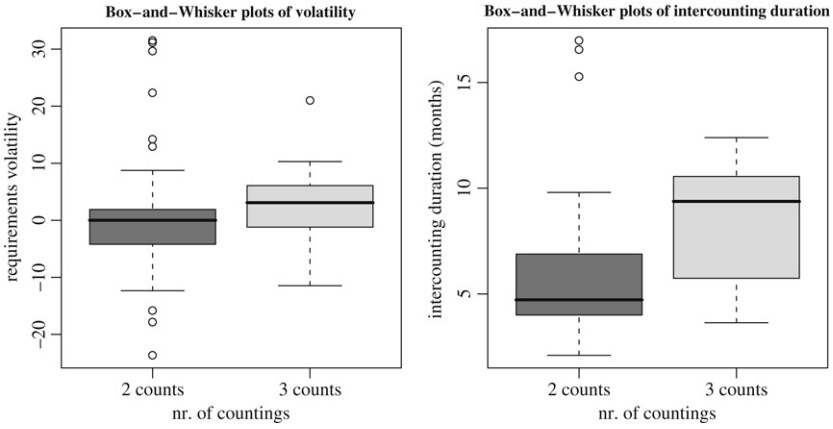


**Fig. 13.** Differences between twice or thrice counted bancassurance projects.

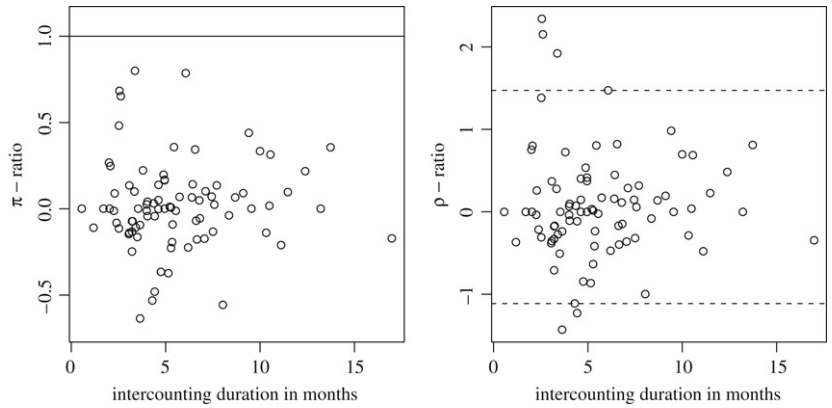


**Fig. 14.** Comparing volatility ratios $\pi_1$ and $\rho_1$ for the bancassurance portfolio.

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 is confirmed by the boxplots in the second plot in Fig. 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.

### 5.5. Inspecting the volatility ratios

Now we turn back to our volatility metric $\pi$ from Eq. (18). We have calculated the $\pi_1$-ratio from Eq. (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 Fig. 14. The same exercise can be done for our $\rho$-ratio metric presented in Eq. (21) which also takes the project size into account. This results in the second picture in Fig. 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. Volatility risks 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 8 we have listed a statistical summary of the volatility ratios $\pi_1$ and $\rho_1$.

In Fig. 14 the value of the tolerance factor $p$ for both ratios is 1. The scattering of $\pi$ and $\rho$ in figures like Fig. 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 Fig. 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

**Table 8**
Volatility ratios $\pi_1$ and $\rho_1$ for the bancassurance portfolio

| Ratio | Min. | First quartile | Median | Mean | Third quartile | Max. |
|---|---|---|---|---|---|---|
| $\pi_1$ | −0.636 | −0.111 | 0 | 0.0254 | 0.136 | 0.799 |
| $\rho_1$ | −1.429 | −0.292 | 0 | 0.0831 | 0.329 | 2.339 |

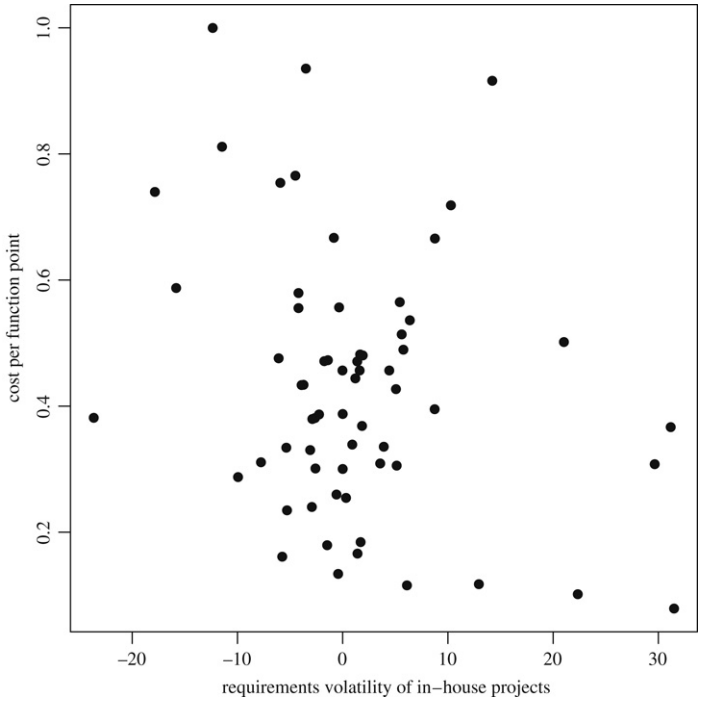


**Fig. 15.** Cost per function point versus requirements volatility for the bancassurance portfolio.

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 Fig. 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 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 dangerously high volatility, this metric certainly belongs in IT dashboards for IT governors. In Section 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.

### 5.6. Cost per function point

In Fig. 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, but 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 Fig. 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.

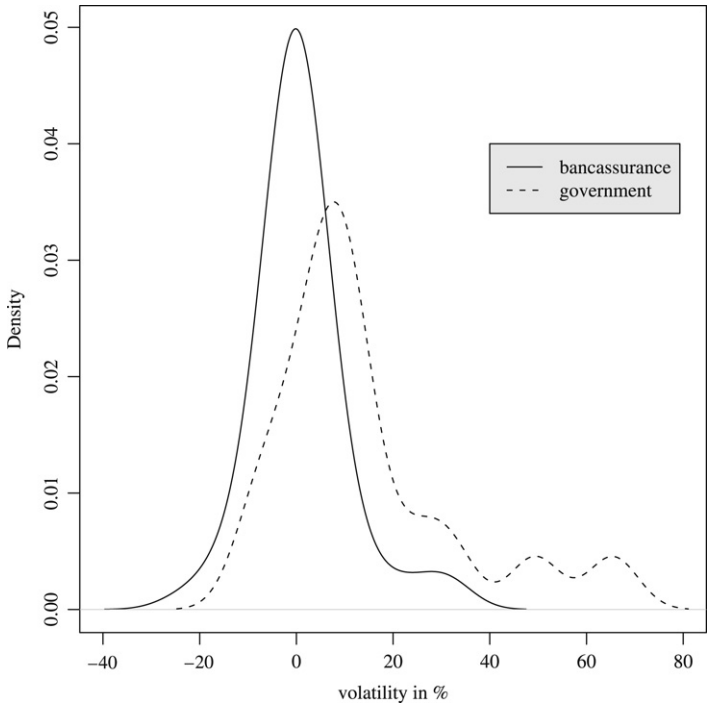


**Fig. 16.** Applying our bancassurance benchmark on governmental projects.

## 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 $r$ we created a probability density function. This function and our bancassurance benchmark are plotted in Fig. 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 $D = 0.5087$ with a $p$-statistic of 0.0002416 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 Fig. 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 Fig. 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 Fig. 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 Fig. 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 Fig. 14 indicating that this subproject was well into the danger zone already after its intermediate size estimate.

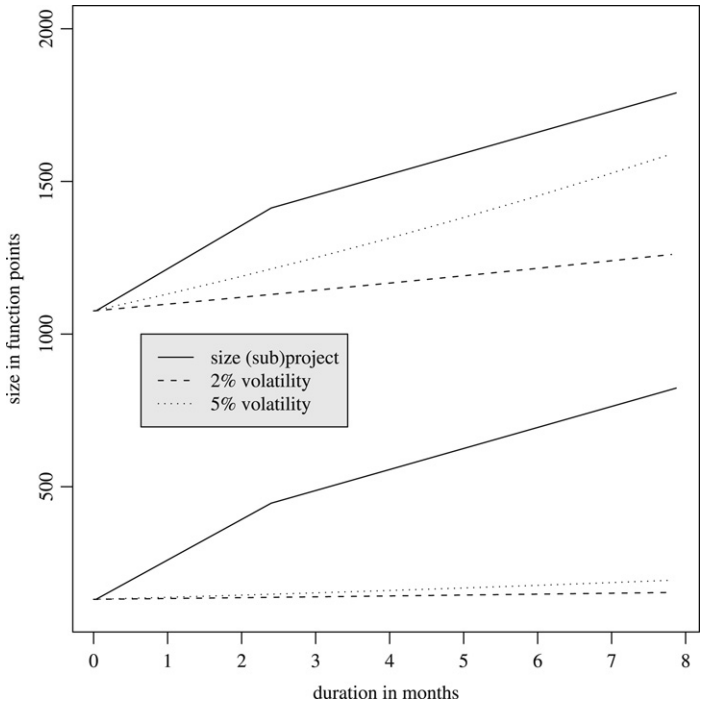


**Fig. 17.** Requirements growth over time.

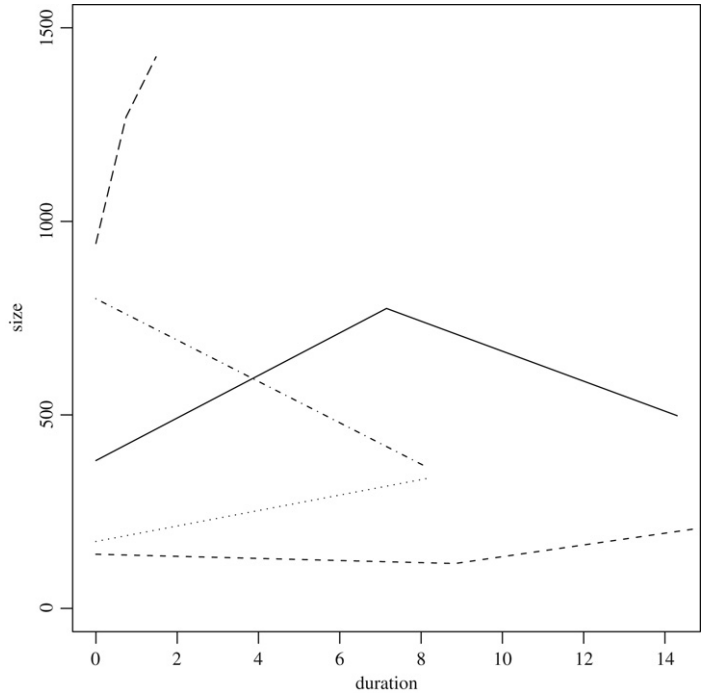


**Fig. 18.** Requirements growth over time.

These governmental volatilities are plotted alongside our benchmark data in Fig. 19 as solid dots. We remind the reader that in this plot we plotted $\pi_{0.68}$ and $\rho_{0.68}$, whereas in Fig. 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 Fig. 19 represent the bancassurance benchmark data. In Fig. 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

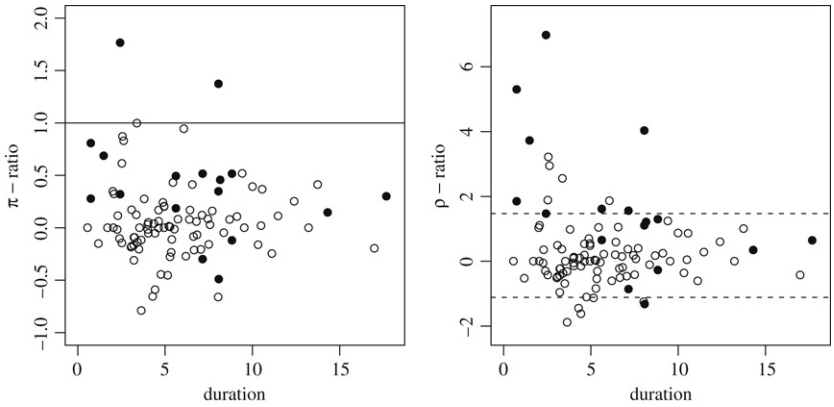


**Fig. 19.** Volatility ratios $\pi_{0.68}$ and $\rho_{0.68}$ for governmental projects versus the bancassurance portfolio.

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 Fig. 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 Fig. 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 [34] 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 paper 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.

## 7. Volatility variations for outsourcing

Many organizations outsource their IT function or parts of it. As Jones mentions in his book [39], 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 [55] 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 9**
Size variations for in-house and outsourced projects from the bancassurance portfolio

| Projecttype | Min. | First quartile | Median | Mean | Third quartile | Max. |
|---|---|---|---|---|---|---|
| In-house | 14 | 64.75 | 116.5 | 141.2 | 190.2 | 552 |
| Outsourced | 55 | 111.2 | 223 | 376 | 374.2 | 2282 |

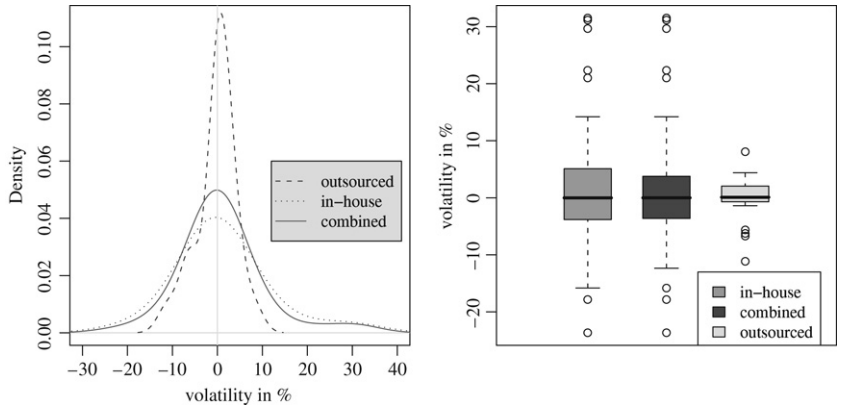


**Fig. 20.** Volatility variations for in-house versus outsourcing.

**Table 10**
Kurtosis and Skewness of volatility of in-house versus outsourced projects

| | In-house | Outsourced | Combined |
|---|---|---|---|
| Kurtosis | 2.174717 | 0.6720551 | 3.266893 |
| Skewness | 0.964278 | −0.7271956 | 1.088345 |

In Table 9 we show the size characteristics for the in-house versus outsourced projects. We see in Table 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 Fig. 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 Fig. 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 Fig. 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 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 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 Fig. 20 are the smaller sized projects. In this portfolio, the outsourced projects are larger in function point size than the in-house developed projects.

### 7.1. In-house or outsourced?

As Table 9 states, the mean for in-house projects is 141 function points and for outsourced projects 360 function points. The plots in Fig. 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 1 and his findings on differences in average sizes of in-house and outsourced projects [37, 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

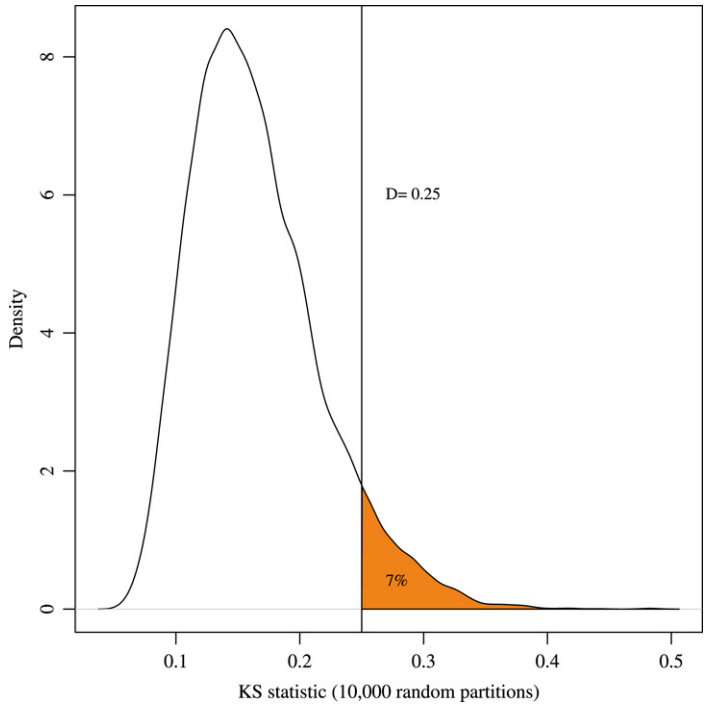


**Fig. 21.** Positioning in-house versus outsourced partitions among random partitions.

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' 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 Fig. 21 with the test statistic for the in-house versus outsourced partition plotted as a vertical line. The percentage of randomly 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 differences 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 current large distance, although not statistically significant, can barely be induced by sheer coincidence.

## 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. 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 function 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. Function (24) states for the bancassurance portfolio the statistically fitted productivity function *fppm*, a simple statistical fit which is a variation of the benchmark in [74, p. 61, Formula 46]. Function (24) benchmarks for a certain size $f$, expressed in function points, its productivity expressed in function points per staff month. Eq. (24) is plotted in Fig. 22.

$$fppm = \frac{677.6572}{f}. \tag{24}$$

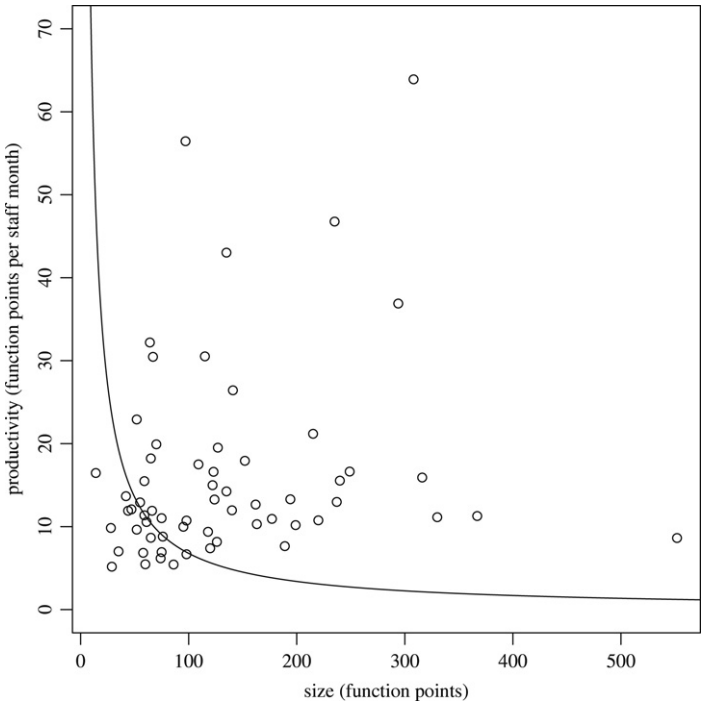


**Fig. 22.** Project hours versus productivity.

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 function points. This is not in accordance with reality, 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 Fig. 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 increasing requirements creep, but were not able to manage the requirements change, resulting in a low productivity. In the upper halve 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 Fig. 23.

### *8.1. Size and volatility*

To assure that the differences in volatility in Figs. 23 and 25 did not stem from size differences in the different projects Fig. 24 has been made. In Fig. 24 the size in function points is added to the plot, in which larger sizes imply a circle with a larger diameter. Fig. 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, turned out to explain the boomerang shape.

In Fig. 25 we present the boomerang picture again, but now for only three sub-portfolios each containing 7–13 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' type 5 enhancements [39]. 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 high cohesion between IT and business in this part of the bancassurance portfolio. Because of cohesion, small changes to the requirements were often made after requirements sign-off, resulting in high volatility rates, which is not desirable given the legacy portfolio.

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

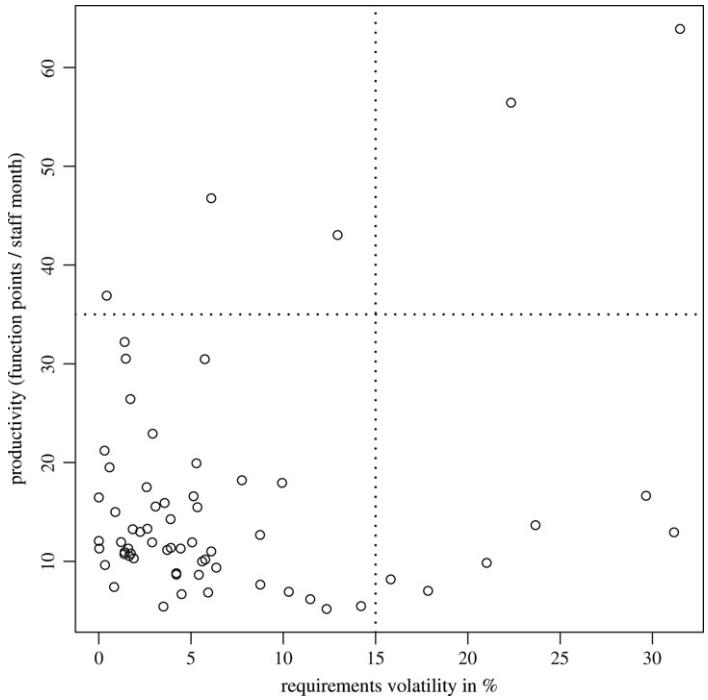


**Fig. 23.** Requirements volatility and function points per staff month.

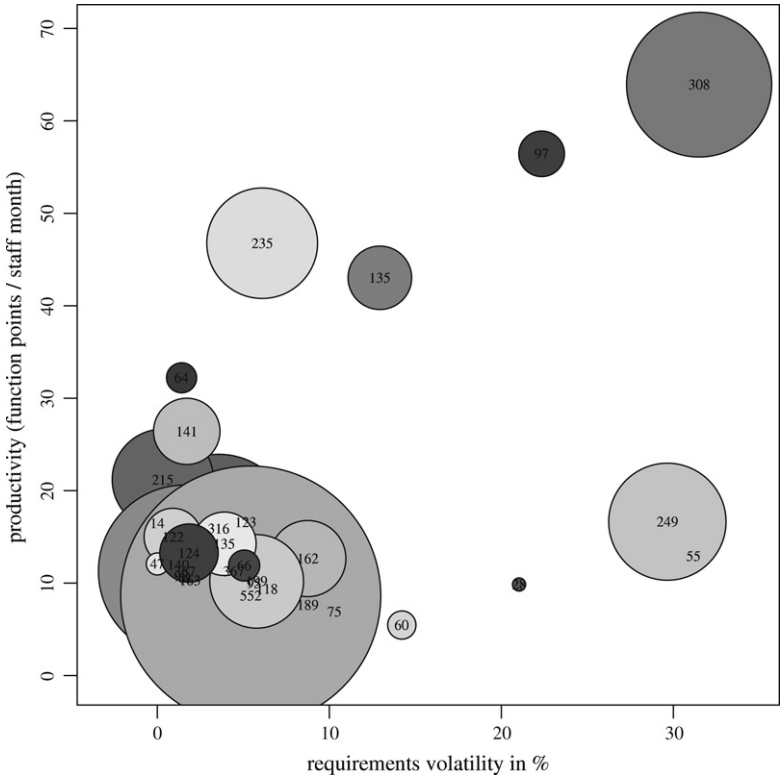


**Fig. 24.** Volatility variations versus the productivity of projects and size in function points.

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 dealing with volatility. The last group, partition *z*, does not show a high requirements change,

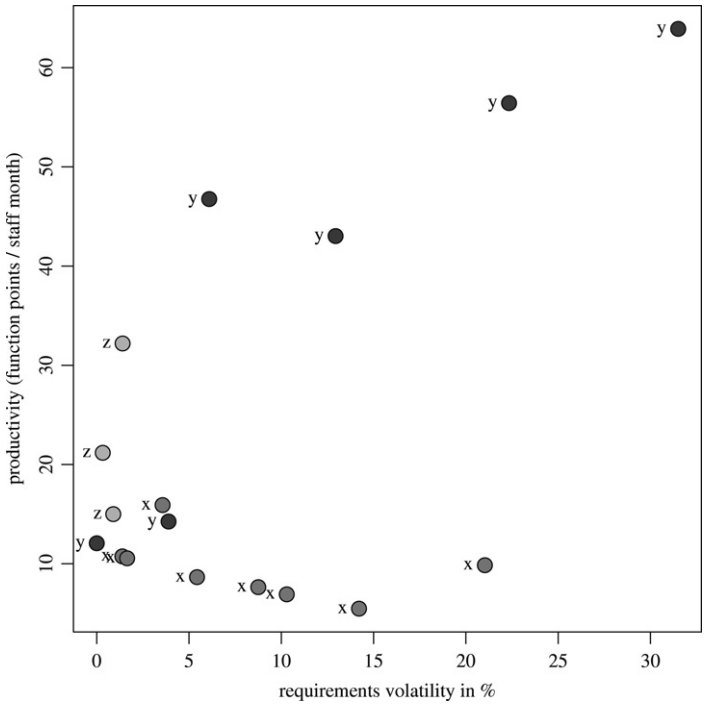


**Fig. 25.** Volatility and function points per staff month for different partitions of the bancassurance portfolio.

**Table 11**
Volatility quadrant division

| | Low volatility | High volatility |
|---|---|---|
| High productivity | $r < 15\%\, fppm > 35$ | $r > 15\%\, fppm > 35$ |
| Low productivity | $r < 15\%\, fppm < 35$ | $r > 15\%\, fppm > 35$ |

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 Fig. 23 in four quadrants, creating a diagram with four different types of projects. In Table 11 we listed the values corresponding to the quadrant division. The matrix in Table 11 corresponds with the quadrants in Fig. 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.

## 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 quick 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 [33,39,37], 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 [37, page 78] Jones lists function point conversion rates for various programming languages.

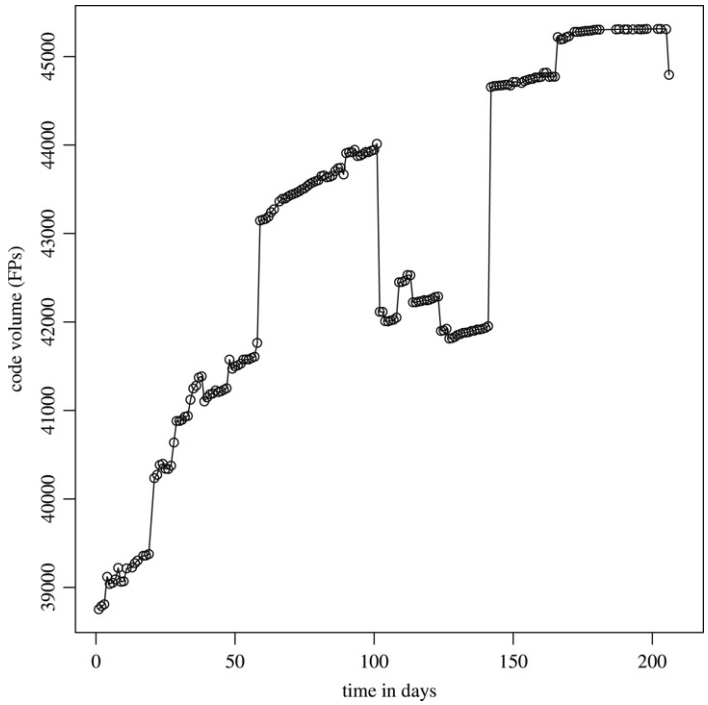


**Fig. 26.** Software product line code volume over time.

### 9.1. Volatility of a software product line

In Fig. 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 Fig. 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 Fig. 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 (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 (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 (25) using the *SizeAtDay* on day $n$ and day $n + 1$.

$$r = \left( \left( \frac{SizeAtDay_{n+1}}{SizeAtDay_n} \right)^{30.5} - 1 \right) \cdot 100. \tag{25}$$

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 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,

**Table 12**
Software product line requirements volatility characteristics

| $r$ | Min. (%) | First quartile (%) | Median (%) | Mean (%) | Third quartile (%) | Max. (%) |
|---|---|---|---|---|---|---|
| All | −73.97 | 0.04 | 0.70 | 4.92 | 1.70 | 571.70 |
| Cleaned | −29.57 | 0.05 | 0.69 | 1.69 | 1.63 | 39.21 |

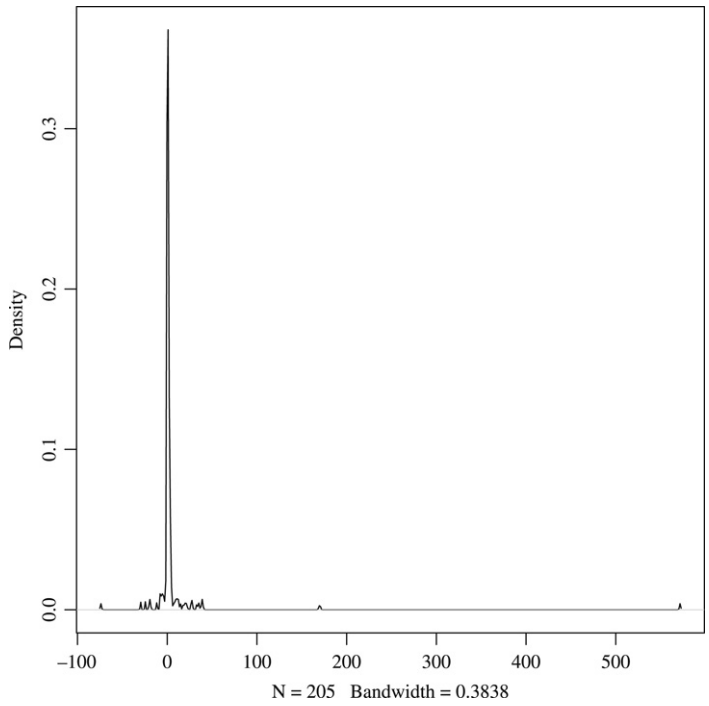


**Fig. 27.** Probability density plot for the software product line.

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' average for system software, 2%, is again close to our empirically found average.

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 [18], 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' 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' 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 Fig. 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 Fig. 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. Fig. 28 results in an extended benchmark for the expected probability density of the volatility for different professional environments:

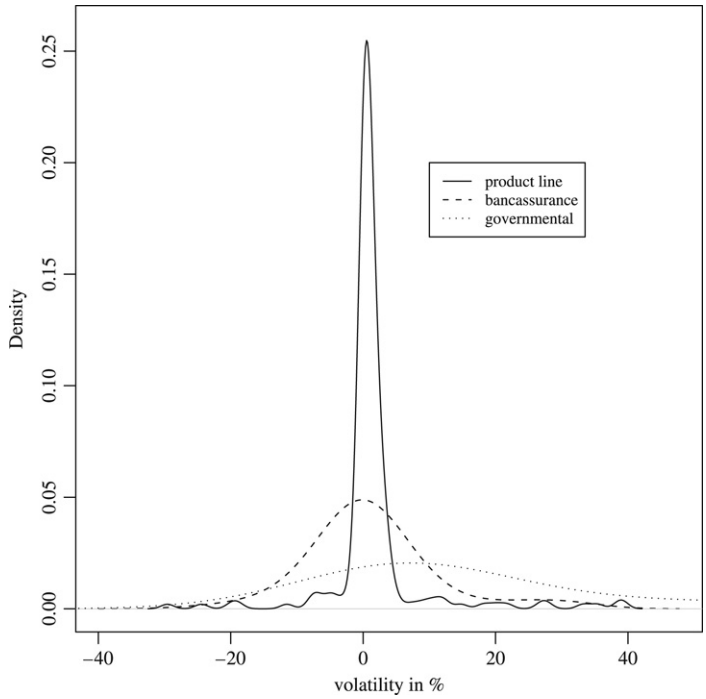


**Fig. 28.** Requirements volatility rate density plots for various industries.

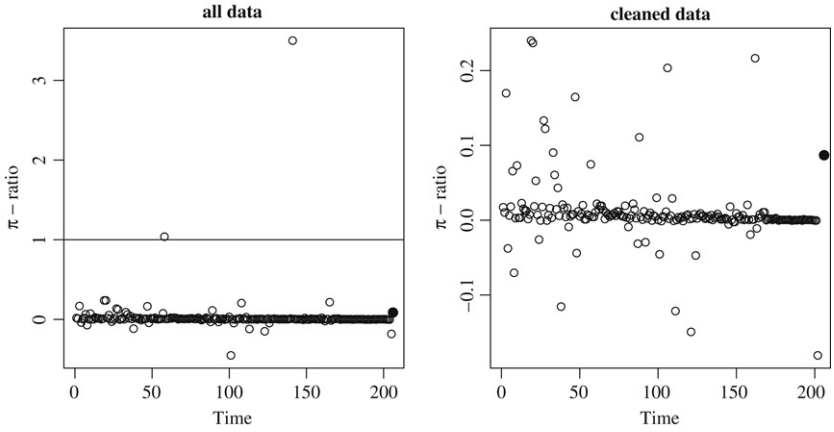


**Fig. 29.** The $\pi_1$-ratio for a systems software portfolio.

- 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.

### 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 Fig. 29 we present the $p$-proportional volatility ratios $\pi$, calculated with Formula (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 Fig. 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

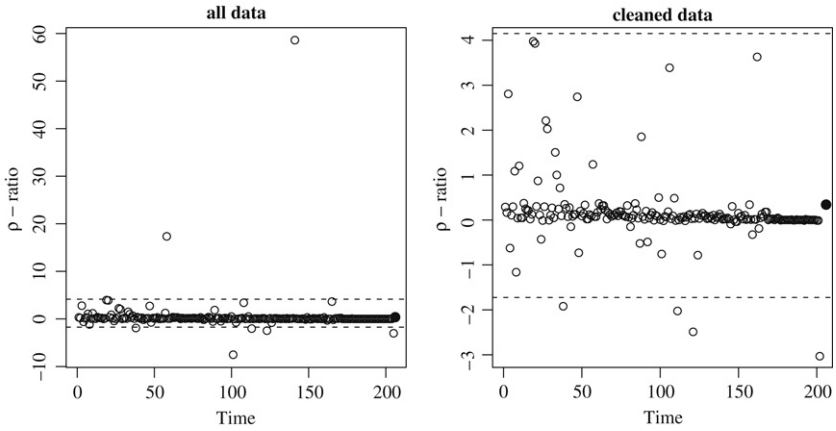


**Fig. 30.** The $\rho_1$-ratio for a systems software portfolio.

**Table 13**
Example of a requirements volatility dashboard

| *project* | $r$ (%) | $p$ | $r_\pi$ (%) | $\pi_{0.68}$ | $r_\rho$ (%) | $\rho_{0.68}$ |
|---|---|---|---|---|---|---|
| Project X | 6.52 | 0.105 | 18.68 | 0.349 | 5.89 | 1.107 |
| Project Y | 25.63 | 1.433 | 18.68 | 1.372 | 6.36 | 4.030 |
| ... | ... | ... | ... | ... | ... | ... |

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 Fig. 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 Fig. 30. These plots contain the overall $\rho$-ratios from begin to end as solid dots. On the left-hand side of Fig. 30 we plotted the $\rho$-ratios for the daily volatilities expressed in monthly rates for all data. On the right-hand side of Fig. 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 Fig. 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.

## 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 [77]. 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 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 Fig. 11 from the low-risk bancassurance portfolio. Besides the four volatility metrics we have also shown in Table 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.

Project X in Table 13 is the project in Fig. 17 discussed in Section 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 Fig. 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 4, project X has a volatility rate $r$ of $((1790/1076)^{(1/8.055)} - 1) \cdot 100 = 6.52\%$ and the subproject an $r$ of $((823/131)^{(1/8.055)} - 1) \cdot 100 = 25.63\%$. 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.

**Table 14**
Lambert $W$ function quick reference for intercounting duration $t$ in months

| $t$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $W(t)$ | 0.567 | 0.852 | 1.050 | 1.202 | 1.327 | 1.432 |
| $t$ | 7 | 8 | 9 | 10 | 12 | 14 |
| $W(t)$ | 1.524 | 1.606 | 1.679 | 1.746 | 1.863 | 1.964 |
| $t$ | 16 | 18 | 20 | 22 | 24 | 26 |
| $W(t)$ | 2.053 | 2.133 | 2.205 | 2.271 | 2.332 | 2.388 |

The values in Table 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 $W(t)$.

$$\pi_{0.68} = \frac{r_{\text{act}}}{(e^{\frac{W(0.68 \cdot t)}{t}} - 1) \cdot 100} \tag{26}$$

$$\rho_{0.68} = \frac{r_{\text{act}}}{\left(e^{\frac{W\left(\frac{0.68 \cdot t}{\log f}\right)}{t}} - 1\right) \cdot 100}. \tag{27}$$

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 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 Eq. (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 Eq. (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 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 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 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 quantile, 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 13 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.

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. Fig. 31 shows an example of the dashboard plots for project Y and restates our conclusions of Section 6. These dashboard plots of requirements volatility combined with the dashboard in Table 13 are insightful to compare a project with its peers on all four volatility levels and is a management tool for monitoring requirements volatility.

### *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 15 combines all summaries, of which many were already provided in earlier sections.

## 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 paper 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.

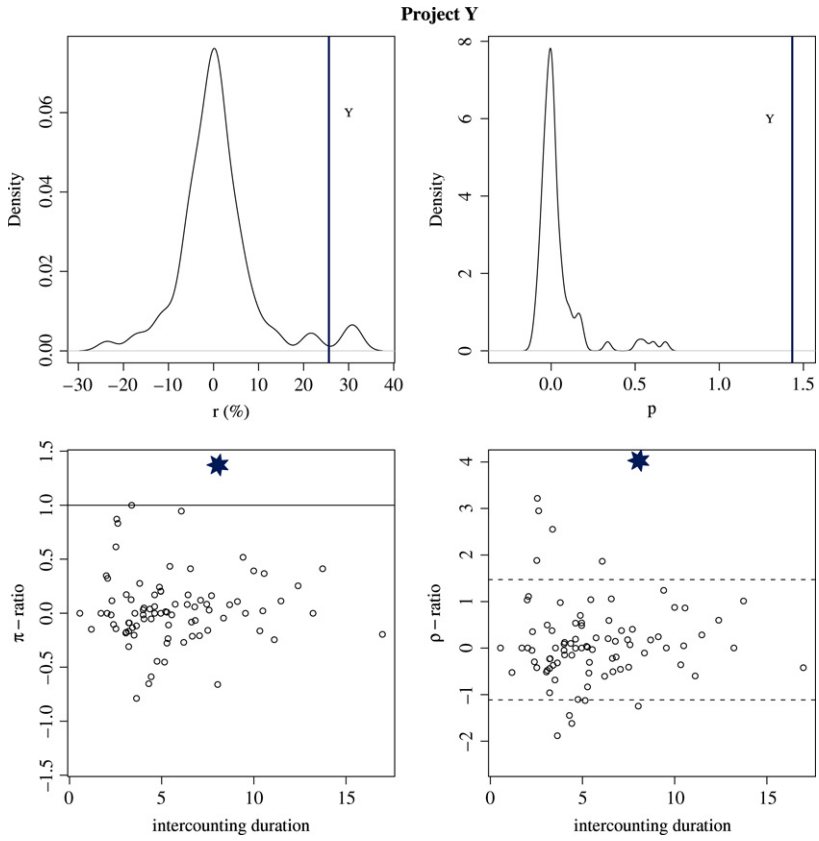


**Fig. 31.** Graphical representation of a volatility dashboard for $r$, $p$, $\pi_{0.68}$ and $\rho_{0.68}$.

**Table 15**
Statistical summary of the analyzed data

| | Min. | First quartile | Median | Mean | Third quartile | Max. |
|---|---|---|---|---|---|---|
| | Bancassurance | | | | | |
| $r$ (%) | −23.660 | −3.553 | 0.000 | 0.903 | 3.708 | 31.500 |
| $p$ | −0.101 | −0.0281 | 0.000 | 0.0397 | 0.0457 | 0.678 |
| $\pi_1$ | −0.636 | −0.111 | 0.000 | 0.0254 | 0.136 | 0.799 |
| $\rho_1$ | −1.429 | −0.292 | 0.000 | 0.0831 | 0.329 | 2.339 |
| | Government | | | | | |
| $r$ (%) | −9.143 | 3.816 | 8.475 | 14.730 | 18.740 | 65.460 |
| $p$ | −0.0442 | 0.0560 | 0.147 | 0.388 | 0.311 | 1.714 |
| $\pi_{0.68}$ | −0.490 | 0.184 | 0.349 | 0.428 | 0.517 | 1.764 |
| $\rho_{0.68}$ | −1.320 | 0.644 | 1.296 | 1.726 | 1.849 | 6.981 |
| $\pi_1$ | −0.414 | 0.153 | 0.295 | 0.341 | 0.437 | 1.379 |
| $\rho_1$ | −1.029 | 0.493 | 1.020 | 1.258 | 1.269 | 5.051 |
| | Systems software, cleaned data set | | | | | |
| $r$ (%) | −29.57 | 0.05 | 0.69 | 1.69 | 1.63 | 39.21 |
| $p$ | −0.35 | 0.00048 | 0.0069 | 0.014 | 0.016 | 0.33 |
| $\pi_1$ | −0.181 | 0.00029 | 0.0042 | 0.0103 | 0.00996 | 0.24 |
| $\rho_1$ | −3.031 | 0.0049 | 0.071 | 0.172 | 0.165 | 3.97 |

- 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 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. Our paper 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 paper 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 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.

## Acknowledgments

This research received partial support by the Dutch *Joint Academic and Commercial Quality Research & Development (Jacquard)* program on Software Engineering Research via contract 638.004.405 *Equity: Exploring Quantifiable Information Technology Yields* and via contract 638.003.611 *Symbiosis: Synergy of managing business-IT-alignment, IT-sourcing and offshoring success in society*. We like to thank a number of organizations that will remain anonymous for discussions on their requirements volatility. Furthermore, we would like to thank the the anonymous reviewers for their encouraging words and very good suggestions.

## References

[1] D.A. Adamo, S. Fabrizi, M.G. Vergati, A light functional dimension estimation model for software maintenance, in: E. Chikofsky, R. Kazman, C. Verhoef, (Eds.) Proceedings of the first IEEE International Conference on Exploring Quantifiable Information Technology Yields, EQUITY 2007, 2007.

[2] A.J. Albrecht, Measuring application development productivity, in: Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, 1979, pp. 83—92.
[3] A.J. Albrecht, J.E. Gaffney, Software function, source lines of code, and development effort prediction: A software science validation, IEEE Transactions on Software Engineering 9 (9) (1983) 639–648.
[4] G. Anthes, No more creeps! are you a victim of creeping user requirements? Computerworld 28 (18) (1994) 107–110.
[5] E. Arranga, I Archbell, J. Bradley, P. Coker, R. Langer, C. Townsend, M. Weathley, In cobol's defense, IEEE Software 17 (2000) 70–72,75.
[6] C. Barry, M. Lang, A comparison of traditional and multimedia information systems development practices, Information and Software Technology 45 (2003) 217–227.
[7] B. Boehm, Software Engineering Economics, Prentice Hall, 1981.
[8] F.P. Brooks Jr., The Mythical Man-Month – Essays on Software Engineering, anniversary edition, Addison-Wesley, 1995.
[9] Robert N. Charette, Large-scale project management *is* risk management, IEEE Software 13 (4) (1996) 110–117.
[10] R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, D.E. Knuth, On the lambert W function, Advances in Computational Mathematics 5 (1996) 329–359.
[11] Rita J. Costello, Dar-Biau Liu, Metrics for requirements engineering, Journal of Systems Software 29 (1) (1995) 39–63.
[12] Alan M. Davis, Software Requirements: Objects Functions, and States, Prentice Hall, 1993.
[13] J.B. Dreger, Function Point Analysis, Prentice Hall, 1989.
[14] Gerald L. Dillingham, et al. Report to congressional requesters: NATIONAL AIRSPACE SYSTEM — FAA has made progress but continues to face challenges in acquiring major air traffic control systems, Technical Report, United States Government Accountability Office, US GAO, GAO-05-331, 2005.
[15] Mark O. Hatfield, et al. AIR TRAFFIC CONTROL — status of FAA's modernization program, Technical Report, United States General Accounting Office, US GAO, GAO/RCED-95-175FS, 1995.
[16] L. Euler, De serie Lambertina plurimisque eius insignibus proprietatibus, Acta AcademiæScientarum Imperialis Petropolitinæ (1783) 29–51.
[17] FAA. Blueprint for NAS modernization 2002 update. Available via: www.faa.gov/nasarchitecture/Blueprint2002.htm..
[18] D. Faust, C. Verhoef, Software product line migration and deployment, Software–Practice and Experience 33 (2003) 933–955.
[19] Ryan George Frasera, Jocelyn Armarego, Kanagasingam Yogesan, The reengineering of a software system for glaucoma analysis, Computer Methods and Programs in Biomedicine 79 (2005) 97–109.
[20] D. Garmus, D. Herron, Function Point Analysis — Measurement Practices for Successful Software Projects, Addison-Wesley, 2001.
[21] Donald C. Gause, Gerald M. Weinberg, Exploring Requirements: Quality Before Design, Dorset House Publishing, 1989.
[22] Robert L. Glass, Persistent software errors, IEEE Transactions on Software Engineering 7 (2) (1981) 162–168.
[23] Robert L. Glass, Software Runaways, Prentice Hall, 1998.
[24] Theodore F. Hammer, Leonore L. Huffman, Linda H. Rosenberg, Doing requirements right the first time, CROSSTALK The Journal of Defense Software Engineering (1998) 20–25.
[25] S.D.P. Harker, K.D. Eason, J.E. Dobson, The change and evolution of requirements as a challenge to the practice of software engineering, in: Proceedings of the IEEE International Symposium on Requirements Engineering, IEEE Computer Society, 1993, pp. 266–272.
[26] Joel Henry, Sallie Henry, Quantitative assessment of the software maintenance process and requirements volatility, in: Proceedings of the 1993 ACM conference on Computer science, ACM Press, 1993, pp. 346–351.
[27] Dan X. Houston, Gerald T. Mackulak, James S. Collofello, Stochastic simulation of risk factor potential effects for software development risk management, Journal of Systems and Software 59 (2001) 247–257.
[28] International function point users group, Software engineering ifpug 4.1 unadjusted functional size measurement method counting practices manual, Technical Report ISO/IEC 20926:2003, International Standardization Organization. www.iso.org, November 2003.
[29] Micheal Jackson, Software Requirements And Specifications, Addison-Wesley, 1995.
[30] Pankaj Jalote, Aveejeet Palit, Priya Kurien, V.T. Peethamber, Timeboxing: A process model for iterative software development, Journal of Systems and Software 70 (2004) 117–127.
[31] Talha Javed, Manzil e Maqsoo, Qaiser S. Durrani, A study to investigate the impact of requirements instability on software defects, ACM Software Engineering Notes 29 (4) (2004) 1–7.
[32] D.R. Jeffery, G.C. Low, M. Barnes, A comparison of function point counting techniques, IEEE Transactions on Software Engineering 19 (5) (1993) 529–532.
[33] C. Jones, Applied Software Measurement: Assuring Productivity and Quality, second edition, McGraw-Hill, 1996.
[34] C. Jones, Patterns of Software Systems Failure and Success, International Thomsom Computer Press, 1996.
[35] C. Jones, Strategies for managing requirements creep, IEEE Computer 29 (1996) 92–94.
[36] C. Jones, The Year 2000 Software Problem — Quantifying the Costs and Assessing the Consequences, Addison-Wesley, 1998.
[37] C. Jones, Software Assessments, Benchmarks, and Best Practices, in: Information Technology Series, Addison-Wesley, 2000.
[38] C. Jones, Conflict and litigation between software clients and developers, 2001. Version 10 – April 13. Technical Note.
[39] C. Jones, Estimating Software Costs, second edition, McGraw-Hill, 2007.
[40] C.F. Kemerer, Reliability of function points measurement — a field experiment, Communications of the ACM 36 (2) (1993) 85–97.
[41] C.F. Kemerer, B.S. Porter, Improving the reliability of function point measurement: An empirical study, IEEE Transactions on Software Engineering SE-18 (11) (1992) 1011–1024.
[42] Don Kerr, Heidi Winklhofer, The effect of rapid rural industry changes on the development of a decision support system for dairy farmers in Australia, Computers and Electronics in Agriculture 50 (2006) 61–69.
[43] Petri Kettunen, Maarit Laanti, How to steer an embedded software project: Tactics for selecting the software process model, Information and Software Technology 47 (2005) 587–608.
[44] Gerald Kotonya, Ian Sommerville, Requirements Engineering: Processes and Techniques, Wiley, 1998.
[45] J.H. Lambert, Observationes variae in mathesin puram, Acta Helvetica Physico-Mathematico-Anatomico-Botanico-Medica III (1758) 128–168. Available via: www.few.vu.nl/~erald/lambert.pdf.
[46] Lucas Laymana, Laurie Williamsa, Daniela Damianb, Hynek Buresc, Essential communication practices for Extreme Programming in a global software development team, Information and Software Technology 48 (2006) 781–794. Special Issue Section: Distributed Software Development.
[47] Dean Leffingwell, Calculating the return on investment from more effective requirements management, American Programmer 10 (4) (1997) 13–16.
[48] Annabella Loconsole, Definition and validation of requirements management measures. Ph.D. Thesis, Umeå University, 2007.
[49] Annabella Loconsole, Jürgen Börstler, An industrial case study on requirements volatility measures, in: 12th Asia-Pacific Software Engineering Conference, APSEC'05, IEEE CS Press, 2005, pp. 249–256.
[50] Nancy M. Lorenzi, Robert T. Riley, Organizational ISSUES=change, International Journal of Medical Informatics 69 (2003) 197–203.
[51] Yashwant K. Malaiya, Jason Denton, Requirements volatility and defect density, in: Proceedings of the 10th International Symposium on Software Reliability Engineering, 1998, pp. 285–294.
[52] E. Mendes, N. Mosley, S. Counsell, Comparison of Web size measures for predicting Web design and authoring effort, IEE Proceedings Software 149 (3) (2002) 86–92.
[53] N. Nurmuliani, Didar Zowghi, Sue Fowell, Analysis of requirements volatility during software development life cycle, in: Proceedings of the 2004 Australian Software Engineering Conference, ASWEC04, IEEE, 2004, p. 28.
[54] Fra Luca Bartolomeo de Pacioli, Summa de arithmetica, geometrica, proportioni et proportionalita. Venice, 1494. Available via: http://www.cs.vu.nl/~x/yield/yield.pdf.
[55] R.J. Peters, C. Verhoef, Quantifying the yield of risk-bearing IT-portfolios, 2007. Available via: http://www.cs.vu.nl/x/yield/yield.pdf.
[56] L.H. Putnam, W. Myers, Measures for Excellence – Reliable Software on Time, Within Budget, in: Yourdon Press Computing Series, 1992.

[57] L.H. Putnam, D.T. Putnam, A Data verification of the Software Fourth Power Trade-Off Law, in: Proceedings of the International Society of Parametric Analysts – Sixth Annual Conference, volume III(I), 1984, pp. 443–471.
[58] Isabel Ramosa, Daniel M. Berry, João Á. Carvalho, Requirements engineering for organizational transformation, Information and Software Technology 47 (2005) 479–495.
[59] Suzanne Robertson, James Robertson, Requirements-Led Project Management: Discovering David's Slingshot, Addison-Wesley, 2004.
[60] Suzanne Robertson, James Robertson, Mastering the Requirements Process, second edition, Addison-Wesley, 2006.
[61] L. Rosenberg, L. Hyatt, Developing an effective metrics program, in: Proceedings of the European Space Agency Software Assurance Symposium, The Netherlands, 1996.
[62] S.A. Ross, R.W. Westerfield, B.D. Jordan, Fundamentals of Corporate Finance, fifth edition, Irwin McGraw-Hill, 2000.
[63] Frederick T. Sheldon, Krishna M. Kavi, Robert C. Tausworth, James T. Yu, Ralph Brettschneider, William W. Everett, Reliability measurement: From theory to practice, IEEE Software 9 (1992) 13–20.
[64] Harry M. Sneed, Peter Brössler, Critical success factors in software maintenance: A case study, in: Proceedings of the International Conference on Software Maintenance, ICSM '03, 2003, pp. 190–198.
[65] Ian Sommerville, Pete Sawyer, Requirements Engineering: A Good Practice Guide, Wiley, 1997.
[66] J. Stapleton, DSDM — Business Focused Development, 2nd edition, Addison-Wesley, 2003.
[67] J. Stapleton, P. Constable, DSDM — Dynamic System Development Method, Addison-Wesley, 1997.
[68] George E. Stark, Paul Oman, Allan Skillicorn, Alan Ameele, An examination of the effects of requirements changes on software maintenance releases, Journal of Systems Software Maintenance: Research and Practice 11 (1999) 293–309.
[69] George E. Stark, Al Skillicorn, Ryan Ameele, An examination of the effects of requirements changes on software releases, CROSSTALK The Journal of Defense Software Engineering (1998) 11–16.
[70] Martin J. Steele, Mansooreh Mollaghasemi Ghaith Rabadi, Grant Cates, Generic simulation models of reusable launch vehicles, in: Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers, Winter Simulation Conference, 2002, pp. 747–753.
[71] M. Takahashi, Y. Kamayachi, An empirical study of a model for program error prediction, IEEE Transactions on Software Engineering 15 (1) (1989) 82–86.
[72] P. Tavolato, K. Vincena, A prototyping methodology and its tool, in: R. Budde, et al. (Eds.), Approaches to Prototyping, Springer-Verlag, 1984.
[73] J.W. Tukey, Exploratory Data Analysis, Addison-Wesley, 1977.
[74] C. Verhoef, Quantitative IT portfolio management, Science of Computer Programming 45 (1) (2002) 1–96. Available via: www.cs.vu.nl/~x/ipm/ipm.pdf.
[75] C. Verhoef, Quantifying software process improvement, 2004. Available via: www.cs.vu.nl/~x/spi/spi.pdf.
[76] C. Verhoef, Quantifying the Value of IT-investments, Science of Computer Programming 56 (2005) 315–342. Available via: www.cs.vu.nl/~x/val/val.pdf.
[77] C. Verhoef, Quantifying the effects of IT-governance rules, Science of Computer Programming 67 (2–3) (2007) 247–277. Available via www.cs.vu.nl/~x/gov/gov.pdf.
[78] Hugh J. Watson, Celia Fullerb, Thilini Ariyachandraa, Data warehouse governance: Best practices at Blue Cross and Blue Shield of North Carolina, Decision Support Systems 38 (2004) 435–450.
[79] A.L. Weigel, D.E. Hastings, Interaction of policy choices and technical requirements for a space transportation infrastructure, Acta Astronautica 52 (2003) 551–562.
[80] Gerald M. Weinberg, Just say no! improving the requirements process, American Programmer 8 (10) (1995) 19–23.
[81] G.M. Weinberg, Quality Software Management: Volume 2 First-Order Measurement, Dorset House, 1993.
[82] A.S. White, External disturbance control for software project management, International Journal of Project Management 24 (2006) 127–135.
[83] Karl E. Wiegers, Software Requirements, second edition, Microsoft Press, 2003.
[84] Wikipedia. Bancassurance, Wikipedia http://en.wikipedia.org/wiki/Bancassurance, 2008 (accessed 01. 02. 2008).
[85] Didar Zowghi, N. Nurmuliani, A study of the impact of requirements volatility on software project performance, in: Proceedings of the Ninth Asia-Pacific Software Engineering Conference, APSEC'02, 2002, p. 3.