

Quantitative IT portfolio management

C. Verhoef

*Free University of Amsterdam, Department of Mathematics and Computer Science,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands*

Received 19 April 2002; received in revised form 10 July 2002; accepted 15 July 2002

Abstract

We present a quantitative approach for IT portfolio management. This is an approach that CMM level 1 organizations can use to obtain a corporate wide impression of the state of their total IT portfolio, how IT costs spent today project into the budgets of tomorrow, how to assess important risks residing in an IT portfolio, and to explore what-if scenarios for future IT investments. Our quantitative approach enables assessments of proposals from business units, risk calculations, cost comparisons, estimations of TCO of entire IT portfolios, and more. Our approach has been applied to several organizations with annual multibillion dollar IT budgets each, and has been instrumental for executives in coming to grips with the largest production factor in their organizations: information technology.

© 2002 Elsevier Science B.V. All rights reserved.

Keywords: IT portfolio; IT portfolio management; IT portfolio database; Cost allocation formulas; Seismic IT impulse; Operational cost tsunami; ROI threshold quavering; IT portfolio risk; IT portfolio exposure; Cost allocation distribution; Rayleigh distribution; Sech square distribution; Damped sine distribution; Generalized T distribution; Generalized F distribution; Total cost of ownership; Cost–time analysis; Lifetime analysis; Failure time analysis; Survival function; Hazard rate

1. Introduction

It is known from extensive research being conducted by the former CIO of the US Department of Defense, Paul A. Strassmann [119,120,123,124], that there is no relation between information management per employee and return on shareholder equity. Also there is no relation between profits and annual IT spending. So he shows that there is no direct relation between spending on computers, profits or productivity. Indeed, there are companies—in the same industry—each spending about the same on IT of which the one makes high profits, and the other makes huge losses [125]. This leads to

E-mail address: x@cs.vu.nl (C. Verhoef).

shotgun patterns showing the absence of correlations between any kind of return and the intensity of IT investments. The only vague correlation that Strassmann ever found was that when from two comparable enterprises one is spending slightly less than the other, the less spending organization is doing slightly better. This loose correlation leads one to suspect that governance of IT investments aids in creating value with IT instead of destroying profits. Indeed a continuous stream of reports on value destruction eventually led to the so-called Clinger Cohen Act [46], explicitly stating that the CIO's job is critical to ensure that the mandates of the Clinger Cohen Act are implemented. This includes that IT investments (we quote from [46]):

Reflect a portfolio management approach where decisions on whether to invest in IT are based on potential return, and decisions to terminate or make additional investments are based on performance much like an investment broker is measured and rewarded based on managing risk and achieving results

The US Government has to come to grips with IT portfolio management: any acquisition program for a mission critical or mission essential IT system for the US Government must be developed in accordance with the Clinger Cohen Act of 1996. To that end, the US General Accounting Office proposed a framework for initiating and maturing IT investment management [90]. But also outside the public sector there is increasing interest in deploying IT portfolio management. In a 2002 survey among 400+ top IT executives 60% reported an increase in the level of pressure to prove ROI on IT investments. But 70% believe their metrics do not fully capture the value of IT, and nearly half lack confidence in their ability to accurately calculate ROI on IT investments [71]. The Federal CIO Council summarized in 2002 the first lessons learned for IT portfolio management. The report does not mention quantitative approaches to manage IT portfolios [24]. This report should be seen complimentary to our work: the lessons learned provide a first insight in implementing qualitative aspects of IT portfolio management in organizations. This paper deals with *quantitative IT portfolio management*. In particular, we consider quantitative aspects of IT development, operations, maintenance, enhancement, and renovation for bespoke software systems. For other possible contents of an IT portfolio, such as license management for COTS systems, processing hardware infrastructure, network equipment, and so on, tools and techniques are available to deal with them. For instance, several companies are specialized in license management, and hardware/network infrastructure investment issues are better understood than software cost issues. In addition, Strassmann indicates that the focus on hardware costs is wrong: on the average this accounts for 5% of the life-cycle cost for information management so hardware is not the dominating factor [122, p. 409]. Bespoke software is our main focus. To the best of our knowledge quantitative IT portfolio management—the subject of this paper—is a terra incognita.

Executives of large organizations with substantial IT budgets learned the hard way that spending more is not the winning strategy. Some of them realized that after a long string of staggering IT investments plus their challenges, they must start to control their IT portfolios. Most executives consider IT spending as a black hole: no matter how much resources are thrown at IT, there is no clear justification of returns—IT is on top of the executives. We consulted executives about a variety of investment

Table 1
Distribution of organizations over CMM levels

CMM level	Meaning	Frequency of occurrence (%)
1 = Initial	Chaotic	75.0
2 = Repeatable	Marginal	15.0
3 = Defined	Adequate	8.0
4 = Managed	Good to excellent	1.5
5 = Optimizing	State of the art	0.5

related issues. For obvious reasons this consultancy was done under nondisclosure agreements. To give you an idea of the work, think of company-wide IT portfolio analyses, assessing current IT investments, projecting probable consequences of major IT investments for the IT budget in the coming years, quantifying IT portfolio risks (failure risks, cost overruns, underbudgeting risks), assessing the likelihood of added value of IT investments, assessing the IT portfolio of a potential target for a merge or acquisition, calculating minimal ROI threshold for an IT portfolio, etc. Based on these experiences, we developed and used formulas that form a mathematical underpinning of quantitative IT portfolio management. In other words: formulas that help in getting on top of IT. The examples we treat to explain these formulas are composed for that purpose, and do not relate to specific companies.

The bad news for many executives is that the area of software development is fairly immature. The failure rates of software projects are high: about 30% of software projects fail, 50% are twice as expensive, take twice as much time, and deliver half the functionality, and only 20% of the software projects is on time, within budget, and with the desired functionality [59,47–49]. In absolute figures, Standish group estimated in 1995 that this costed in the USA \$81 billion on failed projects, and another \$59 billion on serious cost overruns. Immaturity is further illustrated by the fact that 75% of the organizations worldwide are still at the lowest level of the capability maturity model (CMM), a five point scale developed by the SEI [93] indicating the maturity of an organization's software process.

In Table 1, taken from [66, p. 30], we show a recent distribution of organizations over CMM levels. As you can see, the majority of the organizations have chaotic processes to build and maintain software. Level 1 means that no repeatable process is in place, in particular, there is no overall metrics and measurement program. In a 2001 survey 200 CIOs from Global 2000 companies were asked about their measurement program. More than half (56%) said they did high-level reporting on IT financials and key initiatives. Only 11% said to have a full program of metrics used to represent IT efficiency and effectiveness, the rest (33%) said they had no measurement program at all [109]. This shows that the number of organizations without a proper metrics program is huge.

This implies that there is no substantial IT-related historical data to build a corporate governance structure on. No wonder that it is hard to detect relations between IT spending and return on investment. Imagine the consequences for an enterprise whose

financial department is anarchic. The problem for executives of level 1 organizations who want to get in control of their software portfolios *now*, becomes then very compelling. For, how to measure anything at the corporate level if there is no relevant data available to accumulate management information from?

Our approach to quantitative IT portfolio management provides you with insight in an IT portfolio, even in the case of level 1 organizations. A level 1 organization has to compensate for the lack of historical information by utilizing and deploying benchmark information. We developed a set of mathematical formulas based on public benchmark information to quantitatively manage IT portfolios. When you have historical data and can establish internal benchmarks, you can use these to instantiate our formulas.

The simplest formulas can be handled by spread sheets or a pocket calculator. For the more involved analyses, a scientific calculator like the HP49G or the TI89 can be used. For advanced issues, it is convenient to use statistical/mathematical packages ranging from spread sheets with plugins, to packages like SAS [31], SPSS [114], SPlus [129,73], Matlab [81], Maple [135], or Mathematica [132].

The set of formulas has shown to be a useful executive's armature to analyze, assess, and control IT portfolio issues, to counteract bombardments of jargon ridden empty promises by the trade press, software vendors, consultants, or proposals from their own internal IT departments. These formulas comprise relevant benchmarked IT-project information, which is often not provided to the executives, if only since the IT jargoneers have no clue how to come up with the information themselves. To use these formulas successfully you do not need to acquire extensive IT knowledge: we have completely hidden this aspect (but it is incorporated via benchmark information). The typical academic qualifications of the executive staff that we dealt with has an MBA combined with an M.Sc. or Ph.D. in an exact science. Think of economy, econometrics, astrophysics, experimental and theoretical physics, chemistry, biochemistry, mathematics, financial mathematics, business mathematics, statistics, biostatistics, electrical engineering, and so on. We encountered hardly ever educational backgrounds with a strong focus on IT, and when executives had such a background, they were combined with degrees in other exact sciences. Executive staff who were exposed to our formulas could understand them, could work with them, and were helped by them in that they lost their naivity with respect to IT decision making.

Does IT portfolio management pay off? In one organization the initial investment in research and development, plus the inevitable errors made during this endeavor, were about \$250 000 dollar. An additional \$120 000 dollar were needed for training. In this organization, the effort to keep the IT portfolio database up to date costs about \$50 000 dollar. The return is depending on the size of the IT portfolio measured in dollars. For a 500 million dollar IT portfolio direct cost savings per annum of 3–5% of the total portfolio value were established by better decision making: killing bad performing projects, mitigating risks, abandoning negative ROI investments, removing system redundancies, etc. As reported in CIO Magazine: just compiling an IT portfolio database saved one company \$3 million and another company \$4.5 million because the holistic IT portfolio view enabled them to spot redundancies [5]. Those redundancies could be eliminated, reaping the benefits.

For organizations above level 1, the accuracy of the underlying benchmarks can be significantly improved, and the underlying mathematics as presented in this paper can be instantiated accordingly. So, the principles of quantitative IT portfolio management remain unchanged. We will for the rest of this paper assume the level 1 situation so that the majority of current organizations can apply our results, yet when appropriate point out how our work applies to CMM 2+ levels.

2. How to tell a program from a security

In the 1970s, it was hard for many to grasp the difference between hardware and software maintenance, and Leslie Lamport explained this in a short note [74] entitled *How to tell a program from an automobile*. We borrowed this title to explain that you cannot simply apply security portfolio management to IT portfolios.

As far as we know there is no related work reported on in the realm of quantitative IT portfolio management, although many of us think that security portfolio management is strongly related. The reason is that the *goals* of both security portfolio management and IT portfolio management are largely identical. We will show that the *means* to reach this common goal do not coincide.

A lot of important work has been done in quantitative security portfolio management. And at first sight it seems a promising idea to support the quantitative part of IT portfolio management with the theory that 1990 Nobel Laureate Harry Markowitz developed on security portfolio management [80]. We heard this from several executives who were exposed to his work, but also the US Federal CIO council seems to play with this idea [24]. Moreover, the trade-press quotes people who think that applying this so-called modern portfolio theory is the *noble endgame* in IT portfolio management [5]. But what is this theory about? In the words of Markowitz [80, p. 205]:

Thus, in large part, this monograph is a discussion of the relationships and procedures by which information about securities is transformed into conclusions about portfolios.

The current paper is also a discussion of the relationships and procedures by which information about IT systems is transformed into conclusions about IT portfolios.

Markowitz investigated the available information about securities, the questions that need an answer and found the appropriate mathematical techniques to provide sensible answers to the problems. In the current paper we do exactly the same: we gather and investigate the data that is commonly available on IT projects, we investigated the questions that need an answer, and by working on the answers, we related the problems to the appropriate mathematical techniques to provide sensible answers.

Given these parallels, it may be hard for the financial expert, who might never have been exposed to information technology, (by building, maintaining, operating, enhancing, or retiring IT) to tell the difference between a program and a security.

Likewise, for the IT expert, who might never have been exposed to the nature of securities (by selling them, buying them, comprising them into portfolios, or advising others about these issues), it might be hard to tell the difference as well. As one executive put it [5]:

The cancellation rate of the largest IT projects exceeds the default rate on the worst junk bonds. And the junk bonds have lots of [portfolio management tools] applied to them. IT investments are huge, risky investments. It's time we do [Markowitz's portfolio management].

So they both are high risk investments, and another seeming parallel is implied. In the same article an IT portfolio manager said she had learned that some people argue against using modern portfolio theory since it cannot be overlaid exactly, and that the math is too difficult [5]. But what kind of math is being used in Markowitz's book [80]? As Markowitz states it [80, p. 186]:

The problem of maximizing expected return subject to linear constraints, [...] is a linear programming problem. [...] the problem of finding the portfolio whose smallest [return at time t] is as large as possible can be formulated as a linear programming problem.

The nature of securities is that you can select them, you can calculate their historical return, and based on these data you can use linear programming to calculate the optimal selection that maximizes expected return while minimizing variance by diversification. The underlying mathematics is elementary linear algebra, plus elementary statistics, and the use of standard linear programming techniques such as the simplex method. While this is not immediately digestible for the uninitiated, the math is not inherently complex. So, we agree that the complexity of the math should not be an argument against using modern portfolio theory as proposed by Markowitz. This is obviously also supported by the author of [5] given the supportive title of the article: *Do the MATH*.

But what about this overlay? The nature of securities is that you can invest and disinvest in them and implement a decision without prohibitively large costs, and often in a matter of seconds. Modern portfolio theory essentially asserts that investing and disinvesting at all times is enabled. The heart of security portfolio management is to monitor, control, and optimize the security selection process.

The Y2K problem and Euro conversion problem have clearly shown that you cannot simply junk IT systems, as is possible with underperforming securities. If you would have applied modern portfolio theory to IT systems that were suffering from the Y2K problem, you should have sold them, and bought better performing ones. For a start, to whom would you sell an IT system, especially, an IT system whose best-before-date is due? Maybe the competitor wants to buy them, if only to reverse engineer them to reveal your successful business rules, or the features that you cannot implement, so they can get ahead of you. Obviously, this is not an option, but even throwing them away, and restarting from scratch with new systems is not an option. For, IT is comprising business knowledge, often accumulations of it for many years. And converting such business knowledge into IT is a laborious, error-prone, and painful process, so abandoning all that valuable knowledge is like burning an entire profitable security

portfolio. Moreover, implementing such IT systems takes years of time. So the whole idea that there would be a free selection process, that companies are totally free to choose in which IT they want to invest, that organizations can abandon systems like junking bonds, is not in accord with the realities of large software portfolios.

Let us give a typical example showing the nature of IT investments, and the freedom to choose. Suppose you have a good idea, like a credit card, or an ATM. Both inventions are IT intensive, and both ideas were implemented by leading financial corporations with deep pockets. In the beginning, they created value for both themselves, and their customers. But then the competitors united and answered with commoditized shared credit cards and a shared ATM network—not sharing with the initiator of course. While still creating value for the customers, this competitive answer destroyed not only the profits for the initiators, but also the margins for the entire market. The discretionary IT investment, thus turned into a commodity, and now you cannot operate a financial corporation without credit cards, or ATMs. They have become must-have's while the profit margins are gone—in some cases you even make a loss because of them. So, you are not free to disinvest in credit cards or ATMs anymore, since the innovation set off within the industry and customers insist on these services. This is sometimes called: *creating value while destroying profits* as explained in detail for the banking sector in [118]. But according to Markowitz's portfolio theory, these are portfolios to disinvest in: they cost money and do not deliver any positive return. Again, Markowitz's theory is here a bad advisor, because these types of projects are really non-discretionary.

Let us give another example to make this apparent. Once you opted for a particular operating system and accompanying languages it is no longer the case that you can easily switch. Once you *select* a technology for your IT, you cannot easily abandon it. In the words of Michael Porter: there are low entry barriers, but high exit barriers [97, p. 22]. IT systems often make essential use of the underlying operating system idiosyncrasies, clarifying the prohibitively high switching costs. This is not the case with securities. Also, people *work* with IT systems and they have to learn a new IT system, which is not the case for a security. So also the switching costs for users of the IT systems are high. Moreover, switching to a different computer language or even another dialect is hardly possible for existing software assets [126]. When attempting to convert to another language, you also need to convert your IT personnel successfully. A 2001 Gartner report shows that converting a Cobol developer to a professional Java developer has a chance of failure of about 60% [37, p. 22]. Also a change of technology implies often a change in business process. This has shown to be a challenge in itself, with a high failure rate [113]. So, information technology becomes illiquid after a first selection as opposed to securities or other financial assets that do have liquidity. Changing information technology bears large risks, is virtually impossible both technically and organizationally, and takes huge amounts of time. So again in this situation, Markowitz's portfolio theory is not an adequate tool to support IT decision making.

Also the issue of diversification is not simply transposed to the IT situation. In order to mitigate risks, and stabilize expected portfolio return, a diverse security portfolio is a good idea, and Markowitz gives the underlying mathematical tools to optimize this

situation. What does diversification mean in the realm of IT? Should we refrain from many similar systems and make the IT systems more different? For instance, by using more languages, different operating systems, a host of development and maintenance tools? For IT you often try to do the opposite: consolidate different but similar systems into a single overall system: a product line that deals with the variation points of the similar systems successfully [10,22]. Also standardization is a keyword in the IT branch: use only a few languages, a limited number of operating systems, and a few support tools. So it is not a good idea to diversify in a technical sense, because of knowledge investment, transfer, complexity, etc. Apart from these technical aspects, there is also a business aspect. If you are a company building integrated radar systems for warships, then you are building those, and not enterprise resource planning systems for the automotive industry. To implement both types of systems, you need a lot of domain knowledge of both areas, e.g., for the warships you need to know about developing software under military regulations, whereas for the automotive industry entirely different issues are at stake. It is simply not very productive to combine uncorrelated IT domains. So also here, you see that the notions that are natural and relevant for security portfolio management are without rhyme or reason for IT portfolio management.

Another aspect is that for securities there is a rich body of historical information. In contrast, many IT systems lack all historical information, and an entire branch in software engineering—reverse engineering—is devoted to cope with this problem [20]. The information that is around, is out of date, since deployed IT systems are in continuous flux, and IT developers are not stars in writing documentation. While the value of a security may fluctuate on the stock exchange, the object itself does not change a bit. So the nature of the available information is different, the nature of the objects is totally different, but also the relevant questions are different. Markowitz's modern portfolio theory focuses on selection as a tool to minimize risk, and maximize return, while IT portfolio management is not at all about selecting and abandoning. So modern portfolio theory as proposed by Markowitz is not applicable *at all* to IT portfolio management. Therefore, we are not surprised to read in the same article that advocates the use of Modern Portfolio Theory (MPT) [5]:

Who's using Markowitz's Modern Portfolio Theory in IT? Not many.

Simply, because it is not applicable. Also in paper [3] it was observed that there are problems with applying Markowitz's modern portfolio theory:

However, we have not yet seen any example of a substantial software project actually using these techniques to help in their decision making process. We attribute this to the fact that obtaining economic data in software projects is much harder than in financial markets from where these techniques have been borrowed.

The idea that the data is the problem is erroneous, as we will see later on. The reason is that the nature of software does not resemble the nature of a security.

Not only at the portfolio level but also at the single system level, people are thinking of using modern portfolio theory. As an example, we quote a paper [16] investigating the potential of using modern portfolio theory to guide decisions in software

engineering:

We view each software activity as an investment opportunity (or security), the benefit from the activity as the return on investment, and the allocation problem as one of selecting the “optimal” portfolio of securities.

The paper claims that many decisions are ad hoc, and that portfolio selection could improve that situation. But then the subjects of selection should comply with modern portfolio theory. According to modern portfolio theory, you can lower risks with large numbers of uncorrelated securities. We quote Markowitz [80, p. 102]:

We see that diversification is extremely powerful when outcomes are uncorrelated. [...] To understand the general properties of large portfolios we must consider the averaging together of large numbers of highly correlated outcomes. We find that diversification is much less powerful in this case. Only a limited reduction in variability can be achieved by increasing the number of securities in a portfolio.

This leaves us to answer two questions: Is the amount of activities large? And, are these activities uncorrelated? According to the activity-based cost estimation literature, there are at most 25 main activities in software development and deployment [66]. Moreover, many of these activities are correlated, and if they are not correlated, there is no free choice. Analysis, design, coding, testing, and operations: they are all strongly correlated. While you can drop analysis and design, it is already well-known that this is not leading to the best possible IT systems. You do not need MPT to decide on these issues. So MPT does not transpose that easily to the software world. The authors of paper [16] admit that applying MPT did not work out in a subsequent paper [17]:

We have been attempting to apply financial portfolio analysis techniques to the task of selecting an application-appropriate suite of security technologies from the technologies available in the marketplace. The problem structures are sufficiently similar that the intuitive guidance is encouraging. However, the analysis techniques of portfolio analysis assume precise quantitative data of a sort that we cannot realistically expect to obtain for the security applications. This will be a common challenge in applying quantitative economic models to software engineering problems, and we consider ways to address the mismatch.

The authors seem to think that MPT is the solution, and the problem is missing data. This is not true.¹ But their findings are confirming the fact that 75% of the organizations is not mature with respect to their software development and deployment [66].

¹ Apart from the arguments that we give in this paragraph, there is also a fundamental argument indicating that the premises for applying MPT are not fulfilled. This argument may be less easy to comprehend upon first reading this paper. In [41, p. 293] it is stated that if the number of activities in a software project is large, and if these activities are uncorrelated, then according to the central limit theorem, the cost allocation distribution will approach the Gaussian distribution. Empirical evidence indicates that cost allocation functions for R&D projects, software projects, and IT portfolios are not normally distributed. This is shown in [87–89, 101–104, 98] and in this paper.

The number of main activities is so low and their correlation so high, that applying MPT is senseless. Moreover, there are alternative approaches to select activities or technologies. For instance, in [82] an extensive list of best practices is presented. Each with an indication of entry barriers, and the risk of applying the technology. Also in [65] a table is presented providing the approximate return on investment of deploying certain software technologies. In [63,66] an elaborate treatise of success and failure factors for development and deployment of software in various categories is discussed, backed with quantitative data.

There are many more differences that come to mind, but we hope you get the idea by now: a program is not a security, it will never become one, and better data is not going to help. The goal of this paper is not to argue that security portfolio management does not correspond in a simple one-to-one manner to IT portfolio management, but to provide you with the mathematical underpinning that does enable quantitative IT portfolio management. It all starts with collecting the available information, and the hope that this information can be used to answer questions relevant for quantitative IT portfolio management.

3. Gathering information

Since level 1 organizations have chaotic software development and deployment, not a lot of relevant IT portfolio information is readily available, let alone uniformly accessible. We give a few possibilities of information availability ranging from the ideal situation to the worst possible case.

- All the information needed for quantitative IT portfolio management is uniformly accessible via an IT portfolio database, which is continuously updated. Examples of important project information are initiation date, delivery date, staff build-up, staff size, various IT specific indicators, total development costs, annual costs of operation, total cost of ownership, risk of failure, net present value, return on investment, internal rate of return, pay back period, risk adjusted return on capital, and so on.
- Initiation date, delivery date, staff size, and total project development costs are available.
- Initiation date, delivery date, and total project development costs are available.
- Project duration and total project costs are available.
- Total costs are available.
- No management information is available.
- Source code of the system is available.
- Source code is not available.

Except for the ideal situation, we have experienced all of the above cases—or combinations of them—in every substantial IT portfolio. Mostly, we were able to somehow derive the single most important IT specific key indicator underlying quantitative IT portfolio management formulas. It is the number of function points [1] for each application in the IT portfolio. We will indicate shortly how we do this.

3.1. Function points

A function point is a synthetic measure expressing the amount of functionality of a software system. It is programming language independent, and it is very well suited for cost estimation, comparisons, benchmarking, and therefore also a suitable tool for developing quantitative methods to manage IT portfolios [108,66]. In a textbook on function point analysis [40, p. xvii, 28] we can read:

As this book points out, almost all of the international software benchmark studies published over the past 10 years utilize function point analysis. The pragmatic reason for this is that among all known software metrics, only function points can actually measure economic productivity or measure defect volumes in software requirements, design, and user documentation as well as measuring coding defects. [...] Function points are an effective, and the best available, unit-of-work measure. They meet the acceptable criteria of being a normalized metric that can be used consistently and with an acceptable degree of accuracy.

Function points have proven to be a widely accepted metric both by practitioners and academic researchers [70, p. 1011]. For executives it is important to know how reliable these metrics are. In [33, p. 132] an accuracy of 15–20% is mentioned, as well as a 2300% variance in productivity when other metrics were used. This was due *only* to extremely wide variations in 7 definitions of the number of source lines of code (SLOC) of a computer program. In addition, you need to know what the so-called interrater reliability of accredited function point analysts is. Interrater reliability is the consistency between raters. If the variation is high, then the method is not as good as when the variation between raters is low. Moreover, since there are more methods to count function points, what is the intermethod reliability? Empirical research reports that the median difference in function point counts from pairs of raters using Albrecht's function point counting method was about 12%. The correlation across two different methods that were used in this field study was 0.95 [69, p. 88]. So function points can be seen as an objective measure and the intermethod reliability is sufficiently high as to consider comparing function point totals that resulted from more than one counting method. Apart from people, you can also use tools to count function points for existing software assets. One method called backfiring has an accuracy of about 20% [66, p. 79]. One of our students developed for a large financial enterprise a function point counting tool that automatically counts function points with a maximal deviation of 3% of manually counted function points by accredited function point analysts [86]. This is a language specific tool and only counts function points for information systems in Cobol with SQL and/or CICS on a mainframe.

To reassure you at this point: there is no need to understand the details of function points in order to use our results. We use them for several purposes: to recover missing management information, to derive some of our formulas, and to check our projections against the actual amount of function points of an application (if the function point totals are known already). In this paper we mostly encapsulate the IT specific function point metric and derive formulas solely expressed in the language of management: cost, project duration, staff size, risk, return, etc. What is important for

now to know is that function points are a suitable basis for quantitative IT portfolio management.

3.2. Recovering management information

For some systems, neither management information nor up-to-date documentation is available. In that case we have to resort to the IT artifacts themselves to infer the management information that we minimally need for quantitative IT portfolio management. For the majority of these systems, the source code is available. The number of function points can be estimated using backfiring [62]: with a tool we count the number of logical computer statements, and depending on the used language, the number of function points can be estimated via a conversion table. For instance, a system with 100 000 Cobol statements, is 937 function points according to benchmark.² Via extensive benchmarks it has been empirically found that 1 function point of software is equivalent to 106.7 Cobol statements. For the about 500 languages in use worldwide there is a list with such conversion factors and there are tools available that implement backfiring. Backfiring has a somewhat larger margin of error than other techniques, nevertheless it suffices for recovery of function point totals for the often small part of the IT portfolio lacking all management information. If the amount of systems without any kind of management information is large, we need to resort to more accurate tools that scan the source code to calculate the amount of function points (e.g., the tools in [86]).

Usually about 5% of an IT portfolio lacks not only all management information but also the source code itself [65, p. 129]. Then we use a tool called a disassembler that turns the binary code into a list of assembler instructions. For assembler instructions we can use the backfiring approach: for a variety of assembly languages conversion factors are available. So, e.g., a load module on an IBM mainframe lacking the sources, that consists of 300 000 assembler instructions after disassembly, is also about 937 function points (the conversion factor for IBM mainframe assembler/3X0 is 320). If the amount of lost sources is substantial, then you need more sophisticated tools. They are available, in case you wondered [21,38]. But when you are missing the majority of the source code, you have other priorities than quantitative IT portfolio management. For example, in one case an enterprise-wide inventory to set up quantitative IT portfolio management revealed a huge exposure: we detected a business unit where more than half of the IT systems could no longer be recompiled, due to lacking source code. The exposure was unacceptable since the systems needed a Y2K update, which was almost impossible without source code. Executive management immediately acted to mitigate the risks.

So if there is only source code or an executable we can recover function point information, and from that we can infer management information as we will show later on. As an aside, you can already see why function points are so well-suited for IT portfolio management. We can compare assembler projects to Cobol projects without any problem. The conversion factors 106.7 and 320 are in fact expressing that

² We will use the phrase *according to benchmark* in this paper to indicate that the quantitative data is in accord with public benchmarks, but not necessarily exactly accurate for a single instance.

a Cobol statement is about 2.99 Assembler instructions. Just imagine function points to be a universal IT-currency converter between different projects.

3.3. Enriching management information

Suppose that the burdened daily compensation rate of IT personnel is \$1000, with 200 working days per year. If only the total development costs are known, then we can infer more information using our formulas (we discuss them shortly). Namely, for a 5 million dollar IT project it is likely that this project took 20 calendar months with about 15 people involved. Of course, we cannot be sure about this: this is derived using public benchmarks. We can check this with an additional function point count. For instance, if we counted about 3000 function points, we can check that this should have cost 22 calendar months, for 15 people. But if it was only a 1000 function point project, the costs were probably too high. It would be ideal to have function point totals for the entire IT portfolio, but since this implies physical access to source code this is not a short-term viable option for globally operating companies to jump start quantitative IT portfolio management. In the long term, collecting function point totals for large parts of the IT portfolio is feasible.

In most organizations we have encountered the following situation: both project duration and development costs are accessible without too much trouble. With this information we can calculate the costs according to benchmark and compare this with the actual costs. As a rule of thumb, you need to cross check in a few cases:

- only a very long project duration is mentioned (this presumably implies high costs);
- only a very large cost is mentioned (this implies a potentially large project);
- a very long project duration is mentioned *and* very low costs;
- a very large cost is mentioned accompanied by a very short project duration.

Of course, the more management information is available, the less information you have to infer, the more you can use the formulas to validate the provided data, and infer company-specific internal benchmarks and formulas. The more accurate the data underlying our mathematics becomes, the more accurate your quantitative IT portfolio management, up to the point where you can continuously control and monitor past, present, and future IT costs and benefits in your IT portfolio.

Discounting costs. If the cost of a project was 100 000 US dollar 20 years ago, then this project is completely different from a \$100 000 project today. Obviously, inflation is a factor that should be taken into account when dealing with costs over time. As a side-remark, also deflation is a factor that should be taken into account. For instance, in South-American countries depending on the age of a project you may have to divide the numbers by 1000, due to a currency reform.

Current-dollars, than-dollars, future-dollars, currency conversions, and notions like (risk adjusted) discounted cash flows are well-known issues within the area of economy. They deal with correcting the difference in dollars over time. If you eliminate this aspect by converting all our formulas to effort-time analyses—formulas in terms of staff instead of costs—you exchange discounting IT dollars for discounting

IT productivity. This is more accurate than discounting cash flows. But, CMM level 1 organizations do not know their own productivity, do not have historical data on past productivity, and cannot predict future productivity, so they cannot discount effort using IT productivity. Higher-level CMM organizations can discount in this way, and have the potential to eliminate the additional problem of discounting cash flows. For CMM level 1 organizations you can use as a surrogate the appropriate corrective transformations that are known from economy. In this paper we abstract from doing these conversions, although they are necessary for practical long-term estimates. The reason for this is that we want to unravel and expose the unknown territory of IT portfolio management. Once this is unraveled, we can apply the known economic tools and techniques to discount the cash flows.

For large global IT portfolios you sometimes use a mix: for those countries where the discounted cash flows are well-known, with larger accuracy than IT productivity, it is better to discount cash flows. But if it is problematic to discount the cash flows, then taking IT productivity constant over time is a viable alternative: given Brook's law that there is no single technology that can boost programmer productivity by an order of magnitude [13], and given a constant stream of data, programmer productivity is not wildly changing from one year to another in CMM level 1 organizations.

3.4. *Compiling an IT portfolio database*

We showed a few methods to recover function points from software projects lacking all or almost all management information. Now we show how with often recoverable, but still rather limited management information you can compile and check a corporate wide IT portfolio database. It is very useful to collect the following information in a simple database for IT projects in the entire organization:

- initiation date;
- delivery date;
- total project costs.

This is not much as a source of information, but it will be the best you can do in a level 1 organization for the majority of the completed and proposed projects. Of course, in some cases more data is available such as staff size. Needless to say that all additional relevant information is welcome: staff size, how many internal and external staff, how many working hours, and so on. You can use this information to cross check the data, and obtain an impression of its accuracy. However, rich project information will often lack. So we show how to proceed with the above three pieces of information only.

Often large enterprises are not transparently web-enabled so that software project information is mostly paper-based [125]. It is not possible to study all these project documents, but it is mandatory to collect as many as possible. Since this easily tops thousands of documents it is unavoidable that non-experts enter the abovementioned data. We know they make errors. We also experienced that the project information itself contains irregularities, and is far from uniform. The first step after compilation

of the database is to thoroughly validate its contents. For, it will be the source of information to base your IT portfolio management strategy on. We use quantitative methods to check the database.

4. Relating cost, duration, and staff size

For the compiled IT portfolio database containing data for project duration and total development cost we want to check whether the entries make sense at all. To do this, we will derive in this section the first eight formulas for quantitative IT portfolio management.

We explained that for CMM level 1 organizations we have to somehow compensate for the lack of historical data and the lack of an overall metrics program, and in this section you will see how we do this. We use benchmarked relations between cost, duration, and staff size to develop our formulas. Consider the benchmark taken from [64, p. 202]:

$$f^{0.39} = d$$

Here f stands for the number of function points [1] and d is project duration in calendar months (that is, elapsed time measured in months). Recall that it suffices to imagine f to be a universal IT-currency converter. The power 0.39 is specific for MIS software projects that are part of all businesses and omnipresent in financial services and insurance industry.

How much belief should we have in such a benchmark? There is no established tradition in software benchmarking and therefore the amount of projects subject to benchmarking is relatively low. For instance, the latest benchmark release of the ISBSG (International Software Benchmarking Standards Group) is based on the 789 projects that were submitted to their repository in early 2000. We base our work mostly on Jones' database who has probably the largest knowledge base on software projects in the world: in 1996 it contained 6753 projects [62, p. 161]. In 1999 this has grown to more than 9000 projects. We provide the distribution of the projects over size and type in Table 2. The database contains data regarding software comprising over 10 million function points. To compare, a large international bank approximately owns 450 000 function points of software, and a large life insurance company possesses 550 000 function points [62, p. 51]. Jones partitioned his project database also over age range, partitioned it into new, enhancement, and maintenance projects, partitioned it over selected programming languages, and over a number of technologies used; see [62, pp. 162–164] for details. The overall set of projects used in producing the benchmarks is biased: the database contains presumably more large projects than in reality [62, pp. 159–160]. But a CMM level 1 organization has no historical data to derive its own internal benchmarks, so we resort to Jones' work as a surrogate for lack of historical information. Note that the schedule powers vary for different code sizes, and for different industries. We give you a few of such benchmarked powers that you can use depending on the industry you are in, or depending on the type of project. Table 3 contains a few of them and is taken from [64, pp. 202]. For ease of

Table 2
Distribution of Jones' project database around 1999

Size (FP)	End user	MIS	Out source	Commercial	Systems	Military	Total
1	50	50	50	35	50	20	255
10	75	225	135	200	300	50	985
100	5	1600	550	225	1500	150	4030
1 000	0	1250	475	125	1350	125	3325
10 000	0	175	90	25	200	60	550
100 000	0	0	0	5	3	2	10
Total	130	3300	1300	615	3403	407	9155
Percent	1.42	36.05	14.20	6.72	37.17	4.45	100

Table 3
Various powers derived by benchmarking

Power	Projects within range
0.36	OO Software
0.37	Client/Server Software
0.38	Outsourced Software
0.39	MIS Software
0.40	Commercial Software
0.40	Mixed Software
0.43	Military Software

explanation we will mostly use the power 0.39. See Fig. 1 to get an idea of how the various benchmarks look like when we plot these benchmarks for the schedule powers of Table 3.

In addition to the schedule power benchmarks, we mention two other benchmarks, also taken from [64, pp. 202, 203]:

$$\frac{f}{150} = n, \quad \frac{f}{750} = n,$$

where n is the number of staff, necessary to do the project. These are overall benchmarks, not specific for the MIS industry. The left-hand formula applies to software development projects, and the constant 150 is specific for those. The right-hand formula is to estimate staff for keeping an application operational, and the constant 750 is specific for this type of work. This excludes large functional enhancements, for which there are other benchmarks.

We stress that you should not use these formulas as the sole decision means for individual software project contracting purposes, to decide on individual resources for large software projects, or to decide on upcoming individual projects with potential large business impact. Namely, for these applications the above estimation formulas are not specific enough, and might even be harmful. But they can be used for sanity checking on individual projects and are accurate enough for decision making for entire

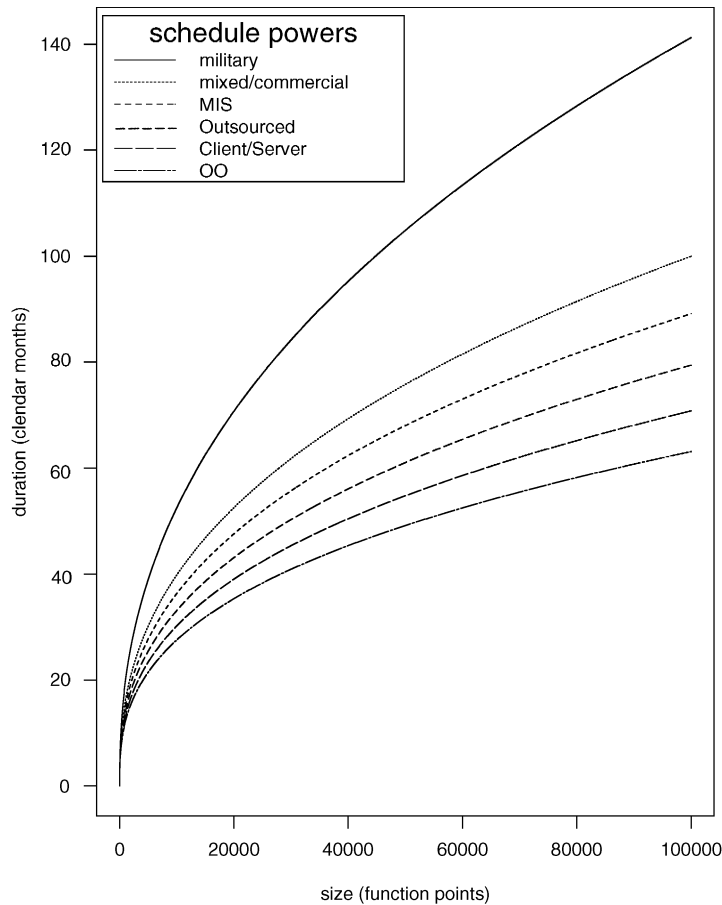


Fig. 1. Visualizing the schedule powers of Table 3.

IT portfolios. In [63] it is stated about these benchmarks (originally intended for manual cost estimation):

Manual methods are quick and easy, but not very accurate. Accurate estimating methods are complex and require integration of many kinds of information. When accuracy is needed, avoid manual estimating methods.

We want to support decision making for IT portfolios and not for individual projects, and the benchmarks plus their derived formulas have shown to be an excellent vehicle for that purpose. Note that optimal accuracy is not feasible at level 1, since the required data richness is simply lacking. Of course, you can use our formulas for sanity checking as well on a per project basis.

We derive from the above benchmarks our first quantitative IT portfolio management formula, that we will use to check the database. We recall that for CMM levels higher than 1 the principles of our approach stay the same, only the benchmarks instantiating

our formulas change. Since there is more and better data available, it becomes feasible to infer company-specific benchmarks, leading to more accurate instantiations of our first formula. We discuss the instantiation using public benchmarks so that level 1 organizations can use it.

We show a simple algebraic derivation to illustrate how you can derive your own instantiation of our first formula in case you want to apply our results. First calculate how many function points the application is according to benchmark.

$$f = d^{\frac{1}{0.39}} = d^{2.564}.$$

We made f explicit using elementary algebraic arithmetic. With the second benchmark, we calculate the number of people for the development project:

$$n = \frac{1}{150} d^{2.564}.$$

So, the total number of calendar months m needed to accomplish the project is $m = d \cdot n$. This amounts to

$$m = \frac{d}{150} d^{2.564}.$$

We assume that there are w working days in a year, and for a given daily burdened compensation rate r we can now calculate the total cost of development $tcd(d)$ for a given project duration d :

$$tcd(d) = r \cdot \frac{w}{12} \cdot \frac{d}{150} \cdot d^{2.564} = \frac{rwd}{1800} \cdot d^{2.564} = \frac{rw}{1800} \cdot d^{3.564}$$

So the first formula for quantitative IT portfolio management is

$$tcd(d) = \frac{rw}{1800} \cdot d^{3.564}. \quad (1)$$

Completely analogously, we derived a formula calculating the total cost of a maintenance project for a given duration d (for costs to keep a system operational you do not know the duration per se, and we will derive other formulas for that, see formula 11). For this second one we used the 750 benchmark for maintenance staff size, everything else being equal. Formula (2) for quantitative IT portfolio management then becomes

$$tcm(d) = \frac{rw}{9000} \cdot d^{3.564}. \quad (2)$$

Of course the rates will vary per organization but also per task: we have used daily rates ranging from 500 to 4000 US dollar. Certain programmers for ERP packages can be more expensive, but also experts in high performance transaction processing at large banks and in the airline industry can be quite expensive (think of TPF experts [55]). We experienced that daily burdened compensation rates for both internal staff and external specialists were always available. Also the number of working days varies

per organization and per country and can range from less than 200 to 230+ days per year. You have to use your own company-specific rates to instantiate formulas (1) and (2) for quantitative IT portfolio management.

If you only know the total cost of an IT project, you can calculate the project duration according to benchmark. We can do this with the dual versions of the first and second formulas. Without bothering you with the details of their (mathematically trivial) inference, we formulate formulas (3) and (4) for quantitative IT portfolio management:

$$dd(c) = \left(\frac{1800c}{wr} \right)^{0.28}, \quad (3)$$

$$md(c) = \left(\frac{9000c}{wr} \right)^{0.28}, \quad (4)$$

where dd is development duration, md is maintenance duration and c is the known total cost of either a development or a maintenance project.

We also experienced that actual staff size is sometimes available in a number of business units. It is very useful to collect this information as well in the IT portfolio database. Formulas (5) and (6) provide you with a benchmarked relation between staff size and project duration:

$$nsd(d) = \frac{d^{2.564}}{150}, \quad (5)$$

$$nsm(d) = \frac{d^{2.564}}{750}. \quad (6)$$

where nsd is the number of staff for development, and nsm is the number of staff for maintenance projects. With formulas (5) and (6) you can detect differences between actual staff size and benchmarked staff size. In the same way, you can detect differences between actual cost and benchmarked cost. Likewise, we can calculate for a given staff size the benchmarked duration, leading to formulas (7) and (8) for quantitative IT portfolio management (QIPM):

$$dd(n) = (150n)^{0.39}, \quad (7)$$

$$md(n) = (750n)^{0.39}, \quad (8)$$

where n is the actual staff size, dd is development duration, and md is maintenance duration.

We already made some example calculations, and we will make some more example calculations to illustrate the use of these formulas to support quantitative IT portfolio management. Throughout the paper we assume for all example calculations a fictitious company with a daily burdened rate of \$1000 both for development, maintenance, internal and externally, and 200 working days per year. This leads to the following

instantiations of formulas (1)–(4) for QIPM:

$$\begin{aligned}
 tcd(d) &= \frac{1000}{9} \cdot d^{3.564}, \\
 tcm(d) &= \frac{200}{9} \cdot d^{3.564}, \\
 dd(c) &= \left(\frac{9c}{1000} \right)^{0.28}, \\
 md(c) &= \left(\frac{9c}{200} \right)^{0.28}.
 \end{aligned}$$

For example, a 36 month development project costs $tcd(36) = \$39.1$ million according to benchmark, and an 18 month maintenance project costs according to benchmark $tcm(18) = \$0.66$ million. We announced earlier that when you know the costs, you can calculate the duration. For a \$5M development project $dd(5M) = 20$ indicating that such a project should take about 20 months. The number of staff $nsd(20) = 15$ leading to 300 calendar months, or 25 man-year, which is indeed \$5M. Likewise, a \$5M maintenance project, takes $md(5M) = 31.5$ calendar months according to benchmark and takes $nsm(31.5) = 9.3$ maintenance staff. This is 293 calendar months, leading to \$4.9 million, which accurately approximates the actual financials.

4.1. Cleaning the IT portfolio database

Using formulas (1)–(8) you can check and clean your just compiled IT portfolio database, by carrying out the process outlined below. Note that the goal is not to comply with our formulas as closely as possible—the goal is to spot the differences so you can locate and correct erroneous IT portfolio database entries after which the real deviations according to benchmarked management information are revealed. These deviations need the attention of the *IT portfolio manager* or IPM, which can be the CIO, a board member, or someone directly reporting to the executive board.

- Calculate the project durations from available data such as start and end dates.
- Complete the portfolio database with lacking project durations, or financials of all projects beyond some threshold. Depending on the size of the company this threshold is higher or lower. For large organizations we experienced that 5 million dollar is an acceptable threshold. For our fictitious organization this means that when reported costs of development or maintenance exceed \$5M, the actual project duration needs to be recovered by inspecting the documents, or if all else fails asking the responsible IT manager. Likewise this implies that for $dd(\$5M) = 20$ calendar month development projects and $md(\$5M) = 31$ calendar month maintenance efforts, the actual data needs to be recovered. We use the following simple guidelines for that:
 - between \$5 and 10 million, contact the system owner and just ask for the lacking actuals;
 - between \$10 and 25 million, in addition to asking for actuals also ask for the full documentation of the IT project;

- for IT investments exceeding \$25M, lacking any other management information (which is not unusual in level 1 organizations), additionally do a function point analysis.

If instead of costs, management information comprises duration or staff size, use one of our other formulas to calculate the appropriate thresholds. For instance, using formula (3) we calculate that $dd(\$5M) = 20$, and $dd(\$10M) = 24$, so the first project duration threshold for our fictitious company is between 20 and 24 months. The others are inferred similarly.

- Calculate only development costs according to benchmark for all projects (using formula (1)). Mostly, IT projects that just keep applications running are combined with projects that add functionality to the business. In the less common case that pure (corrective) maintenance projects are separately reported, you will see a deviation with the formula (1). Then you can correct for this by using formula (2).
- Compare the benchmarked costs with the actual financials in the database and record obvious deviations.
- Assess the deviations, and correct errors in the database if that was the cause. We list a few common causes: project costs can be erroneously converted from local currencies to the chosen currency for the database, no conversion was done, no one could find the Euro sign so the local currency sign is used, with the undocumented assumption that it should be read as Euro amounts. We also encountered the wrong number of zeros, decimal commas in the wrong location, and so on. Also erroneous date information is a common cause for errors. For instance, the initiation and delivery dates are exchanged, or a typo in the year is made leading to a huge difference, but also date formats are not uniform. For a globally operating organization, we experienced that the following issue was a common cause of errors: the date 02/03/04 means in Japan the 4 March 2002, in the USA it stands for the 3 of February 2004, while in other parts of the world the 2 of March 2004 is implied.
- Assess the remaining deviations. First look at projects that are much too low compared to the actuals. They may be corrective maintenance projects. Separate out these projects and use formula (2) to carry out the process we are describing.
- When the financials are much higher than expected, inspect the available project data. It can be that large hardware costs are incorporated in the IT project's budget, then correct for them. It can also be the case that an expensive software package or tool is acquired for this project. Think of ERP implementations. Then correct for the package costs, and check whether the daily compensation rate is sufficient, since ERP specialists can be much more expensive than other programmers. Then separate out these projects and use an ERP instantiation of formula (1) with the appropriate fully loaded daily compensation rate.
- When the financials are much lower than expected, it may be the case that this project was outsourced to a cheap labor country. Inspect the available project data to correct for several things: first of all the power should be altered from 0.39 to 0.38 in accordance with Table 3 for outsourced software. Also check for the number of working days per annum in the country of outsourcing, and finally correct for

the daily compensation rate. Use the off-shore outsourcing variant of formula (1) for development projects, and a variant of formula (2) for operations projects for these items in the IT portfolio database.

- It can also be the case that projects under the same name are clustered. This can be a cluster of simultaneous projects. Typically, you will get a very high total cost for a much too short time frame. Then the projects need to be separated out, and their durations listed separately. You cannot simply add project durations, since there is no linear relation between duration and cost. Or in an equation:

$$tcd(d_1 + d_2) \neq tcd(d_1) + tcd(d_2).$$

For example take a cluster of two projects, with $d_1 = 12$ and $d_2 = 5$. They separately cost $tcd(12) = \$0.77$ resp. $tcd(5) = \$0.03$ million, so in total \$0.80 million. But a $d_1 + d_2 = 17$ month project leads to $tcd(17) = \$2.69$ million, so more than 3 times the total of the two projects.

- You also see entries where the project duration is very long but the costs are very low. Then often people accumulated several related projects into one, and added the various project durations, e.g., the start and finish dates of a Gantt chart [38,104] comprising all subprojects. Also these projects should get separate entries plus the time spent for each project in the cluster. Like above, you cannot add costs because of the nonlinearity of the relation, or in an equation:

$$dd(c_1 + c_2) \neq dd(c_1) + dd(c_2).$$

For example, two subprojects costing $c_1 = 1$ and $c_2 = 2$ million dollar, respectively, will have a duration of $dd(\$1M) = 12.7$ and $dd(\$2M) = 15.5$ months each according to benchmark. While a $c_1 + c_2 = 3$ million dollar project takes $dd(\$3M) = 17.4$ months according to benchmark. However, the combined subprojects take 28.3 months, which would incur a cost of $tcd(28.3) = 16.7$ million dollar according to benchmark.

- If the deviations can no longer be clarified by errors, inaccuracies, linear thinking, or special project characteristics, the error correction process is finished. This does not mean that the contents is correct. It just means that most of the errors are gone, which is good enough for starting quantitative IT portfolio management.
- Calculate all instances of formulas (1) and (2), containing the appropriate rates and number of working days per year, for all projects.

The above process leads in a relatively short time to a reasonably accurate IT portfolio database. Now we are ready to analyze its contents.

5. Analyzing an IT portfolio database

In Fig. 2 we composed a random excerpt of several IT portfolio databases from organizations that went through the above process—let's call this sample S for later reference. We changed daily compensation rates and variations in working days per year to the example rates: a \$1000 daily rate, and 200 working days per annum. Moreover,

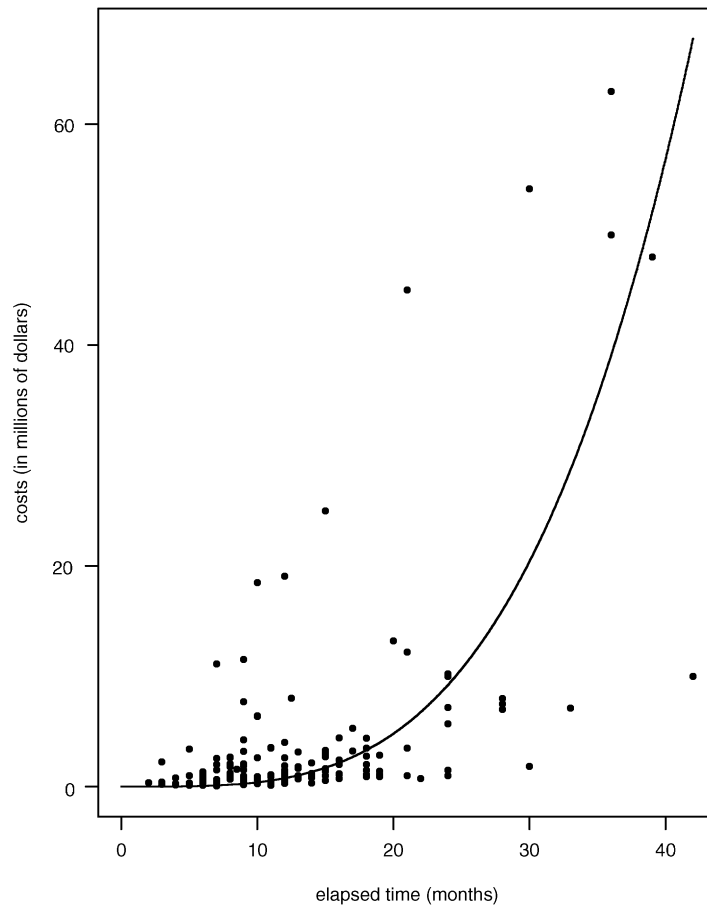


Fig. 2. Sample S from an IT portfolio database.

we did not use different rates for ERP, CRM, maintenance, off-shore outsourcing, and other deviating projects, to simplify explanations.

This figure represents about 200 projects, at a total cost of a little under a billion US dollar with an average project duration of about 18 months. This excerpt contains completed projects with actual financials, and proposed IT projects with estimated cost and duration. The curve in Fig. 2 is a plot of formula (1). There are a few outliers that remain after the error correction process. Further analysis of these outliers is necessary.

5.1. IT projects above the line

There can be several causes for outliers. One of the most frequent causes is that the development schedules are so crammed, that they approach the so-called impossible region [28,102]. This is sometimes due to a sudden business opportunity, a reaction

on a competitor that demands a full focus on speed to market, but can also be due to lack of governance by executive management. Whatever the reason: such projects have a high price tag, are risky, and too many of them can decrease IT performance considerably, leading to value destruction. The project costs and failure risks dramatically increase when the development schedule is compressed to the impossible region. Plotting formula (1) against the development projects in an IT portfolio gives you a quick overview of the outliers, that are probably death march projects [136]. For these projects you can analyze whether the need for speed was really that urgent, and if so, whether the incurred initial costs plus the high risk of failure justified the projects by a surefire business opportunity. For, when the costs balloon due to this type of development, the returns should be obvious, the pay back period should be relatively short, and performance should be measurable.

You will also run into high-risk low-reward projects in the portfolio: death march projects where the hurry is not justified by the potential returns. The two outliers around the 20 million dollar and the one of 25 million dollar were non-discretionary projects. Executive management does not consider these projects as strategic, since they must be done irrespective the strategy. But you can do them the smart and the stupid way. Usually, non-discretionary projects are initiated too late and therefore expose the enterprise to high costs and unnecessary risks. Better governance of these non-discretionary projects leads to better IT performance, by cost avoidance.

So also non-discretionary projects need timely executive attention. Especially, when they potentially affect large parts of the organization. Some frequent examples of unnecessary costly non-discretionary projects that we spotted in various IT portfolio studies are: operating system updates, platform migrations, Y2K projects, Euro conversions, programming language or dialect conversions, conversions to 10-digit bank account numbers (as required by the European Central Bank), and more. The nature of these non-discretionary projects is not special in the sense that they are of an extremely expensive nature, but they become unnecessarily costly by negligence. Non-discretionary projects should not be high-risk no-reward projects destroying value, but sometimes they are. For instance, Y2K costs consumed up to 30% of the total IT budgets in 1999 [125, p. 30]. For non-discretionary projects careful timely planning utilizing as much as automation is key, so that costs are brought to the absolute minimum. Total costs decrease when the schedules are relaxed, that is, when the projects are planned well-ahead of the deadline [102]. Moreover, the projects are done more rapidly and accurately when automated tools are used. Gartner Group advised to use so-called software renovation factories [130] when the code volume exceeds a few million lines of code [50,66] for Y2K updates. This advice remains valid for other modifications that potentially impact large parts of single systems or an unknown number of systems in an entire IT portfolio.

Apart from these unnecessary costs, there will always be death march projects that can be justified by proper business reasons. Executive management should consciously decide on acceptable risks, and put a threshold on death march projects for an IT portfolio. We recall that for this kind of analysis modern portfolio theory [79] is not adequate: you cannot abandon information technology like junking bonds. You have to live with a lot of suboptimal IT. Setting this threshold is just a way to keep the IT

spending within borders, not to select the optimal IT portfolio. One of the additional things that executive management can do is to rank death march projects, in potential added value for the business.

Note that an IT portfolio database not only contains completed projects, but also proposals for new projects and projects in progress. The management data provided by the IT-staff are estimates. Especially discretionary projects with a visionary scent can have gigantic budgets, while no indications on returns, net present value, pay back period, or risks are given. The most prominent examples we encountered were e-business initiatives, enterprise wide CRM or ERP implementations, corporate intranet projects, and complete system overhauls. Often, executive management committed themselves to these huge IT investments, without knowing what the consequences are for the IT budget in the coming years. Of course, you have to explore the new in order to innovate, and surely you will have to allocate costs for such endeavors, no matter how risky. Therefore, you need to set a threshold on such initiatives so that you are consciously in control of the costs, the risks, and the amount of potential loss. Depending on the type of the enterprise and the deepness of its pockets, executives can set thresholds to assure that the IT costs will not balloon so that the normal IT spending is in danger. Later on we will give an example on the consequences of large IT investments, and quantify the necessary minimal ROI and some important risks.

5.2. IT projects below the line

Also outliers in the low range are important to analyze in more detail. IT-staff is not good in estimating software costs [28,62] and severe underestimations are common practice [60] in level 1 organizations. A typical situation is illustrated in Fig. 2: as can be seen, a 33 month project proposal with an actual estimated budget of \$7.1 million dollar was approved. Inspection of the available project documentation showed that this was a cost reduction project (CR project) where savings of about \$12 million dollar in 5 years were projected. When you compare this to the expected development cost based on formula (1), you will find $tcd(33) = \$28.7$ million, which is about 75% more than estimated. It is not hard to realize that this project is going to cost money even when the cost savings are fully realized. In fact, implementation of the CR project will presumably cost more than it saves. It should not have been approved at all, and based on our analysis it was assessed internally and aborted. So our formulas can be used to calculate the exposure of an IT portfolio due to underbudgeting the proposed IT investments. We will deal with different kinds of IT portfolio exposure in more detail later on.

Executives should realize that a formal approval and cancellation policy for IT projects can be based on quantitative IT portfolio management. There is often no official cancellation process, and if it is in place, it is often not adequate, which can be measured by the number of restarts. This situation improves when approval and abortion are based on benchmarked thresholds so that erroneous cost saving operations can be routinely spotted, pruned when underway, and cancelled once and for all. Obviously, such preventive measures increase IT performance, by simple cost avoidance.

5.3. Synonyms, homonyms, redundancies

As soon as you compile an IT portfolio database, you will find multiple occurrences of the same project under a different name (*synonyms*) and multiple occurrences of the same name but describing different projects (*homonyms*). Synonyms can be removed from the database, and homonyms should be renamed so that you can differentiate between them.

In addition you will find existing and proposed systems that are redundant. Recall that in CIO Magazine, it is reported that just compiling an IT portfolio database saved one company \$3 million and another company \$4.5 million because the IT portfolio view enabled them to spot redundancies [5]. While we spotted redundancies as well, and could prevent unnecessary spending at times, you have to be careful with being too enthusiastic with removal of redundancies. There are two types of redundancies: similar proposed systems and similar existing systems.

First we deal with proposals. An often seen effect after an IT portfolio is compiled, is that similar proposals are put together and carried out as one combined joint project. Only if all the envisioned systems are going to be exactly the same it is likely to reap benefits from removing redundancies, by cancelling one but all the proposals. If another approach is taken towards redundancy, this leads to an increase in stakeholders, increase in necessary flexibility, more variation points, increased organizational complexity, multiple ownership adding to the complexity, increased feature creep, and a larger size of the new system. Larger size implies, more time to develop, more risk. Let us make a calculation to illustrate the possible consequences of removing seeming redundancies. Suppose you find two 12 month projects that are fairly similar. Using formula (1), each project will cost \$780 000 according to benchmark. So two of them, cost \$1.56 million. Using formula (3) we can calculate what the development time is of a single \$1.56 million project. This is 14.5 months. It is highly unlikely that the variation points, the increased number of stakeholders, and the other issues just mentioned will be solved in 2.5 months. So the synergy that looks good on management charts is very unlikely to be accomplished. Moreover this synergistic strategy easily leads to a *grand* IT project. Something that is strongly discouraged in the Clinger Cohen act [45], since this is known not to be working out properly. We quote:

Reduce risk and enhance manageability by discouraging “grand” information system projects, and encouraging incremental, phased approaches.

Now let us have a look at existing systems that seem redundant at first sight. While this can be extremely frustrating, it is not uncommon that an organization has multiple similar systems. You cannot get rid of them without significant investments, if at all. For instance, the US Department of Defense owns more than 700 payroll systems, over a 100 personnel training systems, and myriads of intranets [121]. How easy it is to remove such redundancies is also illustrated by the failure of GTE Corporation (then a leading telecommunications provider) to consolidate an overall medicare system. The US Secretary of Health and Human Services announced in 1994 that: “We’re going to move from the era of the quill pen to the era of the superelectronic highway”. This to provide timely payment, and reduce fraudulent claims. GTE spent \$100 million

for a unified medicare system and learned that they had to integrate many separate information systems. In 1997 the project was cancelled. According to GTE this project was far more complicated than anyone anticipated.

Solving this kind of problem is much more involved than removing redundant automobiles, buildings, or other tangible assets. The problem is that these systems look like redundancies from the executive viewpoint, but are more like homonyms. There are multiple variation points, and to consolidate those into one overall system takes sophisticated software engineering technology. You have to migrate the similar systems to a software product line [32]. CMM level 1 organizations are most likely not equipped to initiate, migrate, and deploy such complex software artifacts. Apart from that, redundancy in IT is not necessarily a bad thing. In [32] we call this the *relativistic effects of software engineering*, meaning that the classical way of thinking about software breaks down when the size increases, just like Newton's laws of physics break down when speed increases. For small systems, and small user communities redundancy can be avoided entirely. But as soon as variation points are necessary, the ideal situation is that changes only important for one user should not affect another user. If redundancy is weeded out completely, inevitably, users will have to accept new releases of the software also when the modification was not meant for them. The release effort for the business will then be higher than the cost of dealing with redundancy at the development site. See [32] for quantitative data supporting our arguments in detail. So redundancy is not necessarily a bad thing, and focusing on its removal should not be the prime task of an IPM.

While many business assets can be truly redundant, this is hardly ever the case for IT. In article [5] where the cost savings for redundancy are reported on, it is stated that IT is not special, that it operates like any other part of business. With the above arguments we have illustrated that IT *is* special. Apart from our arguments, the special status of IT has been subject to discussion for decades. For provocative enlightenment, we refer to a classical paper we mentioned earlier: the note by Lamport [73] explaining that maintaining software is entirely different from maintaining an automobile.

6. The black hole: hidden costs

Apart from the high costs of death march projects, high-risk visionary projects, and other costly efforts, there are also hidden costs that you need to be aware of. Let us continue with the CR project example to illustrate what we mean by this. Apart from the fact that the total development costs of the underbudgeted CR project are in the order of \$30 million according to benchmark, there will also be an additional *minimal cost of operation* during the entire deployment phase of this system, excluding separate enhancement projects. But how to quantify such costs? In this section we develop more formulas to calculate this according to benchmark. For instance, we can calculate with these formulas that for the coming 5 years the costs to keep the CR project operational are benchmarked at \$10.4 million. This alone almost annihilates the projected cost reduction of \$12 million in 5 years. Moreover there will be more operational costs if the system is not retired after 5 years of deployment. The *minimal total cost of*

ownership is around \$48.3 million according to benchmark, so even when the cost savings over the first 5 years will more than double over the rest of the deployment phase, to say \$30 million, there is a net loss of in the order of \$20 million over the entire lifetime of the system. So, even when the project was estimated correctly, the operational costs are high, and are prolonging the pay back period significantly, making the entire investment questionable.

It is our experience that the above calculations are not made when assessing IT proposals. However, they show that IT investments can easily lead to profit destruction instead of the expected value creation, not only by underbudgeting but also by not taking the operational costs over time into account. The accumulation of operational costs of IT projects plus the existence of profit destructing IT projects that were not anticipated is giving many executives the feeling that IT is a black hole. This feeling is supported by facts. Many large companies suffer from high operational and maintenance costs, 60% and more of the total IT budget is no exception. The US average in 1994 was that about 45% of the total budget was spent on maintenance [60]. This implies that almost half the annual IT budget was spent on operations and maintenance. These costs are increasing, given the fact that more and more IT personnel is working in maintenance [63,64]. These alarming figures are beginning to attract the attention of corporate management. The findings are no freak accidents, but have been reported over and over again for decades [34,8,26,77,106,6,51,63,102,81]. In those publications percentages between 50 and 80% devoted to maintenance costs are reported.

Also the cost per unit of work differs for development and deployment work on software. Already in the seventies this was empirically found. In [127] a ratio of 1:50 is reported, and in one US Air Force study Barry Boehm found that the cost per instruction was \$30 while the cost per instruction for maintenance was \$4000—a ratio of 1:133 [7]. For executives it is not clear what maintainers do, since the software was running in the first place to their understanding. Therefore, to obtain control over your IT portfolio it is crucial to know about these hidden costs. Only then it becomes possible to control your IT budget for the existing portfolio, and to project the operational costs for future systems that are proposed to be added to your IT portfolio. We will develop formulas that take operational costs into account.

6.1. Minimal total cost of ownership

Of course, it is hard to predict maintenance effort, since a lot of maintenance is not just keeping software running. It is more than that: enhancement to align systems better with new business needs. Those costs are often in your IT portfolio database, and are denoted as development projects (on existing systems). There are however substantial costs connected to the operational phase of software systems that are often not described in IT projects and therefore do not end up in your IT portfolio. For those hidden costs we will develop more formulas supporting quantitative IT portfolio management. Using the formulas it becomes much more clear what the real thresholds for ROI should be (we come back to this later on). In effect, we need to compare the potential value creation with the potential total cost of ownership in order to decide rationally on IT investments.

To be able to say something about total cost of ownership, we need to know how long the software will be operational. With the Y2K problem we have seen that this can be much longer than expected. Or as Strassmann put it [122, p. 253]:

Software is a new form of immortality.

In level 1 organizations where no measurement history is in place, there is also no lifetime data available, so we need to compensate for this lack of information by using a public benchmark taken from [60, p. 419]:

$$f^{1/4} = y,$$

where y is the number of calendar years the software will be deployed (f stands for function points again). From this benchmark, and the benchmark for MIS software ($f^{0.39} = d$), we can immediately derive two new QIPM formulas (9) and (10):

$$y(d) = d^{0.641}, \quad (9)$$

$$d(y) = y^{1.56}. \quad (10)$$

Note that for other industries, you have to adapt the schedule power using Table 3. For example, for a 34 month MIS development project we can calculate that according to benchmark this software will be deployed $y(34) = 9.6$ years, we used this formula in the previous section to see whether the cost reduction project would really save costs.

Formula (10), which is the dual of formula (9) can be used to estimate how much effort is reasonable for projects of which you know how long the software will be necessary. For instance, a company needed a conversion tool to automatically convert Cobol 85 back to an older dialect Cobol 74 [15] for the coming 37 months at most. This is a systems software project, and because of the implementation methods that are going to be used, it is acceptable to use the power for OO software (0.36 in Table 3). So, then the instantiation of formula (10) for the power 0.36 leads to: $d(y) = y^{1.44}$. Its development time should not be longer than $d(37/12) = 5.1$ months. We use the OO schedule power to instantiate formula (1) for this project, leading to: $ted(d) = rw/1800 \cdot d^{3.7778}$. We keep $r = \$1000$, and $w = 200$ days. We calculate that $ted(5.1) = 50\,800$ dollar to develop the tool seems reasonable, given the limited time you need the tool in the first place. If the costs are deviating considerably, it is time to assess the price setting of the tool supplier.

With formula (9) we can derive another formula that for a given project duration d measured in calendar months, gives us the minimal costs to keep the developed system operational. We will show later on that this is indeed a lower bound, so the actual costs could be higher, but presumably not lower than our formula calculates. We are interested in estimating the *minimal cost of operation* during the entire deployment time of a software system, based on the development time of the IT project. This is in fact:

$$mco(d) = y(d) \cdot wr \cdot nsm(d).$$

For, in one calendar year, you work w days at daily compensation rate r , with $nsm(d)$ people, for $y(d)$ calendar years. Combining formulas (9) and (6) then leads to formulas (11) and (12):

$$mco(d) = \frac{wr}{750} \cdot d^{3.205}, \quad (11)$$

$$dd(o) = \left(\frac{750}{wr} \cdot o \right)^{0.312}. \quad (12)$$

Again, dd stands for development duration. We used formula (11) for the earlier mentioned 33 month CR project: indeed the minimal cost of operation $mco(33) = 19.6$ M\$, the deployment phase is according to formula (9): $y(33) = 9.4$ years, so after 5 years, \$10.4M operational cost turns the estimated \$12 million savings into \$0.8 which is easily annihilated by the 75% underestimated cost of development.

Formula (12) is useful for rough estimates for merging, acquisition, and outsourcing. Let us give an example of the latter. You can use formula (12) as an indicator whether outsourced operational costs make sense at all. Often you know the total contractual operational costs o for a number of years to keep a system operational. Suppose there is a contract with an outsourcer to keep a system running for \$10 million a year for the coming 10 years. Then, using formula (12), we can calculate that the development time of this project was probably $dd(\$100M) = 54.8$ calendar months. And this implies that the deployment phase is approximately $y(54.8) = 13$ years. So depending on the actual development time (that could be present in your IT portfolio database) you can get an impression whether the outsourcer is too expensive, or if the software is really that large, you may want to rethink the contract to prolong it. The latter also depends on the added value for the business. The same type of calculation can be done when you want to acquire a company and you know total operational costs. Of course, this is an indicative estimate.

We can derive from formulas (1) and (11) the *minimal total cost of ownership*. Formula (13) supporting QIPM is

$$mtco(d) = tcd(d) + mco(d). \quad (13)$$

So, for a given project duration you can calculate the minimal total cost of ownership $mtco(d)$ of a system over the entire life cycle according to benchmark.

Let us give an example to show that the above formula makes sense. Suppose there is a project proposal that takes 36 calendar months. Total cost of development according to benchmark is $tcd(36) = 39.1$ million dollar. The minimal cost of operation according to benchmark is $mco(36) = 25.9$ million dollar. So, minimal total cost of ownership $mtco(36)$ of this software system is 65.0 million dollar. The first three years 39.1 M\$ is spent, and the subsequent $y(36) = 9.9$ years, 25.9 \$M/year is needed to keep it running. Indeed, 60.1% is spent on development, and 39.9% is needed to keep the system operational. Recall that our estimates use a minimal scenario: keep the systems running without large enhancements. So, our findings converge with the empirical data that are found for several decades [34,8,26,77,106,6,51,63,102,81].

We will derive these percentages for a given project duration. To that end we need the ratio between operational and development costs. Given formulas (1) and (11) formula (14) for quantitative IT portfolio management is

$$r(d) = \frac{12}{5} \cdot d^{-0.359}, \quad (14)$$

where $r(d)$ is the ratio of minimal cost of operation to development cost. Using this ratio we can easily infer two QIPM formulas (15) and (16) providing you with the development fraction, and the operational fraction of the minimal total cost of ownership:

$$df(d) = \frac{1}{1 + r(d)}, \quad (15)$$

$$of(d) = \frac{r(d)}{r(d) + 1}. \quad (16)$$

We plotted formulas (15) and (16) in Fig. 3. As can be seen, the development fraction will slightly increase for larger projects, while the operational fraction will decrease as slowly. This does not imply that operational costs will be smaller for larger projects. It just means that when the initial investment is larger that the operational costs are amortized over a longer period of time. Indeed if the project duration approaches infinity, the development fraction of the total costs will approach 1, and since an infinite length project will never finish, the operational fraction is indeed approaching to 0.

7. From project to portfolio level

At the executive level not only the accumulated minimal TCO per project and TCO of an entire portfolio plays a role, but also the dimension of time is of strategic importance. For instance, executives may want to know how much minimal operational IT costs were spent by a business unit in 1Q99, and how that compared to other business units. Or if they decide to invest next year in new systems, what will be the consequences for the coming years in terms of total costs? All this within the constraints of the current IT budget and the ongoing expenses necessary to keep the existing IT portfolio running. Maybe the plans will lead to unacceptable increases of the total IT budget. Also what-if scenarios can be supported by quantitative data: what if the new systems are introduced in phases? It is clear that for such projections we need to make the step from individual IT projects to entire software portfolios. Therefore, we should be able to superimpose IT project data. We do this by giving our formulas an absolute time dimension. Up till now we have discussed costs in relative time, that is, in terms of durations. The absolute time dimension enables sensible accumulation of IT project indicators. Then we can accumulate costs and benefits of IT projects in all phases: initial, development, deployment, enhancement, retirement phase, and so on.

Level 1 organizations do not deploy connections between projects. Think of reusing high-quality artifacts, such as a software architecture, or reusable requirements. So we

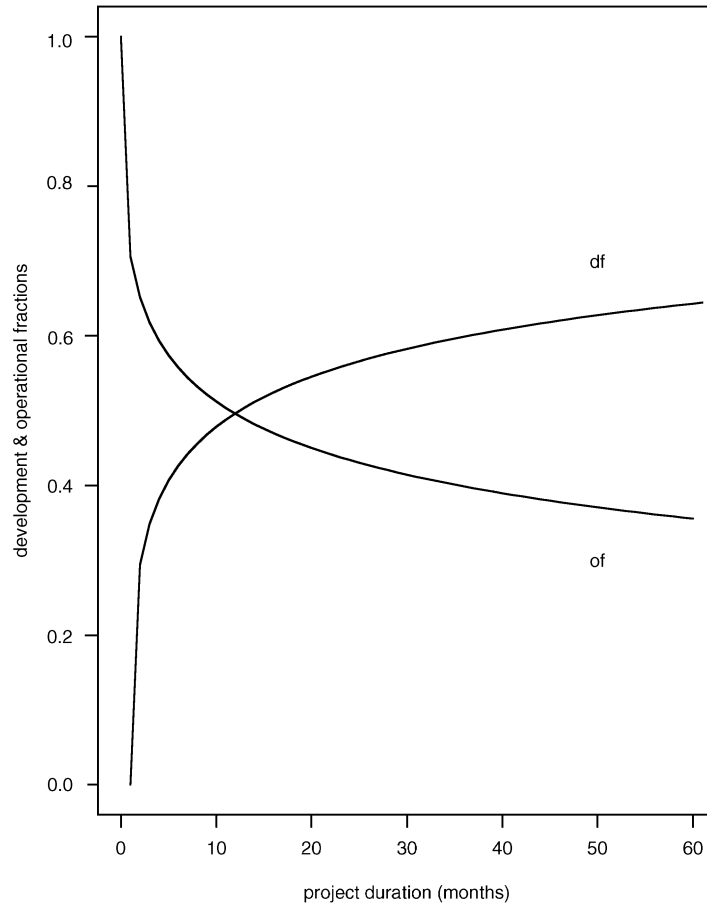


Fig. 3. Development and operational fractions as a function of project duration.

can treat information for each software project independently. This is not leading to an incorrect quantitative model: if there are connections, they are often ad hoc preventing significant cost savings over time as could be the case with software product lines. From the IT portfolio management level it looks like two independent projects, each with its own costs. We just need each project's start date and delivery date. Using the formulas we developed so far, we can easily infer additional formulas with an absolute time dimension. We focus now on formulas that for a given set of systems in the portfolio return the cost allocation at any given time.

7.1. Cost allocation formulas

We consider a software system s as a tuple in a database. In a level 1 organization a realistic assumption is to base yourself on two-dimensional tuples (i_s, d_s) where i_s is the initialization date of system s , and d_s the delivery date of system s . For the

sake of explanation, we abstract from the fact that also project names, organizational unit, and so on should be in the IT portfolio database. In case more information is around we should extend the tuple accordingly. For instance, if the actual financials for development are present this extends the tuple to a three dimensional one. In practical implementations of our formulas for quantitative portfolio management, this implies adding another column in a spread sheet, extending the schema of a database, or another column in a statistical package. For now we assume the minimal scenario and we use the two-dimensional tuple containing only the absolute data on initialization and delivery. The function that is relevant to lift from the project to the IT portfolio level is $ca_s(t)$: the cost allocation for system s at time t . We divide the cost allocation in two parts. Formula (17) for quantitative IT portfolio management is as follows:

$$ca_s(t) = \begin{cases} cad_s(t) + cao_s(t) & \text{if } i_s \leq t \leq r_s, \\ 0 & \text{otherwise.} \end{cases} \quad (17)$$

The number r_s is the retirement date of the system according to benchmark. The retirement date of a software project that started at i_s and was delivered to the business on d_s can be calculated using formula (9). The absolute-time pendant of formula (9) is formula (18):

$$r_s = d_s + 12y(d_s - i_s). \quad (18)$$

For example a software project that started in February 1993, and was delivered in August 1994 took 18 months and will retire presumably in $y(18) = 6.3$ years. So its retirement date r_s is expected to be around December 2000. Note that the $+$ in formula (18) stands for date addition, not addition of real numbers.

In formula (17) we used two other cost allocation functions. We depicted an example plot of formula (17) in Fig. 4. The abbreviation cad is the cost allocation for development, and cao is cost allocation for operation. In mature organizations these could be Rayleigh curves [6,102], but for the majority of the organizations this is wishful thinking. We provide you with the level 1 versions of these functions, but we stress that when you really have more sophisticated curves at your disposal, that the formulas we depict below can stay the same, except that the cost allocation over time is calculated by integration instead of using external benchmarks (we will discuss this issue later on). Formula (19) supporting quantitative IT portfolio management is

$$cad_s(t) = \begin{cases} \frac{tcd_s(d_s - i_s)}{d_s - i_s} & \text{if } i_s \leq t \leq d_s, \\ 0 & \text{otherwise,} \end{cases} \quad (19)$$

So we calculate the total cost of development for the duration of the project in calendar months, and divide that by its duration, leading to a constant monthly amount for the duration of the entire development effort. Before initialization and after delivery the cost allocation is zero. This implies that for a given system and time the above function returns the cost allocation in the month containing the time. This function is the first part of the plot displayed in Fig. 4.

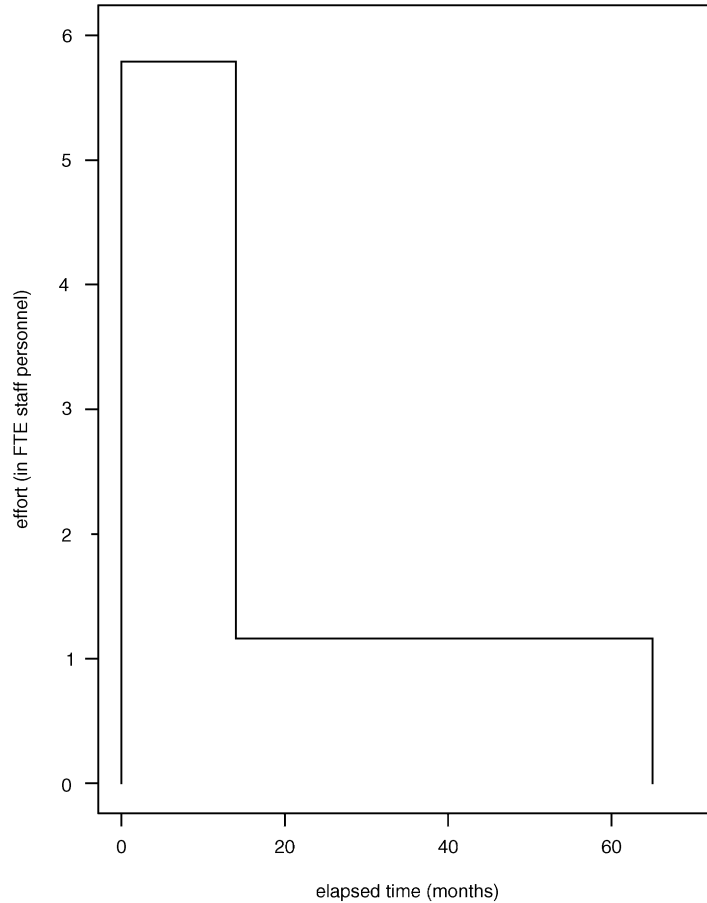


Fig. 4. Example plot of formula (17).

Similarly we provide such a formula for the cost allocation calculating the minimal operational costs (viz. Fig. 4). This is formula (20):

$$cao_s(t) = \begin{cases} \frac{mco_s(d_s - i_s)}{12 \cdot y(d_s - i_s)} & \text{if } d_s \leq t \leq r_s, \\ 0 & \text{otherwise.} \end{cases} \quad (20)$$

One of the ways to obtain insight in how you actually invest in IT is that you are able to monitor IT investments over time. This reveals trends in spending, maybe trends that you wanted to avoid if only you knew. We believe that the time dependency of IT cost allocation is crucial for executives in order to decide on strategic IT investments in a realistic manner. We can accumulate costs over time, which is done via integration. Formulas (21)–(23) for QIPM express the accumulated (minimal) total costs (*atc*), the accumulated development costs (*adc*), and the accumulated operational costs (*aoc*)

over a given time interval T for a certain software system s .

$$atc_s(T) = \int_{t \in T} ca_s(t) dt, \quad (21)$$

$$adc_s(T) = \int_{t \in T} cad_s(t) dt, \quad (22)$$

$$aoc_s(T) = \int_{t \in T} cao_s(t) dt, \quad (23)$$

where ca_s , cad_s , and cao_s are cost allocation functions. They can be the ones that we defined in formulas (17), (19) and (20), but they can also be more sophisticated formulas, like Rayleigh curves, or more sophisticated curves (we explain them later on). Now we can go from the project level to the portfolio level. This is done via summation, since there are only finitely many systems in a portfolio. For a given portfolio P and a given time interval T we can calculate accumulated (minimal) total costs for a portfolio, the accumulated development costs for a portfolio, and accumulated operational costs for a portfolio. Formulas (24)–(26) are:

$$atc_P(T) = \sum_{s \in P} atc_s(T), \quad (24)$$

$$adc_P(T) = \sum_{s \in P} adc_s(T), \quad (25)$$

$$aoc_P(T) = \sum_{s \in P} aoc_s(T). \quad (26)$$

We can use these formulas to answer the questions like the ones we posed earlier. How to compare operational costs of say 5 business units in the first quarter of 1999? For five business units BU_1, \dots, BU_5 you can calculate the total minimal operational IT cost for 1Q99 using the following simple formula:

$$\sum_{s \in BU_i} \int_{t \in 1Q99} cao_s(t) dt, \quad i = 1, \dots, 5.$$

We assume that for these business units we have their systems in the IT portfolio database, and that we have initialization and delivery dates of the projects. With this data, we can calculate for each system the operational cost allocations for the first quarter of 1999, and accumulate these figures for all the systems in the portfolio of a specific business unit. In this way you can compare operational cost per business unit and zoom in on differences, signaling trends that may need attention. For instance, if business unit 1 is now at CMM level 2, does this lead to lower operational costs? And if so, is it more than the costs of obtaining CMM level 2 and maintaining that level? And if so, how much could we save if we would do this for other business units as well? Note that it is not at all obvious whether operational costs will go down in case of better approaches to develop systems. The costs tend to become higher [27],

this is not too much of a problem if the systems create value. It can also be the case that one business unit is consuming the majority of the total operational budget in the corporate IT portfolio. Depending on the criticality and profitability of such a business unit, corporate executives can decide on different strategies: phasing out, selling, or reengineering and so on.

7.2. Hitting the innovation borders

Already in the early 1990s some fortune 500 companies found themselves trapped in the situation where their entire IT budget was gobbled up with updating, repairing and enhancing their aging legacy applications [60]. This does not need to be a dangerous situation if you reached exactly that level of automation that you wanted. But it is more likely that when you are in such a situation, you are exposed to unacceptable business risks. The environment changes fast and unpredictably, so to stay competitive, you have to innovate. This means that you must have the supporting budget. But since all new development adds to the operational pressure, you cannot innovate without limits. So, you need to keep track of how much of the IT budget is spent on operational costs at all times to know in advance whether operational costs start to hinder the amount of innovation that executive management deems appropriate for the company. This implies that to initiate new development you probably first need to retire existing systems, reduce operational costs, or increase the total IT budget. The minimal operational costs are significant as we already indicated with formula (11). We will now derive what this means in terms of absolute time.

We calculate the ratio between operational costs and development costs per unit of time. If you have sophisticated data available you can calculate this using actual values, but in level 1 organizations this information is usually absent. So we use our cost allocation formulas to calculate this ratio. Note that we in fact calculate the ratio between the heights of both rectangulars in Fig. 4. The height of the first rectangular is

$$\frac{tcd_s(d_s - i_s)}{d_s - i_s}$$

and the height of the second rectangular is

$$\frac{mco_s(d_s - i_s)}{12y(d_s - i_s)}.$$

Division yields

$$\begin{aligned} \frac{mco_s(d_s - i_s)}{tcd_s(d_s - i_s)} \cdot \frac{d_s - i_s}{12(d_s - i_s)^{0.641}} &= \frac{12}{5} \cdot \frac{(d_s - i_s)^{-0.359}}{12(d_s - i_s)^{0.359}} \\ &= \frac{1}{5}, \end{aligned}$$

using formula (14) for the relative ratio, and formula (9). This results in the fixed ratio equation (27):

$$cao_s : cad_s = 5 : 1. \quad (27)$$

Table 4
IT investment impulse

#	tcd	$dd(c)$	$y(d)$	r_s	cad	cao	$\sum cad$	$\sum cao$
50	15	27	99	126	0.6	0.11	27.5	5.49
10	30	33	113	146	0.9	0.18	9.0	1.81
6	75	43	133	176	1.8	0.35	10.5	2.10
3	150	52	151	203	2.9	0.58	8.6	1.73
2	300	63	171	234	4.8	0.95	9.5	1.90
1	600	77	188	260	7.8	1.56	7.8	1.56

So the operational cost allocation per time unit is 20% of the cost allocation per time unit for building the system. Or investing a dollar per time unit in IT development conservatively leads to 20 cents/time unit of operational costs for an extended period of time. This phenomenon is for many business executives counterintuitive: the operational costs of IT are much more significant than they expect from a delivered product to the business. So again we see now in absolute time the cost magnet in the IT budget that is attracting large amounts of hidden resources to keep the delivered systems operational.

7.3. Operational cost tsunamis

Let us give an example of the dynamics of such hidden costs over time. Suppose a corporation merges with other parties, and to consolidate the merge a lot of IT intensive projects have to be carried out, ranging from an enterprise wide CRM system, a few large ERP systems, several enterprise integration projects, HRM overhauls, e-business projects, Internet related projects, and a large number of relatively small ones. In Table 4 we summarized an impulse of \$3.15 billion divided over 72 projects of varying size. We call this IT portfolio M , short for *Merge*. If you think \$3.15 billion is exceptional, consider this quote about IT improvements that stems already from 1984 [53]:

Max Hopper, Armacost's technology expert, planned to announce that Bank-America would spend at least \$5 billion to improve its computer systems over the next few years.

But also the surveys on annual IT spending show that billion dollar investments are not uncommon. For instance, in 1998, the hundred top IT-spending European companies invested together 53.7 billion dollar, which is half a billion per company on the average [125]. Furthermore, the federal government of the United States of America plans to invest \$52 billion in 2003 [24].

The first column in Table 4 gives the number of projects, the second the sum of their estimated cost in millions of dollars. We used formula (3) to calculate their presumable cost of development, for the estimated development schedules. We calculated their deployment time, and the total life time (r_s). We calculated the cost allocation for development per month in millions of dollars using formula (19), and with Eq. (27) we calculated its operational cost per month according to benchmark. We accumulated

Table 5
Accumulation of cost allocation data

Development		Operations			
Time	Cost	Time	Cost	Time	Cost
27	72.9	27	5.49	127	14.6
33	45.4	33	7.3	146	9.1
43	36.4	43	9.4	176	7.1
52	25.9	52	11.1	203	5.2
63	17.3	63	13.0	234	3.5
77	7.8	77	14.6	271	1.6

these costs for all projects leading to the last 2 columns. We supposed that all 72 projects start shortly after the merge. Some of them are outsourced, others are done internally, the IT workforce is extended with myriads of people, and so on. It is not hard to calculate the accumulated costs for the entire portfolio for development and operations in absolute time as summarized in Table 5.

We visualized these data points in Fig. 5. As you can see there is a significant cost impulse in the first 25 months that rapidly declines after about 50 months.

Next, we use the accumulated data to derive the cost allocation function for IT portfolio M . The accumulated cost allocation function for development is the peaky one. We use standard parametric statistical techniques to infer a smooth curve from these data. Using an implementation of a nonlinear least squares regression algorithm [19,54,129,95] as implemented in the statistical system SPlus [129,72] the data points can be fitted to the following curve:

$$cad_M(t) = 7.514287t^{1.258007}e^{-0.07304098t}.$$

Recall that M is the IT portfolio consisting of the 72 projects in Table 4. Before we continue, a few words on the large amount of digits in the above formula. The data has a certain precision, of course, but we try to fit this data as good as possible, which leads to the large amounts of digits. If we would round the above digits, an entirely different curve would show up. This is partly due to the exponential functions: a slight change in its power is a huge change in the behavior of the curve. So we keep these very precise, so that we will not deviate from the input data. Second, the outcomes of using the formulas, are subject to the standard rounding rules. However, we will use high precision outcomes so that readers who redo the calculations can convince themselves that they made the right calculation. Of course, in practical applications you have to round output of such formulas in accordance with the precision of the input data (but not the used coefficients in those formulas).

The other curve represents operational costs. It is a curve with a long wave length and a 100 month lasting cost plateau. This plateau starts right after the IT investment impulse is over. Also the accumulated operational cost allocation function can be fitted to a curve:

$$cao_M(t) = 0.02643959t^{1.692713}e^{-0.003407055t^{1.317413}}.$$

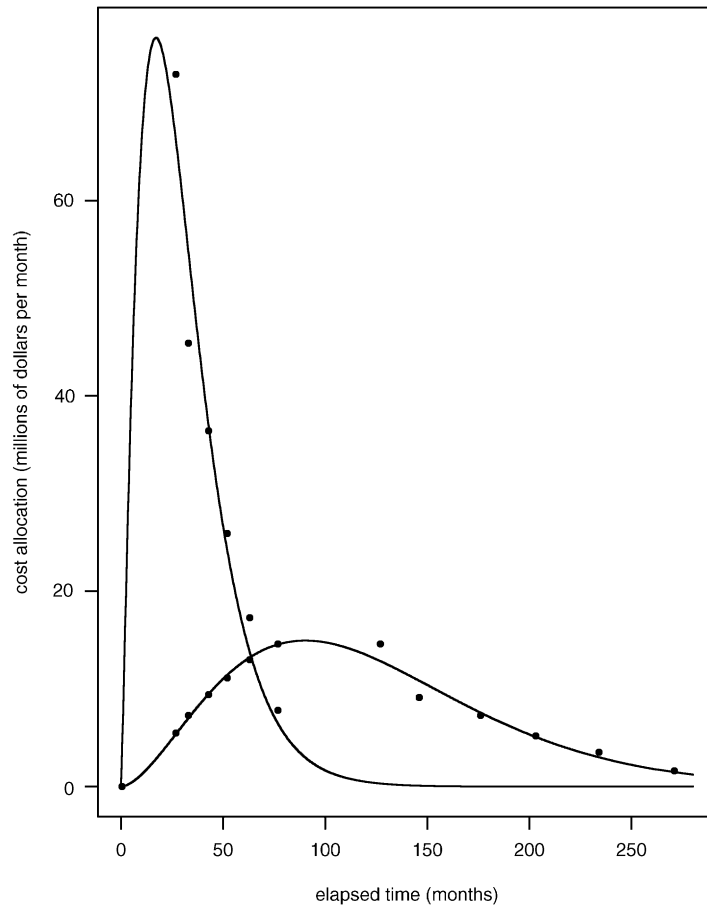


Fig. 5. Seismic IT costs induce an operational cost tsunami.

When the IT portfolio M is turned into reality, and most of the systems have become operational, the IT investment should start to generate added value. But now the operational costs start to rise. They can easily annihilate returns, since the operational costs represent a long lasting significant expense. We call a sudden investment a *seismic IT investment*, since it causes an *operational cost tsunami*, just like geographic seismic events can cause tsunamis (a great sea wave produced by submarine earth movement or volcanic eruption). Operational cost tsunamis are often responsible for the black hole of IT that many executives experience, but cannot reveal. These hidden costs can significantly influence the pace of new development.

By quantitative IT portfolio management you can reveal existing operational tidal waves, but also prevent new tsunamis, by astutely timing the rate of innovation. This implies that it is useful to analyze IT investments over a long period of time to uncover cost waves that are still dominating your IT budget. If you look at Fig. 5 again, you can

see that many years after implementation of the \$3.15 billion IT portfolio, significant operational costs of the seismic IT investment are still influencing the IT costs of the business unit owning the portfolio.

With the accumulated cost allocation functions we have a potentially powerful weapon to forecast future costs. First we calculate the accumulated development cost function for the entire portfolio. We use formulas (25) and (22) for that:

$$\begin{aligned} adc_M(T) &= \sum_{s \in M} adc_s(T) = \sum_{s \in M} \int_{t \in T} cad_s(t) dt \\ &= \int_{t \in T} \sum_{s \in M} cad_s(t) dt = \int_{t \in T} cad_M(t) dt \\ &= \int_{t \in T} 7.514287 t^{1.258007} e^{-0.07304098t} dt. \end{aligned}$$

We take the interval $T = [0, t]$:

$$\begin{aligned} adc_M(T) &= \int_0^t 7.514287 t^{1.258007} e^{-0.07304098t} dt \\ &= 7.514287(419.087 - 368.192\Gamma(2.258007, 0.07304098t)). \end{aligned}$$

You can use a computer algebra system like Maple [135] or Mathematica [132] for this evaluation (but we used formula (39) that we discuss later on). The function Γ is a special mathematical function called the upper incomplete gamma function. This function satisfies the following equation:

$$\Gamma(a, x) = \int_x^\infty t^{a-1} e^{-t} dt.$$

The accumulated development cost function approximates the total IT investment accurately: if we take $T = [0, \infty)$ we should get the total development budget of the portfolio back. Indeed, $\lim_{x \rightarrow \infty} \Gamma(a, x) = 0$, so

$$adc_M(0, \infty) = 7.514287 \cdot 419.087 = 3149.14$$

million dollars. This outcome has a 0.02% difference with the actual investment of \$3.15 billion. For the total minimal cost of operation we cannot do such a “regression test”, since those costs were never envisioned in the first place. But let’s calculate the accumulated operational cost for M as well. We used formula (39) that we discuss later on, but you can also use a computer algebra system like Mathematica [132] or Maple [135] for this.

$$\begin{aligned} aoc_M(0, t) &= \int_0^t 0.02643959 t^{1.692713} e^{-0.003407055t^{1.317413}} dt \\ &= 2262.23 - 2219.22\Gamma(2.043939903, 0.003407055t^{1.317413}), \end{aligned}$$

so now we can see the total impact of operational costs for this portfolio over the entire life cycle of the portfolio. We find $aoc_M(0, \infty) = 2262.23$ million dollar. The total cost of ownership of this portfolio thus amounts to 5411.37 million dollar. So, 58.2% of the costs are devoted to development, and 41.8% is necessary for operations, which is in accord with the many empirical findings we quoted earlier. Note that this IT investment impulse is now \$2.3 billion short. Not all this money needs to be present from the start, but should become available sometime in the future.

When does this future start? When the initial IT investment is fully consumed by implementing it. We can calculate when this is the case. We know that the accumulated total cost allocation for M is as follows:

$$\begin{aligned} atc_M(0, t) &= adc_M(0, t) + aoc_M(0, t) \\ &= 5411.37 \\ &\quad - 2219.22\Gamma(2.043939903, 0.003407055t^{1.317413}) \\ &\quad - 2766.70\Gamma(2.258007, 0.07304098t). \end{aligned}$$

We calculated this formula simply by adding the above two derived formulas. We plot the three accumulated cost functions in Fig. 6. This is an insightful graph giving you an indication of the probable spending situation over the forthcoming years. We can already see that somewhere between month 50 and 60 the money will probably run out. We use this rough estimate as an initial value for root finding. With the computer algebra system Mathematica [132] we solved the root of the following equation (but we could have used Maple [135] or Matlab [80] as well):

$$atc_M(0, t) - 3150 = 0,$$

yielding $t = 57.3123$ months. So, the money runs out after 57 months. After that time stamp we really need a positive return from the IT investment to subsidize the missing 2.3 billion. Not immediately, but in due time. When these returns should become available is our next subject.

7.4. ROI threshold quavering

After about 50 months, most of the systems in the example IT portfolio have become operational, that is, when $atc_M(50) = \$2900$ million is spent. The next hundred months, we need another

$$atc_M(150) - atc_M(50) = 1838.87$$

million dollar, to finalize development and keep the portfolio running. Suppose that the investment plan for our example IT portfolio projected an annual return of 10%, starting after 50 months (which is more than 4 years), then in the first year after these 50 months the portfolio should add a value of 315 million. However, you have to

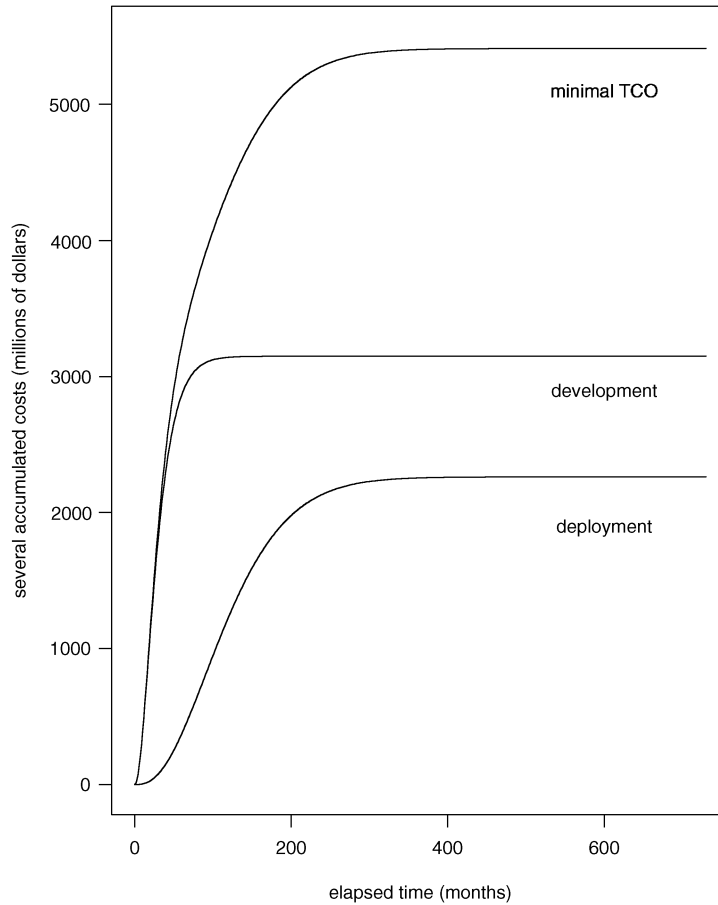


Fig. 6. Accumulated minimal TCO over time.

spend in that year

$$atc_M(62) - atc_M(50) = 387.077$$

million dollar on the IT portfolio as well. So you will make a net loss of 72 million if you just set the ROI threshold on 10%. When we take all the costs into account we get a different picture for the ROI threshold. We calculate the actual minimal ROI threshold, abbreviated *mrt*, that you need in order to achieve a net 10% ROI at all.

The first 50 months is the investment period: no ROI is expected. From that moment on a net 10% ROI annually over the entire investment of 3.15 billion is projected. This amounts to 26.25 million dollar per month. The 3.15 billion is spent at time stamp 57.3, so until that time the ROI does not need to compensate for IT portfolio costs. But after that time stamp, the ROI should pay for the ongoing costs in addition to the 10% bottom line. The minimal ROI threshold of our example portfolio *M* is

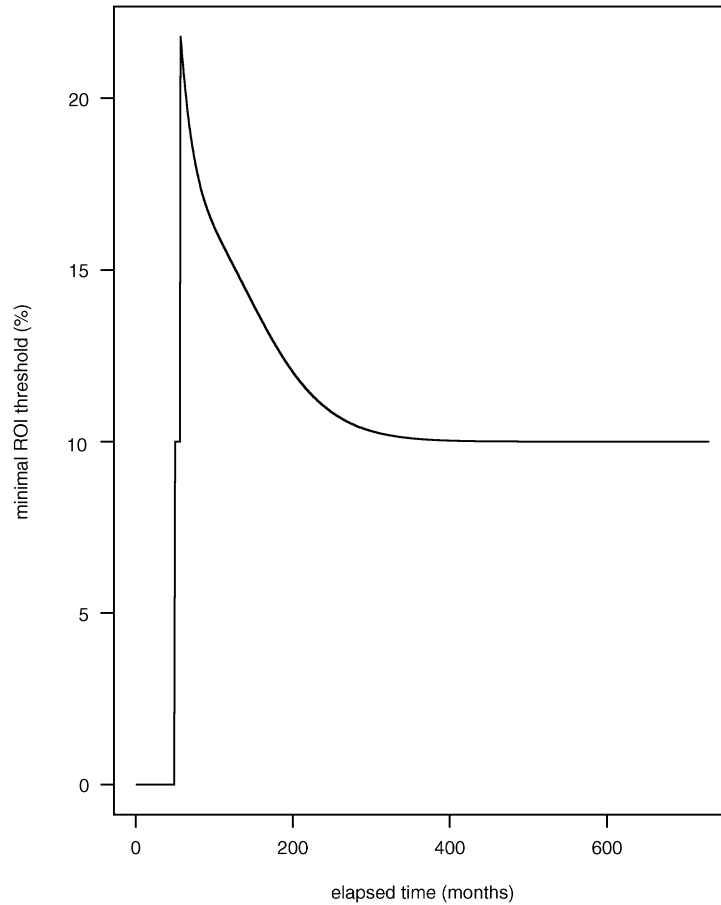


Fig. 7. Minimal ROI threshold over time.

as follows:

$$mrt_M(t) = \begin{cases} 0 & \text{if } 0 \leq t \leq 50, \\ 10 & \text{if } 50 \leq t \leq 57.3, \\ 10 + \frac{ca_M(t)}{2.625} & \text{if } t \geq 57.3. \end{cases}$$

We plotted the minimal ROI threshold in Fig. 7. We call this curve an ROI threshold *quaver* after the shape of an eight note in music notation; we sometimes use the (somewhat awkward) term *iso-net-ROI* line. As soon as the IT investment budget is consumed, you need a return of about twice as much to achieve the 10% bottom line. This is not a short term tremble in the necessary ROI, but a long lasting one. Only decades after the IT investment impulse, the quaver-shaped curve approaches 10%. Obviously, when you do not take hidden costs into account, it is very likely that you

will never have a positive return on investment, if the actual profits of the IT investment cannot compensate the ROI quaver.

8. IT portfolio exposure

Apart from the fact that IT spending is much more costly than many of us envision and that projected returns are not always met (e.g., if ROI threshold quavering is not taken into account) there is also another dimension to quantitative IT portfolio management, that can seriously hinder achieving added value: IT risks. McFarlan [82], proposed already in 1981 that risks of IT projects should be assessed not only separately, but also in the aggregate—as a portfolio. While McFarlan proposed a risk assessment questionnaire, we quantify risk based on benchmark data. We believe that risk assessment will benefit from a combination of quantitative and qualitative information.

In the introduction we already indicated that the risks of software projects are high. As Standish group found, 50% is challenged, and 30% of the IT projects fail. But, can you project these numbers directly on your own IT portfolio? The answer is no. But executives need to get an indication of the ratios of successful, challenged and failed projects. Since there is no historical data on such topics in CMM level 1 organizations, and understandably, this information is usually hidden for executives, we need to compensate for this by using public benchmarks. In this section we show how you can obtain an indication of the IT portfolio exposure based on project durations. This is not necessarily the ideal method to quantify risk, but usually it is the only way for level 1 organizations to get an indication at all.

Adding to the complexity of risks is that some executives think that you can just overhaul IT systems, this is not true. However, it is not a surprise that people think like this: our calculations with respect to IT portfolios revealed that even in the unlikely case that your systems do not need enhancements, the costs to keep them operational are huge. Moreover, enhancing these systems makes things worse: the costs increase. So it makes sense to ask yourself, why not renew these costly systems?

Once these often costly systems are up and running their failure exposure is lower than in a new situation, where childhood diseases, and infant mortality are not uncommon [58,46–48,42,41,44]. You could see a deployed system as a set of executable requirements needing continuous debugging, refinement, and extension. While this can be a frustrating task, cherishing existing business-critical systems is often paying off much more than overhauling these systems by new ones. Strassmann noticed this as well, given that he writes [122, pp. 258, 257]:

For an enterprise with a large accumulation of legacy systems—which includes all established organizations—there are no technical strategies other than evolutionary migration strategies. Defining the path of such migration requires placing limited objectives along the way. The managerial skill in coming up with such a plan and then making it happen will be the ultimate test which only superior information management teams will pass. [...] In the future, information political contests will

Table 6
Information system schedule adherence (1999)

Size (FP)	Early projects (%)	On-time projects (%)	Late projects (%)	Cancelled projects (%)
1	6.00	92.00	1.00	1.00
10	8.00	89.00	2.00	1.00
100	7.00	80.00	8.00	5.00
1 000	6.00	60.00	17.00	17.00
10 000	3.00	23.00	35.00	39.00
100 000	1.00	15.00	36.00	48.00
Average	5.17%	59.83%	16.50%	18.50%

be fought over issues that concern managing software assets. [...] Whoever accepts that conservation of software assets is now the key to all information politics will end up as a leader.

Indeed the software assets of an enterprise may have their moments, but the bottom line is that they are relatively mature, and often the cash cows of the company. Scrupulous quantitative IT portfolio management, containing calculations like the ones we have shown thus far will support you in obtaining the appropriate justifications for investing, or disinvesting in such existing assets.

8.1. Failure rates for IT projects

Benchmark data indicates a very strong correlation between the size of software and the chance of failure. This relation is also strong when a project is challenged, meaning huge cost and effort overruns, while much less than the originally requested functionality is delivered. Based on public benchmark data we inferred simple formulas indicating risks. Like the other formulas, you should not use them to base individual IT project contracts on, but again they are an excellent means to get an idea of IT portfolio exposure. Table 6 summarizes schedule adherence benchmark data for information system projects [65, p. 192].

Based on these benchmarks we fit a curve that can be used to quantify the risk of failure as a function of the project duration. The six observations above are based on many projects, so we consider this a strong benchmark. We assume that IT project failure grows logistically with increasing size. A statistical fit based on the observations gives us formula (28). It is the *chance of failure* for a given information system project given its size in function points (we will use the subscript i to indicate information systems industry):

$$cf_i(f) = 0.4805538 \cdot (1 - \exp(-0.007488905 \cdot f^{0.587375})) \quad (28)$$

We note that formula (28) cannot be used for systems above 100 000 function points: the asymptotic behavior of formula (28) is that the chance of failure approaches 50%

for large sizes whereas we believe that when the size of software reaches infinity, the chance of failure goes to one. However, for a pragmatic indication for the majority of the projects in your IT portfolio, formula (28) can be used. We could have fitted a curve with more appropriate asymptotic behavior. But such a curve is less accurate below 100 000 function points. There are two reasons for not using this alternative: firstly, the majority of the systems is in the range that formula (28) covers, and secondly, for systems that exceed 100 000 function points, we recommend to do a full function point analysis.

Recall that in CMM level 1 organizations you usually only have elapsed time and not the function point size, so we have to make another calculation using the benchmark taken from [64]: $f^{0.39} = d$. This leads to formula (29).

$$cf_i(d) = 0.4805538 \cdot (1 - \exp(-0.007488905 \cdot d^{1.506090})). \quad (29)$$

As an example, the risk of failure for a 36 month MIS project is $cf_i(36) = 0.39$. So 39% according to benchmark. In Fig. 8 we plotted formula (29) to indicate that the chance of failure increases rapidly for longer project durations.

In the MIS industry it is customary to outsource certain parts of an IT portfolio. For instance, 43% of all the outsourcers is working on MIS software [65, p. 264]. To that end it is useful to make comparisons with respect to cost and risk (we elaborate on make-commission decisions later on). We derive outsource software risk formulas by using available public benchmarks. Table 7 summarizes schedule adherence benchmark data for outsourced software projects [65, p. 275].

Similarly to the derivation of formula (28), we carried out a statistical fit based on the observations summarized in Table 7. This leads to formula (30) for quantitative IT portfolio management. It is the *chance of failure* for a given outsourced project as a function of its size in function points (the subscript o refers to the outsource industry):

$$cf_o(f) = 0.3300779 \cdot (1 - \exp(-0.003296665 \cdot f^{0.6784296})). \quad (30)$$

If outsourcers use the function point metric, you can use formula (30). If they do not, you can use the productivity benchmark taken from [64] for outsourcers (that we tabulated in Table 3): $f^{0.38} = d$. This leads to formula (31).

$$cf_o(d) = 0.3300779 \cdot (1 - \exp(-0.003296665 \cdot d^{1.7853411})). \quad (31)$$

8.2. Challenge rates for IT projects

It is convenient to have formulas indicating the chance of late projects. Similarly to the failure rate formulas, we can easily infer a curve using the benchmark for late projects depicted in Table 6. Again we assume that the chance of late projects grows logistically with the size of IT systems. This leads to formula (32) expressing the chance of late projects in the information systems industry for a given function

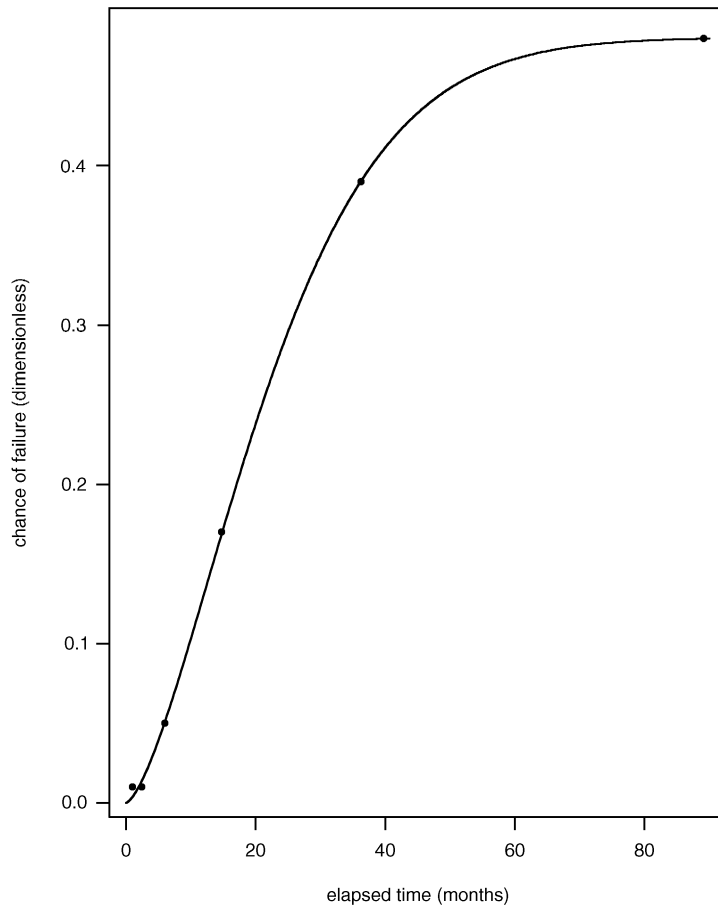


Fig. 8. Chance of failure as a function of project duration.

Table 7
Outsource software schedule adherence (1999)

Size (FP)	Early projects (%)	On-time projects (%)	Late projects (%)	Cancelled projects (%)
1	5.00	93.00	1.00	1.00
10	8.00	90.00	1.00	1.00
100	7.00	85.00	6.00	2.00
1 000	8.00	67.00	15.00	10.00
10 000	1.00	38.00	34.00	27.00
100 000	1.00	26.00	40.00	33.00
Average	5.00%	66.50%	16.17%	12.33%

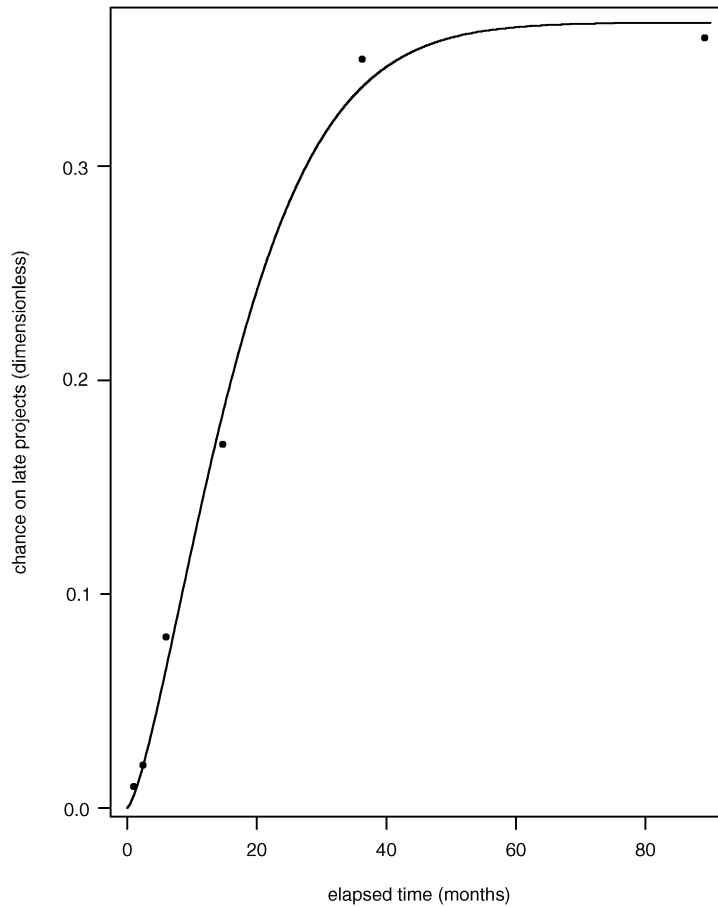


Fig. 9. Chance on late projects as a function of project duration.

point size.

$$cl_i(f) = 0.3672107 \cdot (1 - \exp(-0.01527202 \cdot f^{0.5535625})). \quad (32)$$

The instantiation for the MIS industry using benchmark $f^{0.39} = d$ leads to formula (33).

$$cl_i(d) = 0.3672107 \cdot (1 - \exp(-0.01527202 \cdot d^{1.4193910})). \quad (33)$$

So, the risk on cost overruns for a 36 month MIS project is $cl_i(36) = 0.31$. According to benchmark there is a 31% chance that this project will suffer from serious cost overruns. In Fig. 9 we plotted formula (33).

Likewise, we can do the same for outsource software. Using the data of Table 7 we easily find formula (34) for quantitative IT portfolio management:

$$cl_o(f) = 0.4018422 \cdot (1 - \exp(-0.009922029 \cdot f^{0.5657454})). \quad (34)$$

If the outsourcers work with function points, you can use formula (34) immediately, and if not, you can use formula (35):

$$cl_o(d) = 0.4018422 \cdot (1 - \exp(-0.009922029 \cdot d^{1.488804})). \quad (35)$$

where we used the schedule power 0.38 as listed in Table 3.

8.3. Challenge and failure rates for portfolios

Knowing how to calculate prominent exposures on a per project basis, we can make the step from individual projects to the portfolio level. We accumulate the project exposures to obtain the portfolio exposure. You can then answer questions like: which business unit has the highest exposure to failed projects? For a given portfolio P the *failure exposure* of P is formula (36):

$$fe(P) = \frac{1}{|P|} \cdot \sum_{s \in P} cf(d_s), \quad (36)$$

where $|P|$ is the number of systems in the portfolio, s is a system, d_s is its project duration, and cf is some derived chance of failure function, for instance formula (29) or (31). In this way you can calculate the average failure rate of the entire IT portfolio. It is up to executive management to set a threshold on the overall failure exposure of an IT portfolio.

Similarly, for a given portfolio P the *late exposure* of P is formula (37):

$$le(P) = \frac{1}{|P|} \cdot \sum_{s \in P} cl(d_s) \quad (37)$$

For instance, for the sample portfolio that we depicted in Fig. 2 we can calculate both exposures: $fe(S) = 0.13$ and $le(S) = 0.14$. The sample portfolio has a chance of failure of 13%, and a 14% chance of serious cost overruns. For our seismic IT impulse depicted in Table 4 we can likewise calculate that $fe(M) = 0.126$ and $le(M) = 0.135$. The percentages that Standish Group found, (30% cancelled, 50% challenged, 20% okay), are not found in many business unit level IT portfolios. This is due to the fact that per portfolio, only a fairly small number of very large projects are present. At the corporate level the percentages can be a bit higher, but still not approaching the Standish Group findings so closely, that you can use their benchmark to calculate your IT portfolio exposures. This is due to the fact that large companies often have a lot of smaller business units. When only very large business units are present, also larger projects are undertaken, with their risks. The Standish Group findings are accumulated at the country level (for the USA). Maybe only the largest projects in the surveyed companies were taken into account.

Depending on the nature of the company and the deepness of its pockets, such IT portfolio exposures give you an indication whether you are within the exposure zone that you consider acceptable. If not, it is time to mitigate those risks, and identify the carriers of large exposures; they are almost always large IT projects (as McFarlan also pointed out [82]).

9. Comparison with higher CMM levels

A natural question is whether the accuracy of our approach explained so far will drastically increase when the underlying mathematics is not based on level 1 formulas, but on formulas available to organizations with CMM levels that are higher than 1. It is not easy to make comparisons, since there are hardly any published cost allocation curves (which might be due to the fact that 75% of the organizations are at CMM level 1). Apart from that, many cost estimation techniques were traditionally based on lines of code, instead of function point-like metrics, such as the various versions of function points [1,61], or Tom DeMarco’s bang metric [28]. It is known from the software productivity literature that different definitions for lines of code can lead to an uncertainty of 500%, rendering *comparisons* of different estimates based on lines of code often useless [59, p. 16]. Recall that in [33, p. 132] even a 2300% variance was found for different definitions for lines of code. Therefore, it is not a surprise that in a review article on software cost estimation techniques [84], huge differences were found when about 15 cost estimation techniques were applied to a single project.

Nevertheless we found an example curve in a textbook on software cost estimation. With this published example we illustrate that the results might not lead to radically different decision making than in the level 1 situation. One argument why this is the case, is that we are not using the actual relations between cost, effort, and duration over time, but rather their mean values, expressed by the area under cost allocation curves. If the areas are of the same order of magnitude, all calculations based on the areas under these curves will be of the same order of magnitude as well, and therefore the decision making will not drastically change. Of course, CMM level 2+ organizations have historical data, which enables the derivation of internal benchmarks. They are more precise than the external benchmarks that we use now. So the quality of the decisions will improve, based on the input data that you can instantiate our formulas with, but our method can still be used.

We give an example supporting the fact that more involved cost estimation formulas usually do not change the outcome of IT portfolio decision making. Note that in general, it is a good idea to estimate software costs as accurately as possible.

In Boehm’s book on software engineering economics [6, p. 68] a Du Bridge Chemical software development project is used as a running example. Its distribution is as follows.

$$ead(t) = \frac{mt}{p^2} \cdot \exp - \left(\frac{t^2}{2p^2} \right).$$

The above function is called a Rayleigh curve; *ead* is the effort allocation for development, *m* is again short for man-months, and *p* represents the month at which the project achieves its peak effort. For the Du Bridge Chemical project, Boehm used the following data points: *m* = 91, and *p* = 7 months. Boehm also gives a rule of thumb for *p*: it is half the estimated development time. This implies that the total effort for 14 months is the area under the curve plotted in Fig. 10. We integrate over the effort

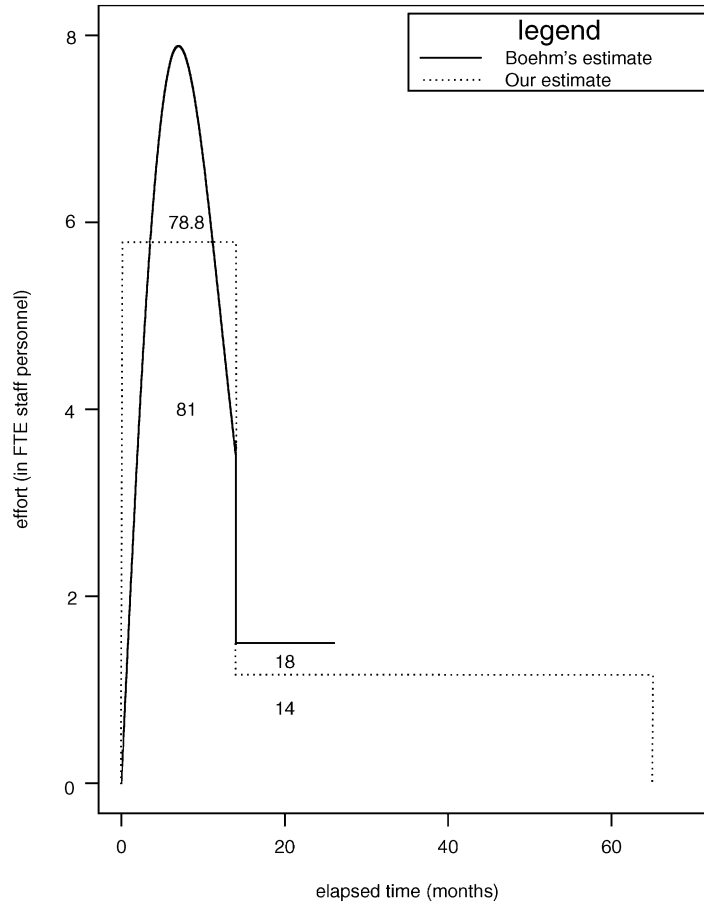


Fig. 10. Rayleigh effort curve and our cost allocation formulas.

allocation formula and find:

$$\int_0^{14} ead(t) dt = 78.7.$$

You can use calculus, a scientific calculator, Matlab [80], Maple [135], or Mathematica [132] to calculate the integral (but we used formula (39) that we discuss later on). After the 14 months, Boehm applies another model. Boehm calculated the effort allocation for the first year of maintenance. He did not use a Rayleigh curve for it, but used a fraction of the total development effort that is exactly the same as our inferred ratio in Eq. (27). However, his ratio was not inferred from general arguments like our ratio, but calculated using actual data. He calculated the so-called *annual change traffic*: he measured the exact amounts of added instructions and modified instructions (but deleted code was not taken into account). Based on that a fraction of m was found: yielding in the first year of maintenance $0.20 \cdot m = 18$ months. In Fig. 10 this

is expressed by the horizontal line at a height of 1.5 from the 14th to the 26th month. So the total effort for the first 26 months of the project according to Boehm is 96.7 man-months.

Let us compare this to our calculations for the level 1 case. The described project is an MIS project: it is a raw material inventory project. So we can use formula (5), which is instantiated with an MIS schedule power. The number of staff necessary for development is $nsd(14) = 5.8$. Since it is a 14 month project, $m = 81$ man-months. We plotted our effort allocation function in Fig. 10 with a dashed line. Using formula (6), we can calculate the required staff for maintenance: $nsm(14) = 1.16$. This is the lower horizontal line that lasts for the entire life cycle of the system.

To compare our calculation to Boehm's work, we take only the first year of maintenance: $m = 14$ months. So the total effort for development plus the first year of operation is 95 man-months. This is less than 2% difference with Boehm's method. His method is clearly meant for CMM levels higher than 1. For, it is not feasible for a level 1 organization to measure the correct staff increase and decrease over time, the peak effort allocation, the number of added instructions, deleted instructions, modified instructions, and so on.

From one example you cannot draw far reaching conclusions, but our approach makes sense. Let's have a second look at the general Rayleigh curve from Boehm's book. The area under a Rayleigh curve [102, p. 46] is exactly the total number of man-months (we use formula (39) for this):

$$\int_0^{\infty} ead(t) dt = \int_0^{\infty} \frac{mt}{p^2} \cdot e^{(-t^2/2p^2)} dt$$

$$= m.$$

In our formulas, we abstract immediately from the staff variations, and treat the number of man-months for development as a constant over time. So the above derivation shows that in general CMM level 2+ organizations will have more accurate data over shorter time frames: with the Rayleigh curve you can predict for each moment in time, the exact effort. While in our case, we use the average from the start, which coincides with the area under the Rayleigh curve. The benefit of having Rayleigh curves is that you then find a better average: not based on an external benchmark, but on actuals.

10. Towards full transparency

When you have the accumulated cost allocation curves for development and deployment both for groups of projects and individual projects, you know the *how*, the *when*, and the *how many* of your IT-dollar expenditure. This adds to the badly needed transparency in IT performance and investments. In the previous section we have seen a few such curves already: a seismic IT impulse and an operational cost tsunami at the portfolio level. In all portfolios we analyzed, we encountered these two extremes plus curves in between these extremes.

10.1. Cost allocation equation

We note that our experience is limited to large companies: it may be the case that different models are necessary in other cases. Based on our experience, we found that IT portfolio costs over time can be accurately approximated by the following cost-time function $c(t)$ returning for a given time t the corresponding cost. We conjecture that this will also hold for IT portfolios that we have not assessed. Formula (38) for QIPM is

$$c(t) = a \cdot t^\alpha \cdot e^{-b \cdot t^\beta}. \quad (38)$$

In our formula a, α, b, β are constants idiosyncratic for the environment in which the work is carried out. Useful relations for the coefficients can be inferred, as we will see later on. Cost could be seen either as effort, its corresponding financial remuneration, or another cost dimension (see Fig. 11 for several plots of formula (38)).

It is already known for a long time that for $\alpha=0$ and $\beta=2$ the above equation is used to estimate effort allocation for research and development projects as shown by Norden [86–88]. Recall that for these α and β our cost allocation equation reduces to the Rayleigh curve. After Norden, Putnam applied Rayleigh curves to software projects [100,103,101]. Also Boehm based his COCOMO model in part on Rayleigh curves. However, he noted that [6, p. 68]:

It is evident that the shape of the Rayleigh distribution in Fig. 5–5 is not a close approximation to the shape of the labor distribution curves for any of the organic-mode software projects shown in Fig. 5–4. This is largely because an organic-mode software project generally starts with a good many of the project members at work right away, instead of the slower buildup indicated by the Rayleigh distribution. However, the central portion of the Rayleigh distribution provides a good approximation to the labor curves of organic-mode software projects.

Boehm thought that Rayleigh curves were not in accord with the actual cost allocation of a certain type of project. So, Boehm used Rayleigh curves only around the peak effort p : between $0.3p$ and $1.7p$. He could have used the so-called decentralized Rayleigh curve. If you need non-zero man power at the start of the project, you should use an additional location parameter, to shift the Rayleigh curve to the left (use $t - \mu$ instead of t). It is not necessary to invent another distribution for that. So by removing such a constraint it is possible to approximate reality much better. We did not display a decentralized version of our cost allocation function, but when we need it, this does not add any difficulty to using our results. Parr invented an alternative distribution for the same reason: a non-zero man power at the start of the project [90]. Again, a decentralized Rayleigh distribution would have solved Parr's issue satisfactorily (we discuss his distribution later on).

There are also cases where a Rayleigh curve is really not a good fit for the given data, including a decentralized version. Then the limitations of Rayleigh curves should not function as a procrustean bed preventing accurate modeling of reality—when that reality is just no Rayleigh curve. Indeed this was also found in a 2001 US Air Force

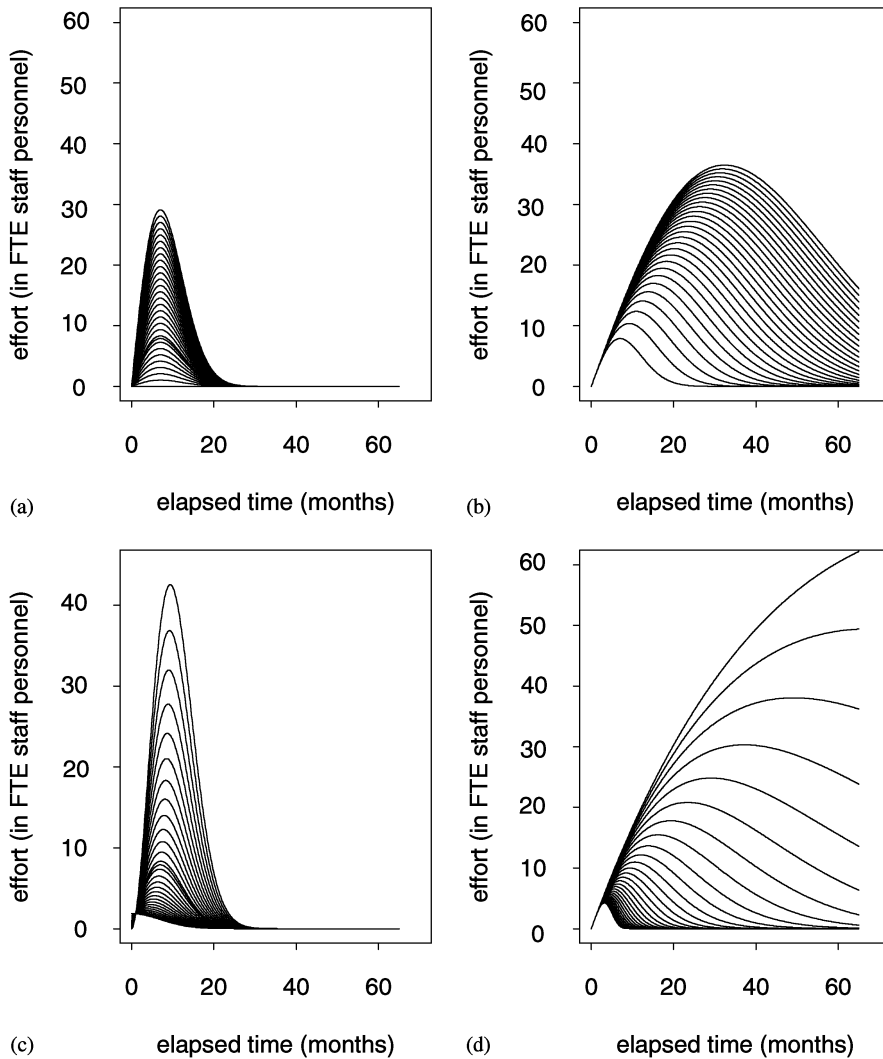


Fig. 11. Varying the four coefficients of formula (38): (a) $0 < a \leq 6.8$; (b) $0.00047 \leq b \leq 0.0102$; (c) $0 \leq \alpha \leq 1.8$; (d) $1 \leq \beta \leq 3$.

study, where a generalization of a Rayleigh curve was necessary to model funding curtailment to R&D programs [97]:

The Rayleigh function has the shape parameter set to a constant of 2. This makes the model somewhat rigid in its ability to model programs. The Rayleigh function forces a proportionate tail using the peak expenditure point as the start. In actuality there are programs where a proportionate tail is not derived from the point of peak expenditures. For example, a program may have a peak

expenditure during one time period and a very short tails—program expenditures stop shortly thereafter. The Rayleigh function would not provide an accurate model of reality in this case because of its rigidity tied to the constant shape parameter.

So Rayleigh curves are not the universal solution to the effort allocation problem. More flexibility is necessary especially when you lift from the project to the portfolio level. To give you an idea of the drastic variation you can achieve by varying the four coefficients of formula (38), we plotted several variations of Boehm’s example Rayleigh curve:

$$\frac{91t}{49} \cdot \exp - \left(\frac{t^2}{98} \right).$$

The original Rayleigh curve plus variants is plotted in each quadrant of Fig. 11. The variation that is possible with formula (38) is richer than those of a Rayleigh curve, and it is essentially needed for the purpose of quantitative IT portfolio management. Fig. 11(a) and (b) are all Rayleigh curves, whereas in 11(c) and (d) we relax the fixed powers characterizing the Rayleigh curve: we vary both powers.

If we look back at the statistically fitted seismic IT impulse to characterize the development cost allocation for our example portfolio M , we see that it is an instantiation of formula (38). It has an $\alpha = 1.25$, which is larger than allowed for a Rayleigh curve. There is a much faster effort buildup than a Rayleigh curve can accommodate. Indeed $\beta = 1$, which is small compared to a Rayleigh curve (where $\beta = 2$). And a smaller β leads to smaller wave lengths. This instantiation of formula (38) corresponds to a rapid staff build-up that cannot be fitted into a Rayleigh curve. Looking at the operational cost tsunami, it is obvious that this is also an instance of formula (38). While $\alpha = 1.69$ is large, the small $a = \frac{1}{41}$ is functioning as a *shock absorber* that dampens the peak. While $\beta = 1.32$ is relatively small, the tiny $b = \frac{1}{294}$ smooths the decay of the wave considerably. The good news is therefore, that you can fit more realistically IT portfolio costs using our cost allocation equation. The bad news is that with more degrees of freedom the curve fitting becomes more involved (more on tools and techniques to support the mathematics later on).

10.2. Cumulative cost allocation equation

It is insightful to partition a portfolio P into sets of IT projects that are somehow related. This relation could be a business unit, or the set of corporate wide systems, or systems in a similar phase: operations, development, retirement, outsourced, dormant, and so on. In this way, accumulating the individual projects does not lead to information loss about the type of investment. Moreover, P is then divided into sensible IT investment chunks just like an asset portfolio is partitioned into sensible categories. You can obtain the corporate view, by adding all these parts into one large function that describes the accumulated costs of the enterprise. In order to do so, we need accumulated cost functions for these groups of systems. From formula (38) we can

infer formula (39):

$$\begin{aligned} a(t) &= \int_0^t c(t) dt = \int_0^t a \cdot t^\alpha \cdot e^{-b \cdot t^\beta} dt \\ &= \frac{ab^{-(\alpha+1)/\beta}}{\beta} \left(\Gamma\left(\frac{\alpha+1}{\beta}\right) - \Gamma\left(\frac{\alpha+1}{\beta}, b \cdot t^\beta\right) \right), \end{aligned} \quad (39)$$

where $\Gamma(x)$ is the Γ function extending the factorial on natural numbers to real (and complex) numbers. The function name a stands for accumulated cost function. In our case, we can express Γ using Euler's identity:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt.$$

An interesting property of Euler's Γ is that $\Gamma(n+1) = n!$ for all $n \geq 0$. The 2-adic $\Gamma(a, x)$ is the upper incomplete Γ function we already introduced to calculate the accumulated costs for the seismic IT impulse and the operational cost tsunami. We note that for $(\alpha+1)/\beta = 0, -1, -2, -3, \dots$ the Γ function is not defined, and therefore also formula (39) is not defined.

10.3. Change in cost equation

Although at CMM level 1 organizations staff buildup on a per project basis is not a feasible metric to collect corporate wide, we can use formula (38) to project staff size globally. We simply take the derivative of formula (38), which leads to formula (40) for quantitative IT portfolio management:

$$cc(t) = (\alpha - b\beta t^\beta) a t^{\alpha-1} e^{-bt^\beta}. \quad (40)$$

The function name cc stands for change in cost function. We can calculate the peak time for the entire IT investment by solving the equation $cc(t) = 0$. This leads to formula (41):

$$pt = \left(\frac{\alpha}{b\beta} \right)^{1/\beta}. \quad (41)$$

In Fig. 5, we depicted the seismic IT impulse, and the resulting operational cost tsunami. With formula (41) we can calculate that the development peak load is at 17.2233 months, whereas the peak load for operations is at 90.30533 months—more than a factor 5 later than the development peak load. We can also calculate the peak costs, by simply calculating $c(pt)$. This leads to formula (42):

$$pc = a \left(\frac{\alpha}{b\beta e} \right)^{\alpha/\beta}. \quad (42)$$

For our example portfolio, we find peak efforts of 76.66294 million dollar for development, and 14.95207 million dollar top cost for operations—a factor 5 less than the development peak costs.

10.4. Putting it all together

The abstract formulas (38)–(40) can be used to obtain a corporate view of your IT portfolio. For a start, we can calculate total cost of ownership for the class of cost allocation functions we defined in formula (38). Formula (43) for quantitative management of IT portfolios is

$$tco = \frac{ab^{-(\alpha+1)/\beta}}{\beta} \Gamma\left(\frac{\alpha+1}{\beta}\right). \quad (43)$$

For our example portfolio, we summarized the development and operations coefficients in Table 8. These coefficients are used to calculate total cost of ownership with formula (43).

So, using formula (43) we find 3149.142 million dollar for development and 2262.23 million dollar for minimal cost of operation. If you look closer at formula (39), you can easily see that tco is the first part of the formula which is indeed independent of time. So, you could see the formula as follows: the first term is the price you will eventually have to pay for that part of the IT portfolio that is described by the cost equation. The second term is time dependent: it is the repayment rate ensuring that the IT portfolio is developed and deployed. Compare this to building and living in a house: the bank pays the sum that you cannot afford to pay instantly. The mortgage is the time dependent part that tells you when and how much installment is due to ensure that you can build and inhabit the house. So in fact, formula (39) gives you the TCO plus your debt to build and deploy the IT portfolio over time. This leads to the repayment factor expressed by formula (44):

$$rf(t) = \frac{ab^{-(\alpha+1)/\beta}}{\beta} \Gamma\left(\frac{\alpha+1}{\beta}, b \cdot t^\beta\right). \quad (44)$$

Given an IT portfolio P , that is partitioned in P_1, \dots, P_n , for which cost functions of the class defined in formula (38) are known, then the corporate cost of ownership for the portfolio at a given time t is given by formula (45):

$$cco(t) = \sum_{i=1}^n tco_i - \sum_{i=1}^n rf_i(t). \quad (45)$$

In Fig. 12, we superimposed for the example portfolio M the accumulated cost equation, the current cost equation, and the change in cost equation for the seismic IT

Table 8
Statistically fitted constants for development and minimal cost of operation for the IT portfolio M .

Constants	Development	Deployment
a	7.514287	0.02643959
α	1.258007	1.692713
b	0.07304098	0.003407055
β	1	1.317413

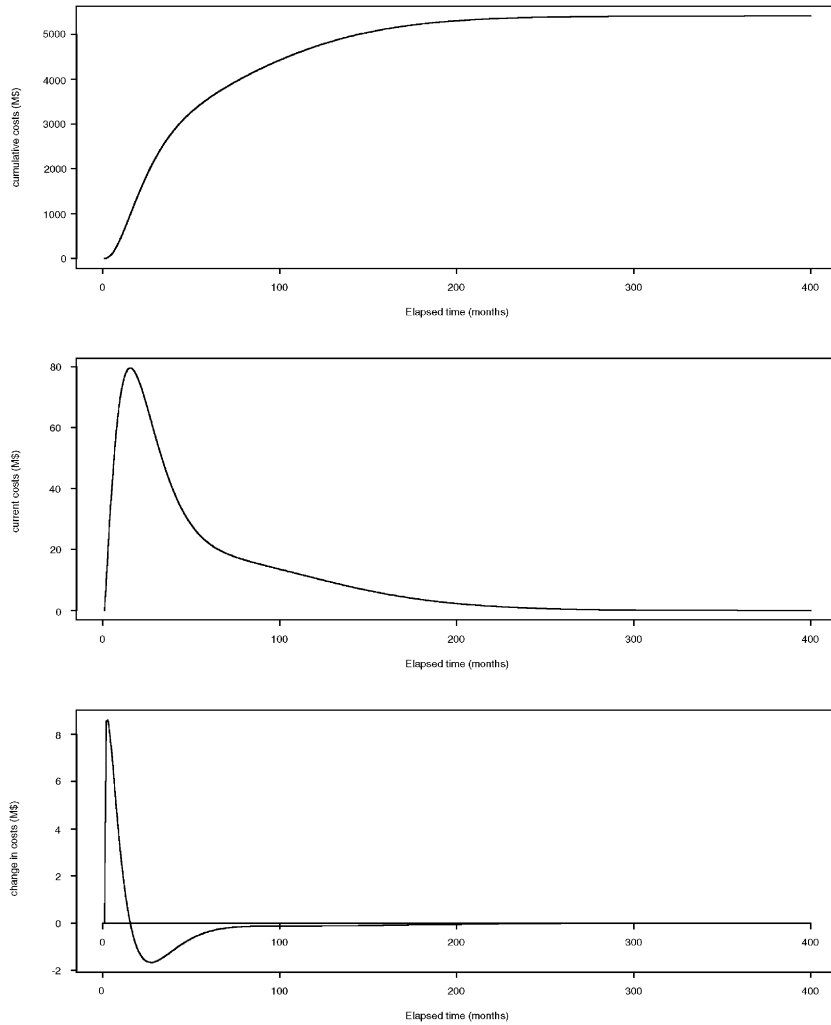


Fig. 12. Superposition of various functions for our example portfolio M .

impulse and the ensuing operational cost tsunami. This gives you a graphical view of the corporate cost of ownership of IT portfolio M . The accumulation of cost functions for many business units does not look as regular as Fig. 12, so in Fig. 13 we plot an accumulation of a variety of IT investments over time. These plots show typical patterns you can expect to find.

Total IT spending comprises IT investments started on various time stamps, with varying intensity, and varying start times of development within such investments. The current situation in many organizations is that they only have insight in the annual total IT spending cost, but not in how these costs are partitioned in related IT

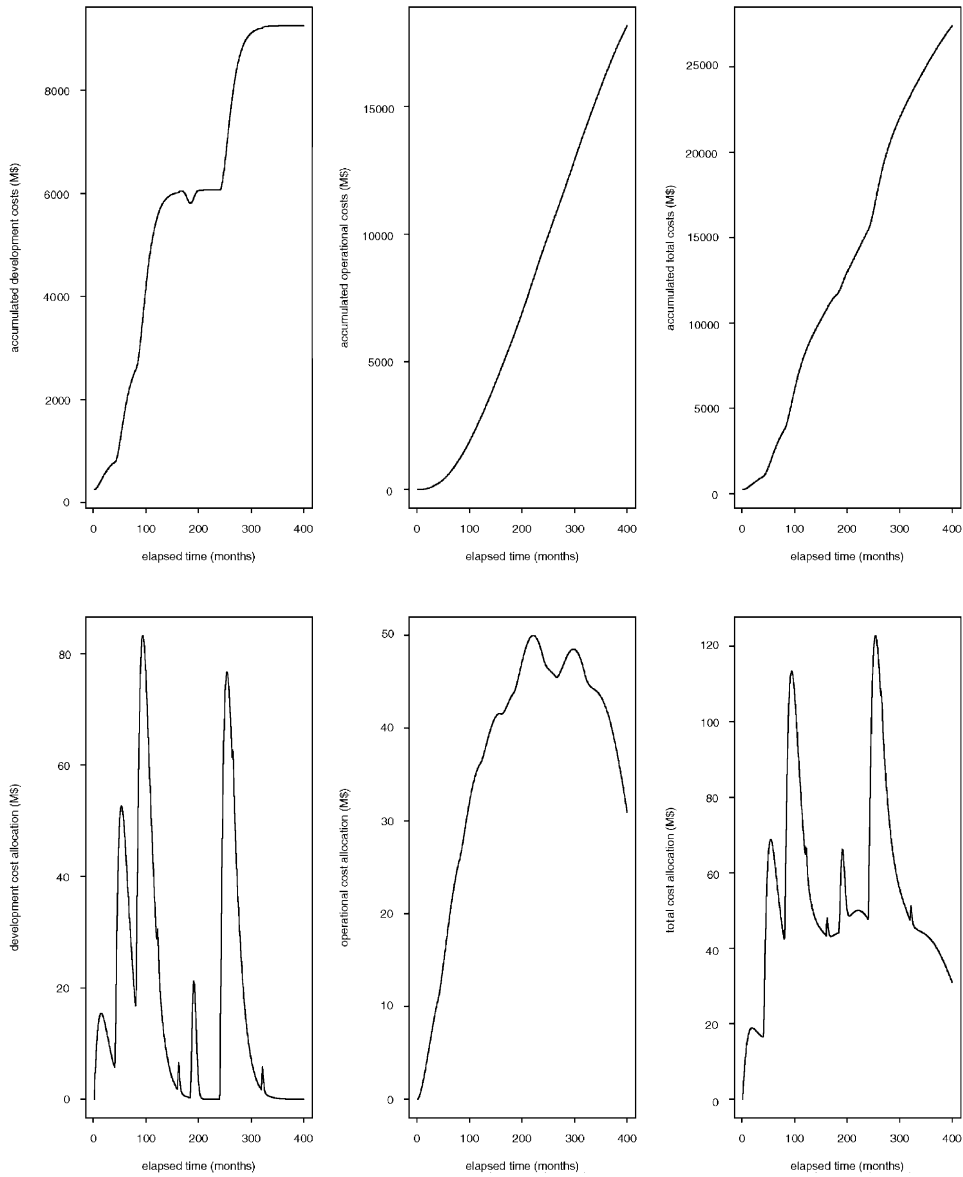


Fig. 13. Typical patterns when you accumulate IT investment costs over time.

investments. By grouping the IT investments over time, and analyzing these partitions, you can try to recover the cost allocation functions for major development, operations, and enhancement efforts. An IT portfolio database is a necessary—but not a sufficient—condition for this. In Fig. 13 we composed an example to illustrate this: we superimposed a number of major IT investments of our fictitious company, their

ensuing operational costs plus their enhancement costs. Some of them can be recognized by their peaks, others are faded out by the more dominating waves. The left-hand plot in the upper row of Fig. 13 shows the accumulation of all the development costs over time for the IT investments. The middle plot represents accumulated operational costs over time, and the right-hand plot is their sum: the total accumulated costs. The lower row plots the cost allocations for the IT investments. They are the cost allocation for development, operational and enhancement costs, and their sum. These accumulations give less insight in how the costs are build-up than if you would have had the cost allocation functions from the onset. The challenge is to uncover the cost allocation formulas from the data that is collected in the IT portfolio database. This is not an easy task, and there is no guarantee that you can completely recover the actual cost allocation formulas that belong to the IT investments of the past. The major investments leave so many traces that their recovery is often within reach. They are the cost waves from the past that are still dominating the current budgets.

11. Quantitative support for decision making

Next we turn our attention to how quantitative information can support decision making. Of course, the entire decision making process comprises of many factors, of which quantitative input is one aspect. Currently, not much quantitative data supports strategic decisions on IT investments. We agree with Strassmann who writes on this topic [122, p. 261]:

Credible financial analyses are necessary before top management can act with an understanding of the consequences of any decision.

We elaborate on quantitative support for a decision that many executives face when IT investments are due: outsourcing or not? For many organizations owning serious sized IT portfolios it is not possible to construct and maintain all the software in-house. So this is partly an in-house matter, and partly taken care of externally. At the executive level, decisions should be made to that end, and quantitative support will help in making the most effective decisions. There are the following possibilities for IT systems or IT portfolios:

- *Make*. This implies that you start (or restart) to make the software from scratch, and you are going to do this within the organization.
- *Buy*. This implies that the functionality can be bought off-the-shelf from an independent software vendor.
- *Commission*. This implies that you commission an outsourcer to build the desired functionality.
- *Renovate*. This implies that there is an existing software asset within your organization and that you want to renovate this to develop the desired functionality.
- *Wait*. This means that you postpone to decide on any of the above.

You can obtain quantitative information to support decision making by using various instantiations of formula (1), and if entire portfolios are considered for outsourcing, these formulas suffice to support decision making. As we noted earlier, these formulas have the proviso that you should not use them as the only means for single system decisions for contracting purposes. Since make-commission decisions are not only taken at the portfolio level, but also at the project level, we will develop some new formulas supporting decision making for contracting purposes. For these formulas we need richer information than estimated development time (or estimated total cost).

The smaller the amount of systems that are subject to make-commission decisions, the more realistic it becomes that you have to know the amount of function points involved. This amount can be obtained by carrying out a function point analysis. So we assume for the moment that for the part of the IT portfolio that is subject to make-commission decisions we know for each IT project its size in function points. We recall it is not necessary to know exactly what function points are except that it is a synthetic measure indicating the size of IT systems.

11.1. In-house development

First we need to obtain an idea of the *productivity* for in-house development of MIS systems. In CMM level 1 organizations there is no historical data around to infer productivity rates, so we use benchmarks to compensate for that. In [65, p. 184,189] six MIS development benchmarks are present that illustrate the relation between the productivity and size (based on many projects). Five of the benchmarks are derived by us from a graph (using a ruler), and one of them was stated in a table. Based on this, we fit a curve through these benchmarks. Formula (46) for quantitative IT portfolio management, expressing the productivity for MIS development (measured in number of function points per staff month) for a given size in function points, is as follows:

$$p_i(f) = 1.627 - 38.373 \cdot e^{-0.06222733 f^{0.424459}}. \quad (46)$$

In Fig. 14, the six dots are representing the benchmarks taken from [65] and the plot through the benchmarks is formula (46). Recall that the subscript i stands for information systems development. As an example, the productivity for in-house staff doing a 1000 function point MIS project is 13.6 function points per staff month (according to benchmark). We note that the asymptotic behavior of formula (46) is not in accord with reality: $p_i(\infty) = 1.627$. But for projects approaching infinite size, the productivity approaches zero. So formula (46) should not be used for projects larger than 100 000 function points.

Using formula (46), we derive alternative formulas of earlier derived formulas supporting quantitative IT portfolio management. But we can also infer an alternative for the benchmark $f^{0.39} = d$. For this we use another benchmark taken from [65, p. 185] that is called the *assignment scope for in-house MIS development*. An assignment scope for a certain activity is the amount of software (measured in function points) that you can assign to one person for doing that particular task. Note that the assignment scope is relatively size independent. We have seen two such assignment

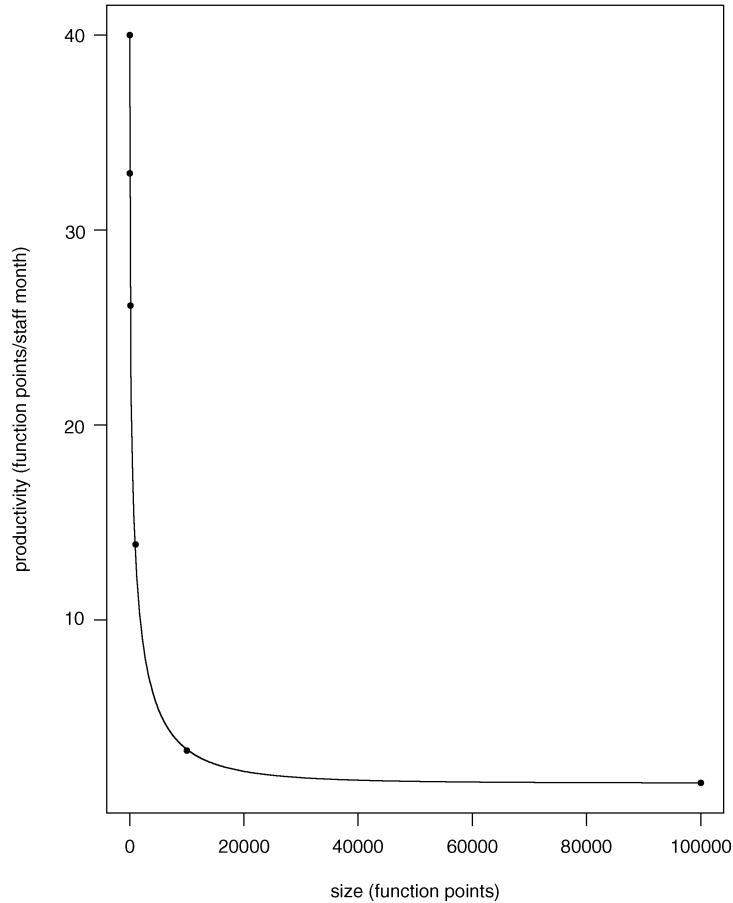


Fig. 14. Productivity for MIS development projects.

scopes: 150 as the assignment scope for average development over all sorts of IT systems and 750 for average operational costs. Indeed, depending on the task, the assignment scope can be different. For all activities that are usually done while developing MIS systems, the average assignment scope is 175 function points. Formula (47) for quantitative IT portfolio management calculates the amount of calendar months an in-house MIS development project takes.

$$d_i(f) = \frac{175}{p_i(f)} = \frac{175}{1.627 - 38.373 \cdot e^{-0.06222733 f^{0.424459}}} \quad (47)$$

For example, a 1000 function point development project takes $d_i(1000) = 12.9$ calendar months. We plotted the earlier used benchmark ($d = f^{0.39}$) against its alternative formula (47) to give you an idea of the deviations (viz. Fig. 15). According to the earlier used benchmark, development takes 14.8 months. The plot with an asymptote around the horizontal line at 107.6 is formula (47), the dotted curve is Jones'

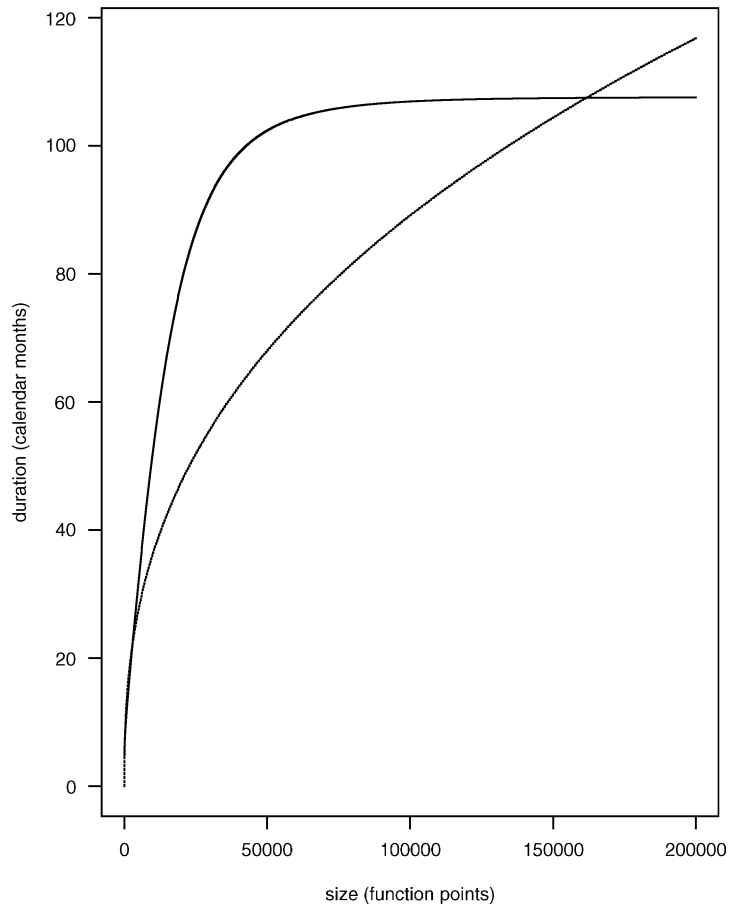


Fig. 15. Comparing two different benchmarks for in-house MIS development.

benchmark $d = f^{0.39}$. As you can see there are deviations so there is no single correct formula in CMM level 1 organizations. Fortunately, the formulas for quantitative IT portfolio management often provide enough quantitative data to help in deciding on portfolio investments.

We derive a formula similar to formula (1) calculating total cost of development for MIS projects using the just derived formula (47). Formula (48) calculates total cost of development for MIS projects for a given function point size.

$$tcd_i(f) = \frac{rw}{12} \cdot \frac{f}{p_i(f)} = \frac{rw}{12} \cdot \frac{f}{1.627 - 38.373 \cdot e^{-0.06222733 f^{0.424459}}} \quad (48)$$

As before, r is the fully loaded compensation rate and w is the number of working days in a calendar year. We derived formula (48) as follows. Using formula (47), we know the schedule in months. Then using the assignment scope, we know that the staff

necessary to do this project must be $f/175$. So the total effort is $f/175 \cdot 175/p_i(f)$ which amounts to $f/p_i(f)$ calendar months. The monthly compensation for this is $rw/12$, thus their product yields formula (48). So a 1000 function point project will cost in our fictitious company $tcd_i(1000) = \$1.3\text{M}$ ($r = \$1000$, $w = 200$).

11.2. Outsourced development

As in the previous section, we derive the same formulas but then specific for the outsource industry. We distinguish them from the in-house formulas by using the subscript o for outsourcing. We start with the productivity for outsourced software projects.

Analogously to the in-house situation we found in [65, p. 267,271] six outsource development productivity benchmarks. Again, five of the benchmarks stem from a graph, and one of them was stated in a table. We fit a curve through the benchmarks. Formula (49), expressing the productivity for outsource development (measured in number of function points per staff month) for a given size function points, is as follows:

$$p_o(f) = 2.63431 - 21.36569 \cdot e^{-0.01805819f^{0.5248877}}. \quad (49)$$

As an example the productivity of a 1000 function point project done by outsourcers is benchmarked on $p_o(1000) = 13.8$ which is a bit higher than in-house development productivity for 1000 function point projects ($p_i(1000) = 13.6$). The asymptotic behavior of formula (49) is not in accord with our experience. Very large projects do not have a lower bound of $p_o(\infty) = 2.63431$ for productivity. So formula (49) should not be used for projects larger than 100.000 function points.

In Fig. 16, the six dots represent the benchmarks taken from [65] and the plot through the benchmarks is formula (49). We use a benchmark taken from [65, p. 269]: the *assignment scope for outsource development*. For all activities that are common in the outsource industry, the average assignment scope is 165 function points. Using this we can infer formula (50) expressing the amount of calendar months for an outsource development project, given its size in function points.

$$d_o(f) = \frac{165}{p_o(f)} = \frac{165}{2.63431 - 21.36569 \cdot e^{-0.01805819f^{0.5248877}}}. \quad (50)$$

Also for outsourcing there is an earlier benchmark that relates function point size to project duration in calendar months: $d = f^{0.38}$. We plotted this benchmark against formula (50) to indicate the deviations (viz. Fig. 17). The solid plot is formula (50), the dotted curve is Jones' benchmark $d = f^{0.38}$.

Now we can derive total cost of development, similarly to formula (48). Formula (51) calculates total cost of development for outsource projects for a given function point size.

$$tcd_o(f) = \frac{rw}{12} \cdot \frac{f}{p_o(f)} = \frac{rw}{12} \cdot \frac{f}{2.63431 - 21.36569 \cdot e^{-0.01805819f^{0.5248877}}}. \quad (51)$$

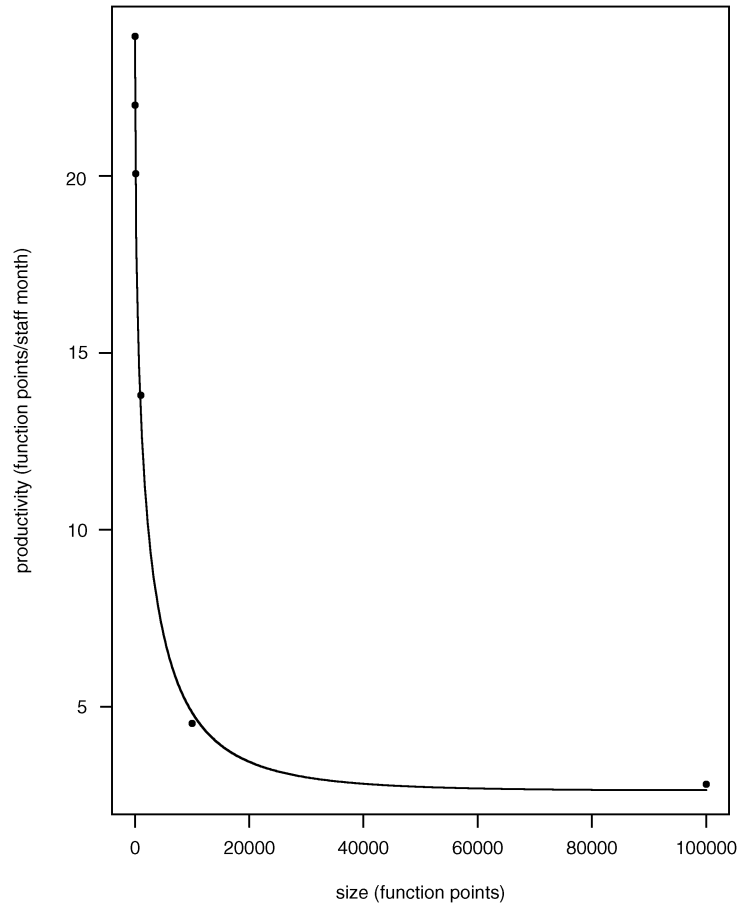


Fig. 16. Productivity for outsource development projects.

So, a 1000 function point system costs $tcd_o(1000) = 1.2$ million dollars (we took $r = \$1000$, $w = 200$).

11.3. Quantitative comparison

With the just derived formulas, we can compare development costs of IT systems done in-house with outsourcing such systems. Of course, there will be different daily rates, and in case of off-shore outsourcing also different working days per year. These different numbers are not hard to obtain when you are discussing contracts with an outsourcer.

As an example, suppose you need a 10 000 function point information system and you want to explore the possibilities for outsourcing. Of course, competitive issues play a role in such decisions. For a start, externals tend to share their knowledge obtained in

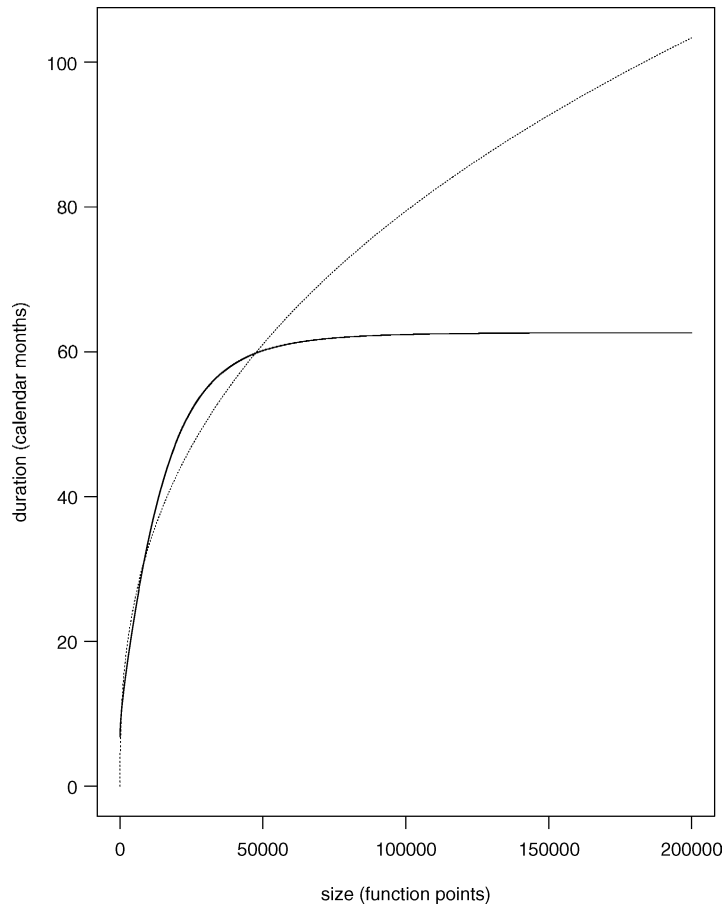


Fig. 17. Comparing two different benchmarks for outsourcing.

your project by doing similar jobs for others. By way of anecdotal evidence consider a quote taken from [44, p. 61], clearly showing that your trade secrets are not always safe when you ask others to implement a discretionary effort:

He showed Michalik a technology that an engineering friend had built for the Swiss bank UBS; [he] told Michalik that he had a killer app on his hands.

So when the IT system is a discretionary effort, you may not want to outsource it, even if the development cost are markedly lower. For instance, the expected return in combination with being the first in your branch, could potentially reap more benefits than lower costs, and the danger of being imitated as soon as returns are apparent for the competitors. Or, if there is no other option than to outsource, consider to protect vital parts of the innovation by one or more patents. Such considerations are outside the scope of this paper. We solely provide the decision maker with quantitative data forming one ingredient of the decision making process.

Table 9
Several indicators to compare in-house development with outsourcing

Indicator	Dimension	Make	Commission
r	\$	666	1000
w	days	200	200
$p(f)$	FP/SM	3.35	4.84
$d(f)$	CM	52.24	34.10
$tcd(f)$	M\$	33.13	34.43
$cf(f)$	%	39.0	27.0
$cl(f)$	%	33.7	33.7

Let $r_i = \$666$ be the fully loaded daily rate for in-house MIS development. We assume $w_i = w_o = w = 200$ working days/year. This leads to a burdened monthly compensation of 11 100 dollar. Let $r_o = \$1000$ be the daily rate of an outsourcer, which leads to a monthly compensation of 16 666 dollar. The monthly compensation we took is not a contrived difference, but in accordance with 2002 compensation rates.

In Table 9 we summarized some important indicators to support decision making. We used some abbreviations as well: FP/SM stands for function points per staff month, CM is short for calendar months, M\$ stands for a million dollar US. In both cases the initial development costs are equal, since the outsourcers are faster with larger projects (according to benchmark). They both have a 33+% chance of being late. Note that schedule slips of in-house development is less expensive than schedule slips of outsourcers. The chance of failure is 12% lower, though. If speed to market is important, and information leaks to the competitor are not too much of a problem, then the quantitative data supports an outsourcing decision. If you expect the system to be mission-critical during its 10+ years of deployment, then it may be better to maintain and enhance it in-house. If the system is planned well in advance, the longer development schedule is not too much of a problem. As you can see, the final decision depends on more than data such as summarized in Table 9.

To get an idea of such comparisons for various software sizes, we plot Fig. 18. The in-house productivity expressed by formula (46) is the solid curve, and the dotted curve is formula (49) calculating the productivity of the outsource industry. The productivity of smaller projects is better for in-house projects than by outsourcers. But larger projects are more productively done by outsourcers. One of the reasons for this higher productivity is involuntary unpaid overtime (so not necessarily better skills). This also clarifies why the chance of late delivery is not that different, and that the schedule in calendar months is much shorter. We depicted the schedule as function of size in Fig. 19. Formula (47) is the solid curve in Fig. 19, and formula (50) is the dotted curve. The development schedule of outsourcers is much shorter when the system-size increases. In Fig. 20 we depict how this translates into development budgets. The solid curve is formula (48) and the dotted curve is formula (51). The comparisons for the 1000 function point example showing that the costs are not dramatically different is an overall trend, both the solid cost curve in Fig. 20 and the dotted outsource variant are less deviating than the development schedules might have insinuated.

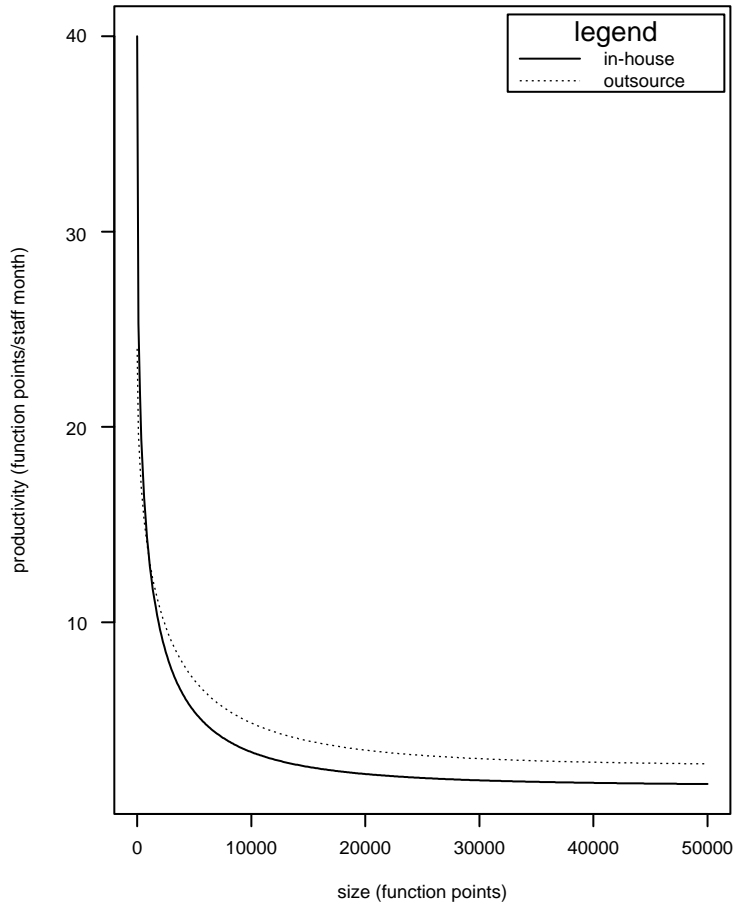


Fig. 18. Comparing MIS development to outsource development productivity.

Another interesting comparison is the risk dimension. We plot in Fig. 21 formulas (28) (the solid curve) and (30) (the dotted curve). The chance of failure for outsource projects is smaller than for in-house development of MIS applications. This is not true for the exposure of being late. In Fig. 22 we depicted formulas (32) and (34) expressing the change on late projects in-house and by an outsourcer respectively. The dotted curve expressing late outsourced projects is above the solid curve for late MIS projects done in-house. So although the chance of failure is smaller, the chance of being late is larger. This might be due to the fact that when you sign a contract with an outsourcer, *not* delivering is an obvious contract violation. So outsourcers deliver, but suffer from more time/effort overruns than the in-house case. In-house development fails more often, but if they do not fail, they deliver less late.

This type of quantitative input supports strategic decision making, but other factors are as important: the business goal of the system, its criticality for the business, the deepness of the stakeholders' pockets, the competitive landscape, etc.

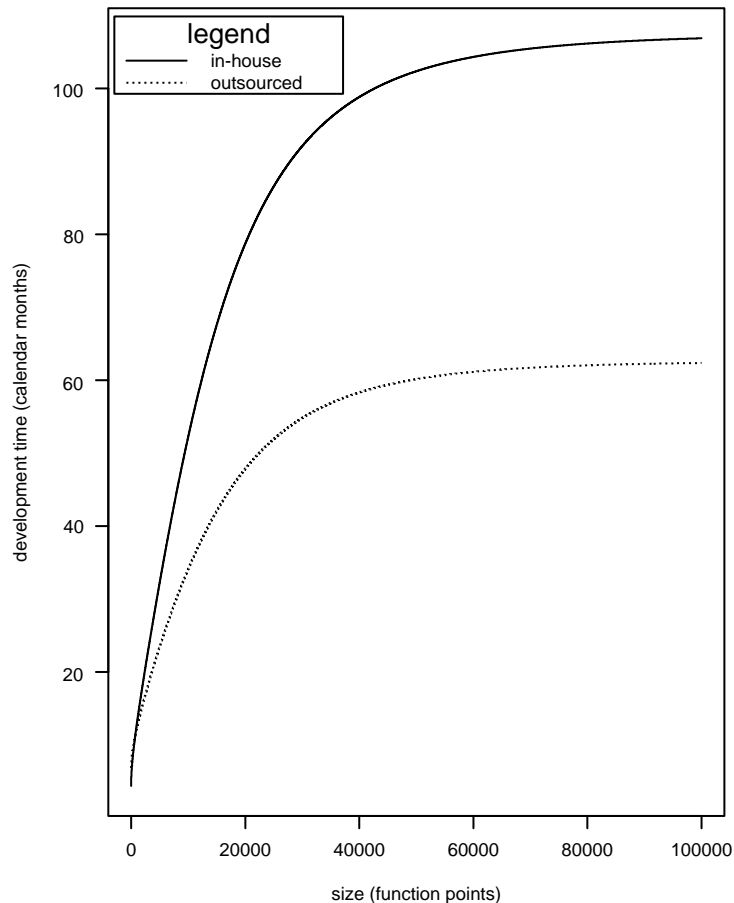


Fig. 19. Comparing MIS development to outsource development schedules.

12. Cost–time analysis and lifetime analysis

We could develop many more formulas for quantitative IT portfolio management, supporting strategic decision making for IT portfolios and sanity checking on IT projects. But at this point we think it is worthwhile to turn our attention to a more fundamental issue. It is the issue whether it is possible to incorporate our empirically found formulas within an existing body of mathematical and statistical knowledge. For, if we are able to connect our work to established theory, we can benefit from findings in that area, and insights from these areas could lead to insights in the formulas we developed thus far. Others have tried to connect quantitative IT portfolio management to modern portfolio theory, and we showed that this correspondence is not as promising as it seemed at first.

After a careful study of our empirically found formulas we are confident to have found this existing body of knowledge. It is called *lifetime analysis* or *failure time*

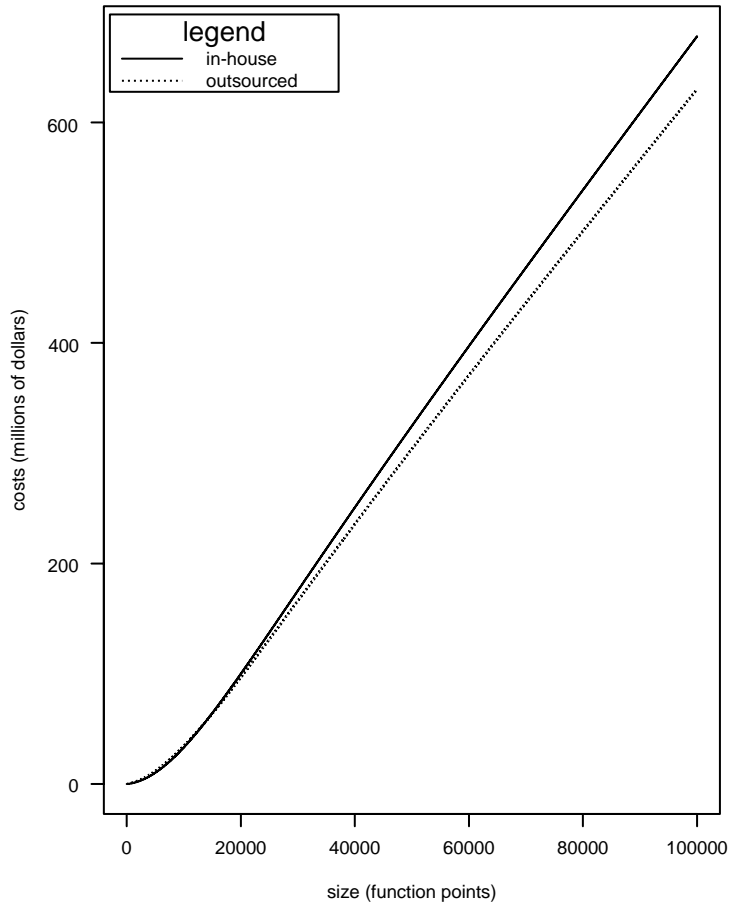


Fig. 20. Comparing MIS development to outsource development costs.

analysis as is applied to cancer research, cure rate estimation, reliability analysis, and other areas ranging from returns on the NYSE to the physical laws that the crushing of coal are subject to. This analysis will follow shortly, and it made us think of our work as *cost–time analysis*.

In the practical software engineering area, there is no established tradition of mathematically describing important phenomena in order to control the engineering process, and the ensuing artifacts [76,43]. In part this is due to the fact that the mathematics is not closely connected to obvious applications that are of immediate use in practice. For fields where this is obvious, such as software cost estimation, not many people relate their work to common practice in mathematics or statistics. Let us illustrate this. Recall Boehm [6] and Parr [90] who go to great length to infer alternatives to Rayleigh distributions whereas they could have used a decentralized version. Another indication is the ongoing discussion how useful mathematics is for the software practitioner. We

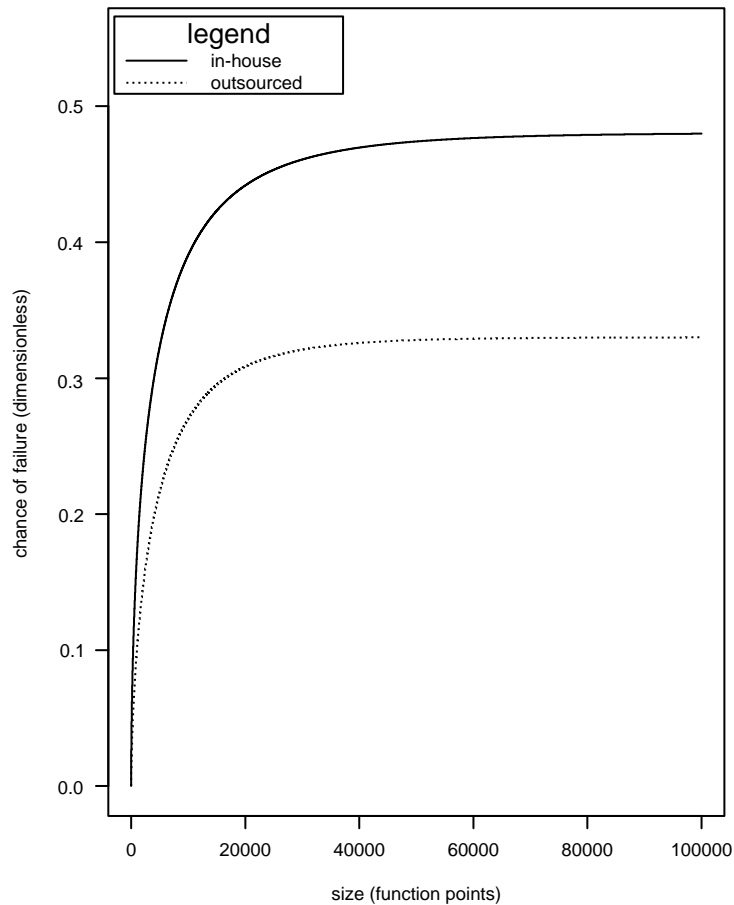


Fig. 21. Comparing MIS development to outsource development chance of failure.

refer to Glass who in IEEE Software [43] writes:

If these mathematics-based [approaches] ... are truly important to the field, then it is in some ... application of software that I have not yet encountered.

We assume Glass thinks of formal methods, and indeed, it is less obvious how (and when) to apply formal methods in industry. But software cost estimation is omnipresent, and cannot be done without mathematical and statistical support. So the practical software area that Glass apparently never encountered where math is staring you in the face is software cost estimation, and the related area of quantitative IT portfolio management—the subject of study in this paper.

Due to the traditional lack of applying mathematics successfully in practical software engineering, the mathematics that *is being used* is not related to the rich body of standard mathematics and statistics. It seems that progress in research on software

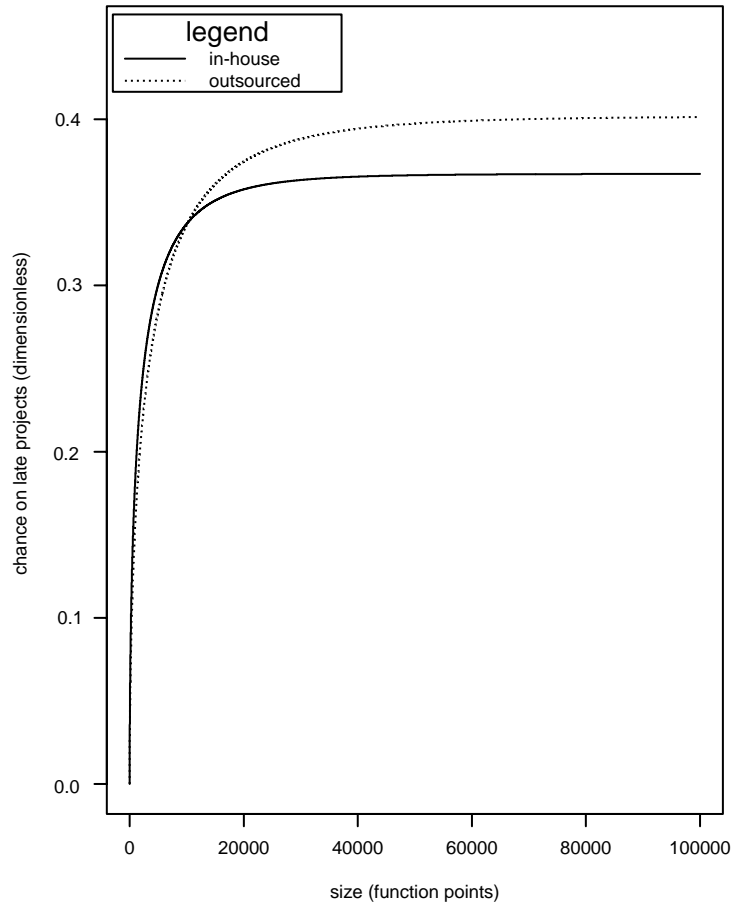


Fig. 22. Comparing MIS development to outsource development chance of late projects.

cost estimation is bogged down by the assumption that you first have to theoretically justify how knowledge accumulation and problem solving processes are modeled mathematically, in order to analytically derive software cost estimation formulas. In other areas where mathematics and statistics are necessary, people explicitly refrain from such practices. For, this only leads to Mickey mouse mathematics³ lacking general applicability, since it is based on too idiosyncratic assumptions reflected by personal or otherwise limited experience, which is a bad advisor when it comes to mathematics.

As another illustration, consider this: in a textbook by Londeix on cost estimation for software development [78, p. 90] a learning function $2at^n$ is considered in an

³ www.mathematicallycorrect.com/glossary.htm

exercise for which the manpower distribution and cumulative manpower cost are to be calculated. Londeix rejects this particular distribution on the following grounds [78, p. 195]:

We can verify that when n increases the time scale is reduced. An increase of n gives a sharp rise of the peak manning relative to its value for $n = 1$. Therefore, a non-linear learning curve would give an early more peaky model which would not be helpful to represent the reality of software development.

This is further justified in the textbook by a reference to Norden's model [88]: since Norden is using a linear learning curve, the nonlinear learning curve cannot be correct. Needless to say that this line of reasoning is erroneous. A learning curve can never be inappropriate *because* someone else is using a different curve. Irrespective of his line or reasoning, Londeix is mistaken: we know from empirical research reported on in [97] that nonlinear learning curves accurately model IT intensive programs. The only limitation that Londeix should have questioned is that the assumption that n is a natural number is too restrictive.

So the general tendency in software cost estimation is to first restrict one selves and within those limitations try to model cost estimation phenomena. This is the wrong ordering as we will argue below.

12.1. Ready, fire, aim

In general it is not smart to fix a specific (theoretical) cost–time model in advance—and then see if this assumption fits reality. Unless the theoretical derivation is conclusive, this should *always* be done the other way around: you decide on a—hopefully general enough—*family* of distributions, and by curve fitting the most likely distribution for that particular effort will turn up as the result of statistical analysis.

The fixed-model approach is not the way to go when you just want to control things accurately, but this ready-fire-aim approach seems the norm in software engineering. It is much better to skip the motivational part all together, and instead do an educated guess that the relation you wish to describe probably fits a very general family of distributions like our family of cost allocation functions (38) is fortuitously doing. Or discriminate among several parametric models, to see which one should be used at all [99]. To quote Lawless [75, p. 13], who touches upon this issue in the realm of lifetime analysis:

Extensive motivation is not provided for the various models. To do this would require a thorough discussion of aging and failure processes and would take us outside the book's intended subject area. Indeed, the motivation for using a particular model in a given situation is often mainly empirical, it having been found that the model satisfactorily describes the distribution of lifetimes in the population under study. This does not imply any absolute “correctness” of the model.

Kalbfleisch and Prentice go a step further, when they write about failure time data analysis [67, p. 3]:

In many situations it is also important to develop nonparametric and robust procedures since there is frequently little empirical or theoretical work to support a particular family of failure time distributions.

They even indicate that you might question the assumption that there would be an analytic relation at all. We are in complete agreement with the observations done by Lawless, Kalbfleisch, and Prentice.

Also consider this: there can be some time between empirically found mathematical tools and their formal underpinning. For instance, the so-called Weibull distribution was empirically found in 1933 by Rosin and Rammler to describe the crushing of coal [107] (it was later attributed to Weibull [131]). However, the first full theoretical derivation based on physical principles stems from 1995 [14]. In the mean time, the distribution has been very instrumental in lifetime analysis [75].

The statistical ready-fire-aim practice is not unique to software engineering. This issue is also noted in other areas. For instance, in the realm of cure rate estimation in clinical trials for diseases such as lymphoma and breast cancer, similar critical remarks, eloquently expressing the viewpoints of [67, p. 67], are made [94]:

In the last decade, mixture models under different distributions, such as exponential, Weibull, log-normal and Gompertz, have been discussed and used. However, these models involve stronger distributional assumptions than is desirable and inferences may not be robust to departures from these assumptions. In this paper, a mixture model is proposed using the generalized F distribution family. Although this family is seldom used because of computational difficulties, it has the advantage of being very flexible and including many commonly used distributions as special cases. The generalised F mixture model can relax the usual stronger distributional assumptions and allow the analyst to uncover structure in the data that might otherwise have been missed.

Indeed, using fixed or restricted families of distributions also restricts the possibility to fit your data, and as the authors of [67,94] correctly observe, using a general distribution, you can uncover structure in your data that with a restricted model is missed. The generalized F distribution was originally intended as a selection process to detect which known less general distribution fits data most appropriately [99].

So, instead of trying to unravel *why* a particular curve is appropriate to model a correlation you can better skip that part for later and readily begin to fit curves that are approximating the relation accurately. *Any* formula, satisfying your purpose is by definition useful—the answer why this formula does fit is just a *nice-to-have*. As you will see shortly, it turns out to be utterly useful to relate our cost–time formulas to established practice in statistical modeling.

12.2. The generalized Γ distribution

In order to do so, it is necessary to turn formula (38) into a so-called probability density function. We normalize formula (38) so that the area under the curve

becomes 1, by dividing formula (38) by the area under it. We use formula (39) for this. Formula (52) is the probability density function variant of formula (38):

$$f(t) = \frac{\beta b^{(\alpha+1)/\beta}}{\Gamma(\alpha + 1/\beta)} \cdot t^\alpha e^{-bt^\beta}. \quad (52)$$

Formula (53) is the cumulative distribution function belonging to formula (52):

$$F(t) = 1 - \frac{\Gamma((\alpha + 1)/\beta, bt^\beta)}{\Gamma((\alpha + 1)/\beta)}. \quad (53)$$

Indeed normalization means that $F(\infty) = 1$, which is not hard to check. Now we can easily relate formulas (52) and (53) to established statistical tools.

Theorem 1. *Formula (52) is equivalent to the generalized Γ distribution [115].*

The generalized Γ distribution is defined as follows:

$$g(t) = \frac{\beta}{\Gamma(k) \cdot \theta} \left(\frac{t}{\theta}\right)^{k\beta-1} e^{-(t/\theta)^\beta}.$$

If we take $\alpha = \beta k - 1$ and $b = 1/\theta^\beta$, we can easily find that formula (52) turns into the generalized Γ distribution. Vice versa, if we take $k = (\alpha + 1)/\beta$ and $\theta = (1/b)^{1/\beta}$, this reduces the generalized Γ distribution to formula (52). So, both distributions are the same.

Obviously then also all the other artifacts coincide, e.g., their cumulative distribution functions are the same.

12.3. Lifetime data analysis

So now we have related our normalized empirically found cost–time family of relations to an existing family for which there are tools and techniques available that we can borrow to effectively deal with the practical side of applying the necessary statistical analyses. This field is not modern portfolio theory, but lifetime data analysis.

In the 1930s functions similar to the generalized Γ distribution were used to analyze the distribution of economic income [2,25]. But also in 2001, it was shown that stock returns on the New York Stock Exchange can be approximated by a generalized log gamma distribution [11]. So there *are* relations between quantitative IT portfolio management and the returns on stock options, but as argued, these relations are not what you would expect, namely that security portfolio theory founded by Markowitz corresponds in some natural way to issues relevant for IT portfolios.

A major application field of the generalized Γ distribution is in the so-called lifetime data analysis [75]. This branch of statistics is also referred to as survival time, or failure time analysis [67] and is widely used in engineering to support reliability analysis, and in the biomedical sciences. While the notion of lifetime should be taken literally, e.g., in biomedical science, in other fields it merely indicates a non-negative-valued variable. Our application comprises cost–time analysis for IT portfolios. There is a one-to-one

correspondence with concepts from lifetime analysis, and the field of software cost estimation, in particular quantitative IT portfolio management.

Let us first review the few most basic concepts of lifetime data analysis (this information can be found in any book on lifetime data analysis), so that we can illustrate the strong correspondence with IT cost issues. Suppose T is a nonnegative random variable representing a lifetime, e.g., of individuals in a population. Let $f(t)$ be the *probability density function* of T . Then the *distribution function* $F(t)$ is defined as follows:

$$F(t) = P(T \leq t) = \int_0^t f(x) dx,$$

where $P(T \leq t)$ denotes the chance that the lifetime T is between zero and t . The *survival function* is defined as

$$S(t) = P(T \geq t) = \int_t^{\infty} f(x) dx.$$

As we can see, when $t \rightarrow \infty$, then the distribution function F will approach 1, whereas the survival function will approach zero: $S(\infty) = 0$. In other words, it becomes harder and harder to survive when time proceeds. An important notion in lifetime analysis is the so-called *hazard function*, also known as the hazard rate, the age-specific failure rate, or the more poetic name: *force of mortality*:

$$h(t) = \frac{f(t)}{S(t)}.$$

The hazard function expresses the instantaneous death rate at time t given that there is survival up till t . The hazard rate thus describes the way in which the instantaneous death of an individual (or the failure of some device) changes with time. So when you follow subjects from birth to death the hazard rate can be a bathtub curve: right after materialization there can be childhood diseases, then a relatively constant rate, and then the rate will go up again. The *cumulative hazard function* is easily defined:

$$H(t) = \int_0^t h(x) dx.$$

With these basic definitions it is possible to derive a number of fundamental relations between the various notions. For a start, the probability density function can be written as a product of two intuitive functions:

$$f(t) = h(t) \cdot S(t),$$

and since $f = -S'$, it is not hard to find that

$$S(t) = e^{-\int_0^t h(x) dx}.$$

Combining the above two formulas:

$$f(t) = h(t) \cdot e^{-\int_0^t h(x) dx}.$$

So, the hazard function can serve as a means to derive the probability density function. For instance, suppose that the hazard function is constant: $h(t) = \lambda$. Then the probability density function is $f(t) = \lambda e^{-\lambda t}$, which is the exponential or Poisson distribution that is often used in reliability and lifetime analysis.

12.4. Correspondence to software cost estimation

Let us give a second example, to start showing the one-to-one correspondence. Norden presupposed that the effectiveness of a group of engineers increases progressively during the life cycle of a project that he represented by a function $p(t)$, where the p is an abbreviation for the *problem* function indicating the level of skill available to solve the problems. He assumes this function to be linear. In fact the function p is what in lifetime analysis is called the hazard function. Norden, thus assumes a linear hazard rate. With the basic formulas for lifetime analysis, finding the probability density function is obvious:

$$f(t) = at e^{-at^2/2}.$$

This is the Rayleigh distribution that Norden derived, using differential calculus. In lifetime analysis, the Rayleigh distribution is also known as the linear hazard rate distribution [4,71].

In the paper [90], Parr proposed an alternative to the Rayleigh distribution to estimate software costs. This model is sometimes called the sech square model due to its formulation:

$$V(t) = (1/4) \operatorname{sech}^2((\alpha t + c_3)/2).$$

We note that $\operatorname{sech}(x) = 2/(e^x + e^{-x})$, and that *sech* stands for *secans hyperbolicus*. Parr goes to great length in deriving this formula using differential calculus. We think that without knowing it, he just proposes a *logistic hazard rate* modeling his ideas on the rate at which problems are solved in software development. When you assume logistic growth for problem solving, the sech square formula follows immediately using the basic relations for lifetime analysis. The hazard function Parr actually assumes is as follows (viz. Fig. 23):

$$h(t) = \frac{\alpha}{1 + ae^{-\alpha t}}.$$

Looking at Fig. 23 you can see that the hazard function first looks like a linear hazard rate, so the beginning is a decentralized Rayleigh curve, and later on, the hazard rate becomes constant so then it will approach an exponential, or Poisson, density function. So it is just a smooth mix of linear and constant hazard rates. Maybe some IT projects do have a problem solving curve that resembles the one of Fig. 23. Then again, maybe others do not. For instance, in [57,56] the work of Parr is used to derive an alternative to Parr's work for the phasing of resources: the damped sine model. Also these authors use differential calculus to theoretically derive an effort distribution function based on

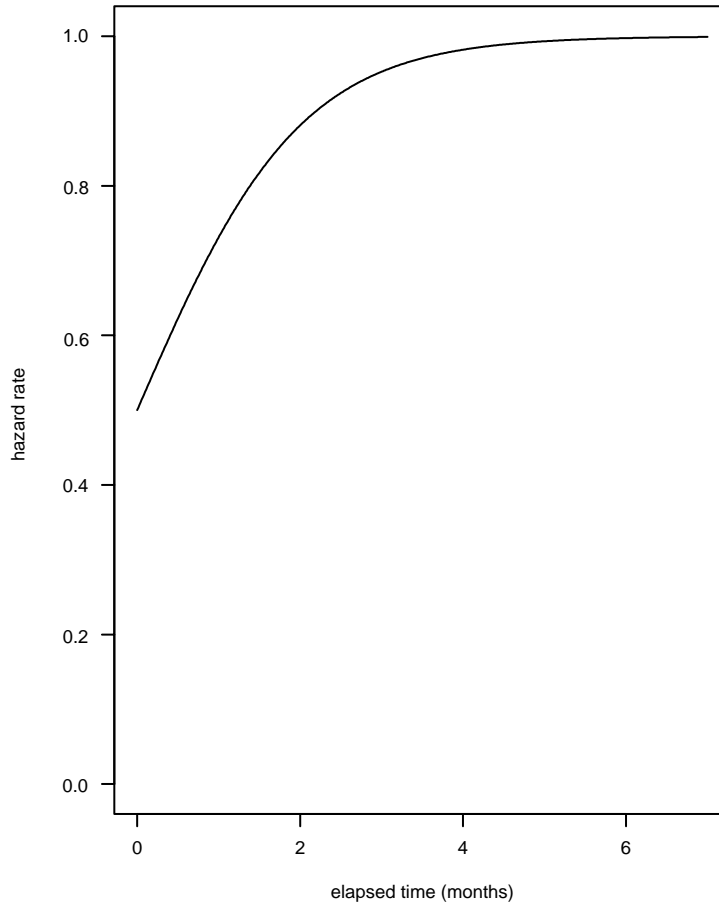


Fig. 23. The shape of logistic growth.

their ideas of problem solving rates. It is the damped sine model (see Fig. 24):

$$f(t) = c \cdot e^{-at} \sin(bt).$$

The shape of the damped sine model could very well be an effort distribution albeit that this function oscillates around zero, thus can be less than zero, which is not intuitive for cost–time analysis. To improve our intuition for this model we calculated the survival rate, and the hazard rate (see Fig. 25 for plots of f, S, h):

$$S(t) = \frac{c \cdot e^{-at}}{a^2 + b^2} \cdot (b \cos(bt) + a \sin(bt)),$$

$$h(t) = \frac{a^2 + b^2}{a + b \cot(bt)}.$$

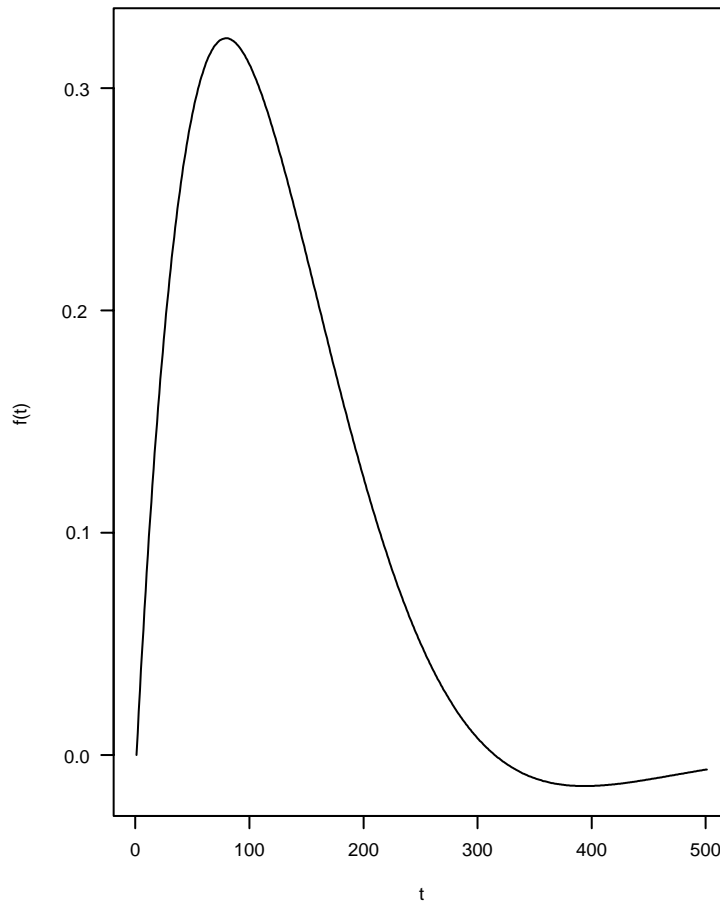


Fig. 24. The damped sine model.

Looking at Fig. 25, the hazard rate seems first linear, and then it approaches asymptotically to infinity. You could say that this hazard rate is roughly the inverse of the logistic hazard rate. So this is a smooth mix of a Rayleigh distribution at the beginning and an infinite hazard rate at the end. The latter is rather nonintuitive, and can be due to the following. One of the things that is modeled in [57,56] is that a project has an endpoint, where the effort model should be exactly zero. This can only be modeled when at some positive point in time a zero can be produced in the model. The sine curve has this property. This then leads to the unnatural hazard rate. We think that trying to model this type of assumption is not helping in coming to grips with cost–time analysis. It only complicates matters considerably. Apart from that, the assumption that effort should be exactly zero is not making much sense for high-momentum work. As is reported on in [30, p. 63] it requires 15 min or more of concentration to reach the state of flow that is necessary to do engineering, design, development, writing, or

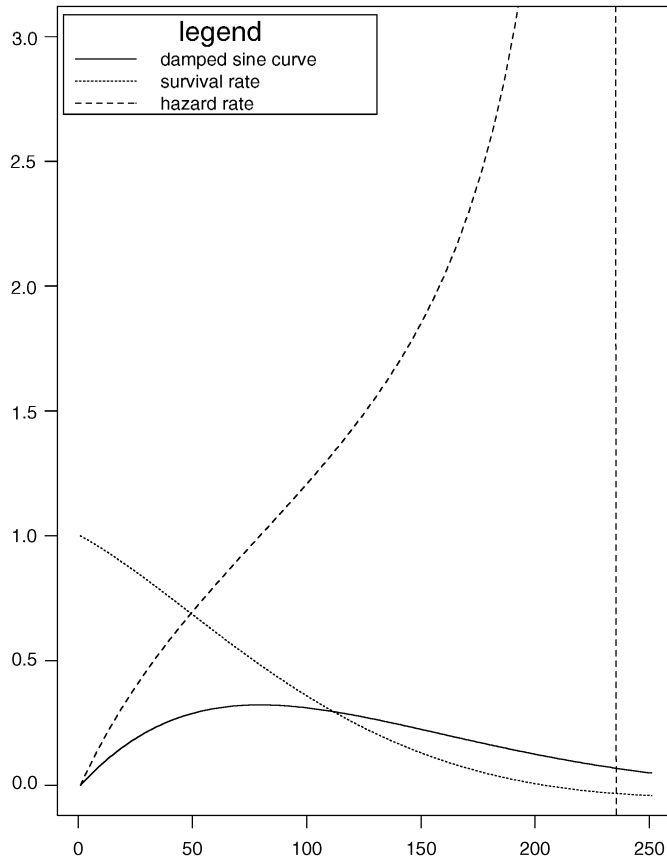


Fig. 25. The survival rate S , hazard rate h , and its product f : the damped sine model.

similar work. This implies that it is not necessary to model such efforts at a granularity smaller than 15 min. As you can see, modeling the fact that a project ends, is leading to problems that should better be avoided. Therefore, it is better to avoid preliminary assumptions on how problems are solved, and fit data using as general as possible families of functions, so that you do not miss trends that you will most probably miss when you are too restrictive about the family of curves to choose from. It is illustrative that in [57,56] a more flexible family of curves is abandoned:

The beta curve provides great flexibility; however, a theoretical justification for use of the curve is lacking.

Although we appreciate the efforts of the authors of [57,56] we are convinced that it would be better to adhere to flexible families of curves, rather than restrict yourself to theoretically derived curves that can easily be too restrictive to model the reality. We prefer pragmatic flexibility over theoretical foundations that may be hard to justify after all.

In the next example we will see that a too restrictive model broke down, and that a more flexible model solved the problems. Moreover, it illustrates the strong relations between lifetime analysis and software cost estimation. Recall the earlier cited US Air Force study by Porter. He used the Weibull distribution for recalibrating the costs and lifetime of R&D programs [97]. Indeed, the linear hazard rate did not give Porter enough freedom to fit the experimental data to Rayleigh distributions. Apparently, the hazard rate, that he calls performance rate for R&D programs is not linear but follows some different pattern. Without realizing this, he assumes the following hazard rate: $\beta t^{\beta-1}$. Also in Porter's paper differential calculus is used for inference of the cost–time function. But again, using the basic concepts of lifetime analysis the probability density function is obvious: $\beta t^{\beta-1} e^{-t^\beta}$. And this is indeed the Weibull distribution that is often used in lifetime data analysis.

As another example, recall the learning function in the earlier mentioned textbook by Londeix [78]. In that book, a hazard rate of $2at^n$ is considered. To derive the distribution, the textbook also uses differential calculus. Again, using the lifetime data analysis basics, it is trivial to infer the answer.

So you could say that for estimating costs of software projects, R&D programs, and IT portfolio management, the hazard function in lifetime analysis corresponds to the problem solving rate, that the survival function corresponds to the percent of work remaining, that is, the residual investment, that the distribution function F corresponds to the accumulated cost function, and the density f corresponds to the manpower buildup function. Therefore, you can see cost–time analysis as lifetime analysis.

This is in accord with the interpretation of the Rayleigh curve: the linear part stands for the learning curve to overcome problems one at a time, which is the hazard function, and the quadratic exponential factor represents the velocity with which you can solve those problems once the solutions are known, which is the survival function.

12.5. Hazard rate and survival function

The hazard and survival function provide central intuitions for a cost–time analysis—just as they justify the use of specific models in lifetime analysis. For any cost–time distribution you can calculate these functions, as we did for a number of known ones, to better understand the distribution, and its feasibility. Vice versa, if you know for a fact what the problem solving rate of an R&D-like project is, then you can immediately infer the correct distribution. But knowing this rate implies that you presumably understand the problem in the first place, so then it is not an R&D project anymore. This paradox shows that trying to theoretically infer a cost–time distribution seems to be feasible only for well-understood problems. Measuring problem solving rates seems infeasible for CMM level 1 organizations, who are lacking an overall metrics program.

To gain insight in the empirically found formula (52) it is therefore worthwhile to calculate the survival and hazard functions. The survival function is easily inferred from formula (53). Formula (54) for quantitative IT portfolio management is

$$S(t) = \frac{\Gamma((\alpha + 1)/\beta, bt^\beta)}{\Gamma((\alpha + 1)/\beta)}. \quad (54)$$

In cost–time analysis, we call the survival function sometimes the percentage of work remaining, or the residual investment. Also hazard functions are present under different names in cost–time analyses. We have come across problem function, performance function, skill function, etc. Using the basic concepts of lifetime analysis, it is easy to derive the hazard function that belongs to formula (52). This is formula (55):

$$h(t) = \frac{\beta b^{[(\alpha+1)/\beta]} t^\alpha e^{-bt^\beta}}{\Gamma([\alpha+1]/\beta), bt^\beta}. \quad (55)$$

While, constant, linear, logistic and simple power hazard rates can be found using intuition or by modeling problem solving processes, the family of hazard rates described in formula (55) is beyond the imaginative powers of many of us. In order to appreciate formula (55) let’s see what happens when we take $\alpha = 1$ and $\beta = 2$ in formula (55). For the upper incomplete Γ function holds: $\Gamma(1, x) = e^{-x}$. Using this, $\Gamma(1, bt^2) = e^{-bt^2}$, so formula (55) reduces to $b\beta t$, which is a linear hazard rate. Thus, the probability density function for $\alpha = 1$ and $\beta = 2$ becomes the well-known Rayleigh curve again. Likewise, if you take $\alpha = \beta - 1$, you immediately reduce formula (55) to the Weibull hazard rate (using that $\Gamma(1, x) = e^{-x}$). This implies that you can use this family of hazard functions to fit a large variety of differently shaped cost–time relations. So the advantage is that you do not need to pick one particular model, and then see if it fits. If the models are general enough, whatever fits best will come out of a statistical analysis.

To illustrate formulas (54) and (55) further, we depicted six related plots in Fig. 26. The left-hand column contains the hazard rate, the survival function, and their product: it is the seismic cost impulse that we earlier discussed (and depicted in Fig. 5). Recall that the cost allocation function f can be found by taking the following product: $f(t) = h(t) \cdot S(t)$. The right-hand column contains the hazard, survival, and cost allocation function for the operational cost tsunami that was the consequence of the seismic IT impulse (also earlier depicted in Fig. 5). Let us compare these rates.

The first row of Fig. 26 shows us the hazard rates, or problem solving rates for both development and operations of the example IT portfolio. We used the coefficients depicted in Table 8 to instantiate formula (55), the result forms the two hazard rates. Although both curves stem from a single family, they both show rather different characteristics, not resembling any thus far known problem solving rate that we know of. The seismic hazard rate is very steep and then approaches an almost constant rate. This is achieved when most of the projects in the IT portfolio are implemented. The tsunami hazard rate on the other hand shows a much slower growth, but not linear. As can be clearly seen, none of the theoretically inferred models (Rayleigh, Parr, or damped sine) nor generalizations of these (think of Weibull curves), would have approximated our cost allocation function accurately. The reason is that the hazard rate (that determines the cost allocation function) is too far off the known models.

The second row of Fig. 26 gives us both survival rates. Again, we used the coefficients in Table 8 and instantiated formula (54) to plot both survival rates. As can be seen, the investment for the seismic IT impulse is spend fast: the left-hand plot rapidly drops to zero. The spending rate for the ensuing operational cost tsunami is

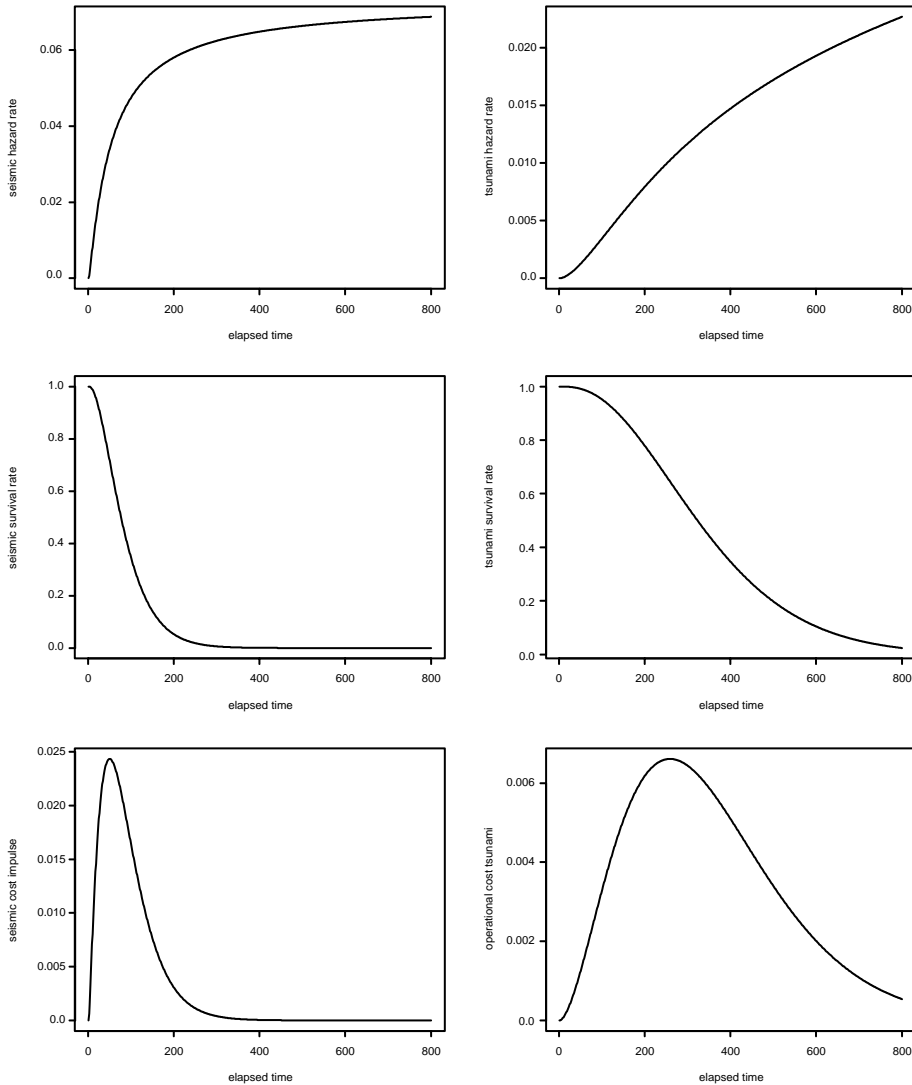


Fig. 26. The hazard rates, survival rates and their products for the seismic cost impulse and operational cost tsunami.

much slower: after a long time it is still necessary to invest. The latter rate clearly indicates that there will be a long period of investments that are followed by the initial IT expense to develop the example portfolio.

Finally, in the third row we plotted the product of the first and second row, giving us the cost allocation functions. Note that we normalized the total cost to 1 in both cases, so the shape of the operational cost tsunami deviates from how it is depicted in Fig. 5. But if you look at the scale used in both sides of the third row, you will see

that the peak of the operational cost tsunami is approximately at the fifth of the peak of the seismic cost impulse, which is also the case in Fig. 5.

12.6. Inference procedures

Despite the generality of the formulas, they are not as applicable if there is no sound inference procedure for our cost allocation function. So, there is another question that needs attention: how easy is it to infer the coefficients for formula (38), or equivalently for formula (52)?

The parameterization as given in the paper so far, is relatively intuitive for human beings, but has limited value when it comes to inference procedures. Especially when you are uninitiated in statistical analysis. With inference procedures like maximum likelihood estimates you can easily run into trouble.

After Stacy's publication in 1962 on the generalized Γ distribution, the problems with estimating the coefficients were reported on by several people [91,117,52,49]. Even with fairly large samples in the hundreds of observations, convergence problems occurred with maximum likelihood estimates. Very different sets of parameters, lead to very similar distributions. Looking back to Fig. 11 it is not hard to imagine that for a given set of data, similar curves can be found by varying the pairs (a, α) and (b, β) , leading to totally different coefficients. This does not ease inference, and several papers addressed these problems [116,98,35,74,133,134]. Prentice [98] reparameterized the distribution, and took away most inference problems: now it is easy to fit curves. First we redisplay the generalized Γ distribution:

$$g(t) = \frac{\beta}{\Gamma(k) \cdot \theta} \left(\frac{t}{\theta}\right)^{k\beta-1} e^{-(t/\theta)^\beta}$$

Prentice's alternative parameters are:

$$\mu = \log(\theta) - 2/\beta \cdot \log(\lambda), \quad \sigma = \frac{1}{\beta\sqrt{k}}, \quad \lambda = \frac{1}{\sqrt{k}},$$

where $-\infty < \mu, \lambda < \infty$, and $\sigma > 0$. The new probability density function becomes

$$f(t) = \begin{cases} \frac{|\lambda|}{\Gamma(\lambda-2)} \cdot \exp\left(\frac{\lambda - \frac{\log(t)-\mu}{\sigma} - 2 \log(\lambda) - e^{\lambda - \frac{\log(t)-\mu}{\sigma}}}{\lambda^2}\right) & \text{if } \lambda \neq 0, \\ \frac{1}{t\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(t)-\mu}{\sigma}\right)^2} & \text{otherwise.} \end{cases}$$

This formula is not intended for human interpretation, but more appropriate for computer manipulation. There are tools around that use the above formula to carry out curve fitting for the generalized Γ distribution. A tool called Weibull++ especially designed for lifetime analysis contains the above formula [105]. Also SAS [31] contains the above formula [111, Section 30.32]. For users of free software: there is a toolbox for describing ocean wave distributions that contains the generalized Γ distribution

[12]—it is not a coincidence that we associate the long operational cost waves with tsunamis. This is GNU licensed software and the Matlab [80] sources are available, which is handy when you want to tweak the code. Recall that we fit the curves using nonlinear regression as implemented in Splus [19,54,129,95]. Indeed, it is not always trivial to find good starting values, and we used techniques similar to [133,134] to find them. Using tools especially geared towards this type of analysis surely improves the ease with which you can carry out your own cost–time analysis.

12.7. The generalized F distribution

While many people resort to using restricted families of distributions for software cost estimation, this somewhat rigid practice breaks fully down when lifting from the project to the portfolio level. One of the reasons is that the projects can be quite heterogeneous. For instance, the productivity of individual programmers can vary widely. The variation in error detection (or debugging) for small programming efforts has been found to be 26 to 1. Remarkably, the subjects had the *same* programming experience [110]. These findings have been confirmed in many studies. In another study a 20 to 1 ratio of development time for programmers with the same experience [83,29,18,9,128]. Usually there will be variation in experience. Comparing low complexity efforts done by capable programmers with high complexity efforts by less capable programmers, can lead to ratio of 1:400 in productivity differences [23, p. 256]. In [23, p. 240] it was found that for large programming efforts, this effect is less pronounced, although still a variation of productivity ratios of 2–4:1 is measured. It will be clear that very heterogeneous cost–time functions on a per project basis will not be uncommon. The decreasing variation is in accord with Markowitz’s work on risk diversification: since the variance for productivity has an upper bound, the variance of the total productivity of the entire programming team will approach zero if the team size approaches infinity [79, p. 107]. And when projects become large, you need more programmers. So for *selecting* programmers for a team, in theory you could use modern portfolio theory. But given the enormous shortage of programmers, in practice there is not much choice.

Although the generalized Γ family is fairly general, we like to point out in this section, that there are even more general distributions, especially designed to deal with heterogeneous data. They are additionally used to test which less general distribution fits the data best. Therefore, we sometimes use another family of distributions that also accurately approximates cost–time data for portfolios. This family is more flexible than the generalized Γ distributions and there are described inference procedures. The disadvantage of this family of distributions is that it is relatively unknown outside the realm of failure time data analysis. We like to briefly discuss the generalized F distribution as proposed by Prentice [98]. Prentice worked on this subject supported by a grant of a cancer research center [98, p. 614]. The primary use of his distribution was to incorporate all the known failure time distributions, so that with a test you could discriminate among them. You could say that he developed mathematical tools to aim before firing. In cancer research also heterogeneous populations occur: some will die, some will be cured, some will not get the disease. We loosely compare this to the heterogeneity of programmer productivity: some are no programmers at all, some

will never finish a program, and some will spread code like a cancerous organism. The failure time could be seen as the time it takes to complete the IT development project.

A positive random variable T is said to have a generalized F distribution with μ and σ as location and scale parameters and s_1, s_2 as shape parameters, if $W = (\log T - \mu)/\sigma$ is the logarithm of a random variable having an F distribution with $2s_1$ and $2s_2$ degrees of freedom. The probability density function of W , enhancing formula (52) with respect to flexibility is

$$f(w) = \frac{(e^w s_1/s_2)^{s_1} (1 + e^w s_1/s_2)^{-(s_1+s_2)}}{B(s_1, s_2)}, \quad (56)$$

where $-\infty < \mu < \infty$, $\sigma, s_1, s_2 > 0$ and B is the beta function defined as

$$B(s_1, s_2) = \frac{\Gamma(s_1)\Gamma(s_2)}{\Gamma(s_1 + s_2)}.$$

The generalized F distribution contains many other distributions: for $\sigma = 1$ formula (56) reduces to the F distribution, if $s_i \rightarrow \infty$, for $i = 1$ or 2 formula (56) becomes the generalized Γ distribution, if $s_1 = s_2$ it reduces to the generalized log-logistic distribution, for $s_2 = 1$ it reduces to the Burr type III, and for $s_1 = 1$ to the Burr type XII distributions. Burr distributions are used in environmetrics to estimate the concentrations of chemicals such that a given percentage of species will survive [112]. Furthermore, the Γ , χ^2 , Poisson, Rayleigh, Log-normal, Log logistic, Pearson type III, Maxwell–Boltzmann and many other distributions are special cases of the generalized F distribution.

Just as with the generalized Γ distribution, formula (56) is of limited interest unless there are tools and techniques to infer the coefficients. There is an Splus package to fit both the generalized Γ and the generalized F distributions. It is called GFCURE, referring to cure rate estimation using the generalized F distribution, and stems from cancer research [94,93].

12.8. Software cost estimates are censored

In clinical trials it is often desirable to analyze the data *before* all the individuals have died. Therefore, it is common to work with so-called censored data. An observation is censored (or right censored) if the exact value of the observation is not known, but only that it is greater than or equal to this observation. For instance, when a software project is estimated to cost half a million dollar, this can be seen as censored data: it will most likely be at least \$500 000. It seems an established idea in software engineering, that it is not possible to say anything about estimates if the data is censored, that is, no actual data is available [28]. But often, we do have censored data to our avail, especially in IT portfolios, where not only finished IT projects, but also projects in progress, and IT project proposals are present. Finished IT projects provide you with life and uncensored data, ongoing and proposed ones are characterized by censored data. Note that you cannot use censor analysis, if the deviations are arbitrary, that is, sometimes too much, sometimes accurate, and sometimes too low. There is no

tradition whatsoever of IT projects being too early, or less costly than anticipated in advance [58,46–48,42,41,44]. This implies that IT portfolio data is right censored. Both in lifetime and failure time analysis a lot of research is done to support analysis in the presence of censored data sets, so we can deploy the results of this work to our advantage and analyze IT portfolios more accurately, even in the case where IT projects are not finished. We have not yet applied censor analysis in practice, since for that you need uncensored data as well. This would imply access to a body of historical data, which is hardly present in CMM level 1 organizations.

12.9. Estimating an empirical survival function

Something that is easily measurable is the spending rate for a set of correlated IT investments over time. This is used in cost uncertainty analysis for systems engineering [40] (containing a software component) to capture the cost distribution over time at an early stage. In that book [40] it is found that in many cases the total cost distribution of a systems engineering project comprises large numbers of uncorrelated cost items, so their accumulation approaches the normal distribution. This assumption enables you to estimate the accumulated cost function at an early stage. As mentioned earlier, empirical evidence shows that cost allocation functions for R&D projects, software projects, and IT portfolios are not normally distributed. This is shown in [86–88,100–103,97] and by us. So you have to do something different.

If you measure accumulated costs, you equivalently have the residual investment, which is the survival function. In lifetime data analysis, it is also not hard to measure how the survival of a population over time develops. Techniques are available to estimate from these observations the empirical survival function. If you have the survival function, you can infer the hazard rate and the cost allocation function. *Product-limit* estimates, also known as Kaplan–Meier estimates, are used in lifetime analysis to estimate empirical survival functions. Since you do not always want to wait until the entire investment is made, your data will be censored. This implies that you need to estimate the empirical survival function with censored data. For details on these methods we refer the interested reader to [67,75]. If there is enough accounting data, and other IT related management data lacks, you can use the residual investment rate to estimate the survival function. Also if quantitative IT portfolio management is consolidated within your organization, you can track the residual investment rate to check whether the survival function that was originally proposed for the IT investment is consistent with the real spending rate. This type of analysis can then signal at an early stage possible problems. In this stage, you are really getting on top of IT.

13. Conclusions

The Clinger Cohen Act of 1996 enforced the use of IT portfolio management but did not explain how this should be done in operational terms. As far as we know, this paper is the first one to describe quantitative IT portfolio management in depth. We hope that organizations in general, and in particular, the ones subject to the Clinger Cohen Act will benefit from the material presented in this paper. Most organizations are CMM

level 1 organizations. We argued that CMM level 1 organizations—who need such tools probably most urgently—can jump start quantitative IT portfolio management by compensating their lack of historical data with external benchmarks. With examples composed from actual quantitative IT portfolio management projects we illustrated our approach. Based on our findings, we were able to relate our work to an existing body of knowledge, from which we could borrow methods, insights, techniques, and tools to our advantage to solve relevant problems in quantitative IT portfolio management. Comparing the advancements made in these other areas with the gap in the software field we feel that much work, some of which has been outlined in this paper, is necessary to nurture and mature quantitative IT portfolio management. When better benchmarks and more historical data becomes available, the numerical values in our formulas will change, but presumably not their generic form. In our opinion, we developed the first iteration of a collection of formulas that form a useful basis for getting started with quantitative IT portfolio management.

Abbreviations

Below we give a lexicographical listing of the used abbreviations plus a brief clarification. Moreover we refer to formulas, tables, figures whenever appropriate. In the list you will find all but one formula. The exception is the fixed ratio equation (27) for which no abbreviation was introduced. It is expressing that the operational cost allocation per time unit is 20% of the cost allocation per time unit for building the system.

a: accumulated cost function. This is the integral of formula (38). See formula (39) for details.

adc: accumulated development costs. The formula for adc_s calculates the accumulated development costs, for a given system s over a given time frame. See formula (22). Likewise, the formula for adc_P calculates the accumulated development costs for a given IT portfolio P over a given time frame. See formula (25).

aoc: accumulated operational costs. The formula for aoc_s calculates the accumulated (minimal) operational costs, for a given system s over a given time frame. See formula (23). Likewise, the formula for aoc_P calculates the accumulated (minimal) operational costs for a given IT portfolio P over a given time frame. See formula (26).

atc: accumulated total costs. The formula for atc_s calculates the accumulated (minimal) total costs, for a given system s over a given time frame. See formula (21). Likewise, the formula for atc_P calculates the accumulated (minimal) total costs for a given IT portfolio P over a given time frame. See formula (24).

c: this symbol is overloaded. It stands for an amount of money, that is cost, but it is also a function name. It is the name we gave to the cost allocation function that generalizes all known cost allocation functions, and that we mainly use to base our cost–time analyses on. See formula (38) for elaborations.

ca: cost allocation. This formula calculates the cost allocation for a system for a given time. It consists of the operational and development cost allocations (see *cad* and *cao* below). See formula (17).

cad: cost allocation for development. This formula calculates for a given time, the cost allocation that you minimally need to develop a system. Typically you can calculate the monthly costs for development. See formula (19).

cao: cost allocation of operations. This formula calculates for a given time, the cost allocation that you minimally need to keep a system operational. You typically use this formula to calculate monthly operational costs. See formula (20).

cc: change in cost. The change in cost equation is the derivative of our proposed cost allocation equation (which is formula (38)). See formula (40).

cco: corporate cost of ownership. This formula gives you for a given IT portfolio and a given time, the corporate cost that you need to spend to own the portfolio. See formula (45).

cf: chance on failing projects. There are several related formulas. We have cf_i : the chance on late projects in the information systems industry. See formula (28) for this chance when a given amount of function points is known, and formula (29) if its development time in calendar months is known. We also have cf_o , where the subscript now denotes the outsource industry. Analogously, we have a formula for a given amount of function points: formula (30), and one if the development time is known: formula (31).

cl: chance on late projects. There are several related formulas. We have cl_i : the chance on late projects in the information systems industry. See formula (32) for this chance when a given amount of function points is known, and formula (33) if its development time in calendar months is known. We also have cl_o , where the subscript now denotes the outsource industry. Analogously, we have a formula for a given amount of function points: formula (34), and one if the development time is known: formula (35).

d: duration. This symbol is overloaded. First it stands for the duration of a project, but it is also a function, that returns the duration for development given a deployment time of a system in years. See formula (10). When subscripted it stands for the amount of calendar months expressing the amount of calendar months for an information systems project (d_i) or an outsource development project (d_o), given its size in function points. See formulas (47) and (50), respectively.

dd: development duration. This formula calculates the duration in calendar months of a development project given its cost. See formula (3). It also calculates the duration in calendar months of a development project given its staff, see formula (7). Finally, it returns the duration of a development project given a known amount of operational costs, see formula (12) for that version.

df: development fraction. This formula gives you the fraction of the total cost of ownership that is devoted to development. See formula (15).

ead: effort allocation for development. This is not a generic formula but is a specific formula tailored to an example. We calculated the effort allocation of an IT project described in the literature. The formula *ead* is depicted in Fig. 10.

f: this symbol is overloaded. First it stands for a given amount of function points. But is also a function. It is the probability distribution function $f(t)$ that belongs to our proposed cost allocation function (defined in formula (38)). See formula (52). It also stands for the probability density function known as the generalized F distribution, as stated in formula (56).

F: this is the cumulative distribution function belonging to function *f*. See formula (53) for details.

fe: failure exposure. This formula accumulates the failure chances of all the individual projects in a portfolio. All you need to know are the (estimated) durations of all the projects in the portfolio. See formula (36).

h: hazard rate. Formula (55) is the hazard rate that belongs to the probability density function that we defined in formula (38).

le: late exposure. This formula accumulates the chances of all the individual projects in a portfolio on being late. All you need to know are the (estimated) durations of all the projects in the portfolio. See formula (37).

mco: minimal cost of operation. This formula predicts the minimal cost to keep a system running given its *development* duration. See formula (11).

md: maintenance duration. This formula calculates the maintenance duration for a given cost. See formula (4). It also calculates the maintenance duration for a given staff size, see formula (8).

mrt: minimal ROI threshold. This is not a generic formula but is a specific formula that calculates minimal ROI threshold of an IT investment example. This example is summarized in Table 4. The *mrt* formula is depicted in Fig. 7.

mtco: minimal total cost of ownership. This formula calculates for a given development duration the minimal total cost of ownership over the entire life-cycle of the system. Minimal means here without functional changes. See formula (13).

nsd: number of staff for development. This formula calculates for a given duration of a development project, the number of staff needed. See formula (5).

nsm: number of staff for maintenance. This formula calculates for a given duration of a maintenance project, the number of staff needed. See formula (6).

of: operational fraction. This formula gives you the fraction of the total cost of ownership that is devoted to maintenance. See formula (16).

p: productivity. We have several formulas for different industries. We have p_i giving the productivity for developing information systems in function points per staff month for a given amount of function points. See formula (46). Analogously, formula (49) returns for a given amount of function points the productivity of outsourcers in function points per staff month.

pt: peak time. With formula (41) we can calculate the time when the peak cost needs to be allocated to a portfolio that satisfies a cost allocation function with the shape of formula (38).

pc: peak cost. This formula calculates the peak cost for a given cost allocation function of the shape proposed by formula (38). See formula (42).

r: the ratio of minimal cost of operation to development cost. See formula (14).

r_s : retirement of system *s*. This formula is the absolute-time variant of formula (9): given an initiation time and a delivery time, it calculates the retirement time for a given system. See formula (18).

rf: repayment factor. This formula calculates for a given cost allocation function of the shape as proposed by formula (38), and a given time, the amount of money that you still have to invest to develop and operate the portfolio described by the given cost allocation function. See formula (44).

S : survival function. Formula (54) is the survival function that belongs to the probability density function (52) that we derived from our cost allocation function (38).

tcd : total cost of development. We have several of these. First there is a formula, that for a given development time calculates the cost. See formula (1). Then there are several formulas that are based on function points, and diversified to industry. We have tcd_i that for a given amount of function points gives the development costs of an information systems project, see formula (48). Analogously, we have tcd_o that returns for a given amount of function points, the development costs of a project done by outsourcers, see formula (51) for details.

tc_m : total cost of maintenance. This formula calculates the total cost of a maintenance project for a given duration. See formula (2).

tc_o : total cost of ownership. This formula calculates the total cost of ownership for an IT portfolio that is described by an instantiation of formula (38). See formula (43).

$y(d)$: years that a system is in its deployment phase. With formula (9) we can calculate for a given development time, the number of calendar years that a system is in deployment.

Acknowledgements

This paper is the result of many fruitful interactions with people from industry to whom we are indebted. In particular, we would like to express our gratitude to Dr John F.A. Spangenberg (Head of IT Performance and Investment Management at ING Group⁴) for his encouragement to report on our work on quantitative IT portfolio management. We thank ING Group for underscoring the relevance of our research on quantitative IT portfolio management by a partial sponsorship. Furthermore, we thank David Faust (Director, Core Services – Global Equities Technology and Operations, Deutsche Bank AG) for his valuable comments, discussions and suggestions for improvements. We thank Hans van Vliet (VU, Amsterdam) for useful comments on the penultimate version of this paper.

References

- [1] A.J. Albrecht, Measuring application development productivity, in the Proc. Joint SHARE/GUIDE/IBM Application Development Symp., 1979, pp. 83–92.
- [2] L. Amoroso, Ricerche intorno alla curva dei redditi, Ann. Mat. Pura Appl. 21 (4) (1932) 123–159.
- [3] J. Asundi, R. Kazman, A Foundation for the Economic Analysis of Software Architectures, in: K. Sullivan (Ed.), Proc. 3rd Workshop on Economics-Driven Software Engineering Research (EDSER-3), 2001. Available via: www.cs.virginia.edu/~sullivan/edser3/asundi.pdf.
- [4] L.J. Bain, Analysis for the linear failure rate distribution, Technometrics 16 (1974) 551–559.
- [5] S. Berinato, Do the MATH, CIO Mag. (2001). Available via: www.cio.com/archive/100101/math.html.
- [6] B. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.

⁴ ING Group is a global player in the financial services and insurance industry, investing approximately 2.5 billion Euro per annum on information technology.

- [7] B.W. Boehm, The high cost of software, in: E. Horowitz (Ed.), *Practical Strategies for Developing Large Software Systems*, Addison-Wesley, Reading, MA, USA, 1975.
- [8] B.W. Boehm, Software engineering, *IEEE Trans. Comput.* C-25 (1976) 1226–1241.
- [9] B.W. Boehm, P.N. Papaccio, Understanding and controlling software costs, *IEEE Trans. Software Eng.* SE-14 (10) (1988) 1462–1477.
- [10] J. Bosch, *Design and Use of Software Architectures—Adopting and Evolving a Product-Line Approach*, Addison-Wesley, Reading, MA, 2000.
- [11] K. Brännäs, N. Nordman, Conditional skewness modelling for stock returns, Technical Report 562, Department of Economics, Umeå University, 2001.
- [12] P.A. Brodtkorb, P. Johannesson, G. Lindgren, I. Rychlik, J. Rydén, E. Sjö, WAFO—a Matlab toolbox for analysis of random waves and loads, in the Proc. 10th International Offshore and Polar Engineering Conf., 2000, pp. 343–350.
- [13] F.P. Brooks Jr., *The Mythical Man-Month—Essays on Software Engineering*, Anniversary Edition, Addison-Wesley, Reading, MA, 1995.
- [14] W.K. Brown, K.H. Wohletz, Derivation of the Weibull distribution based on physical principles and its connection to the Rosin–Rammler and lognormal distributions, *J. Appl. Phys.* 78 (4) (1995) 2758–2763.
- [15] J. Brunekreef, B. Diertens, Towards a user-controlled software renovation factory, in: P. Nesi, C. Verhoef (Eds.), Proc. Third European Conf. on Maintenance and Reengineering, IEEE Computer Society Press, Rockville, MD, 1999, pp. 83–90.
- [16] S. Butler, P. Chalasani, S. Jha, O. Raz, M. Shaw, The potential of portfolio theory in guiding software decisions, in: K. Sullivan (Ed.), Proc. 1st Workshop on Economics-Driven Software Engineering Research (EDSER-1), 1999. Available via: www.cs.virginia.edu/~sullivan/EDSER-1/PositionPapers/jha.pdf.
- [17] S. Butler, S. Jha, M. Shaw, When good models meet bad data—applying quantitative economic models to qualitative engineering judgements, in: K. Sullivan (Ed.), Proc. 2nd Workshop on Economics-Driven Software Engineering Research (EDSER-2), 2000. Available via: <http://www.cs.virginia.edu/~sullivan/edser2/shaw.pdf>.
- [18] D. Card, A software technology evaluation program, *Inform. Software Technol.* 29 (6) (1987) 291–300.
- [19] J.M. Chambers, T.J. Hastie (Eds.), *Statistical Models in S*, Wadsworth & Brooks/Cole, Pacific Grove, CA, 1992.
- [20] E.J. Chikofsky, J.H. Cross, Reverse engineering and design recovery: a taxonomy, *IEEE Software* 7 (1) (1990) 13–17.
- [21] C. Cifuentes, K.J. Gough, Decompilation of binary programs, *Software—Practice Exper.* 25 (7) (1995) 811–829.
- [22] P. Clements, L.M. Northrop, *Software Product Lines—Practices and Patterns*, Addison-Wesley, Reading, MA, 2002.
- [23] S.D. Conte, H.E. Dunsmore, V.Y. Shen, *Software Engineering Metrics and Models*, The Benjamin/Cumming Publishing Company, Inc., Menlo Park, CA, 1986.
- [24] Federal CIO Council, A summary of first practices and lessons learned in information technology portfolio management, Technical Report, Federal CIO Council, Best Practices Committee, 2002. Available via: http://cio.gov/Documents/BPCPortfolio_final.pdf.
- [25] R. D’Addario, Intorno alla curva dei redditi di Amoroso, *Riv. Italiana Statist. Econ.* Fnanza, anno 4 (1) (1932).
- [26] E.B. Daly, Management of software engineering, *IEEE Trans. Software Eng.* SE-3 (3) (1977) 229–242.
- [27] S.M. Dekleva, The influence of the information systems development approach on maintenance, *MIS Quart.* 16 (3) (1992) 355–372.
- [28] T. DeMarco, *Controlling Software Projects—Management Measurement & Estimation*, Yourdon Press Computing Series, Englewood Cliffs, NJ, 1982.
- [29] T. DeMarco, T. Lister, Programmer performance and the effects of the workplace, in: Proc. 8th Internat. Conf. on Software Engineering, ICSE-8, IEEE Computer Society, Silver Spring, MD, 1985, pp. 268–272.

- [30] T. DeMarco, T. Lister, *Peopleware—Productive Projects and Teams*, Dorset House, New York, NY, 1987.
- [31] G. Der, B.S. Everitt, *Handbook of Statistical Analyses Using SAS*, CRC Press, Boca Raton, FL, 2001.
- [32] J. Doe, C. Verhoef, *Software Product Line Migration and Deployment*, 2002, unpublished manuscript. (J. Doe is at the moment of publication still an anonymous author.)
- [33] J.B. Dreger, *Function Point Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
- [34] J.L. Elshoff, An analysis of some commercial PL/I programs, *IEEE Trans. Software Eng. SE-2* (2) (1976) 113–120.
- [35] V.T. Farewell, R.L. Prentice, A study of distributional shape in life testing, *Technometrics* 19 (1977) 69–75.
- [36] J. Feiman, N. Frey, *Migrating legacy developers to Java: costs, risks and strategies*, Technical Report, GartnerGroup, Stamford, CT, USA, 2001.
- [37] L.G. Freeman, C. Cifuentes, An industry perspective on decompilation, in: H. Yang, L. White (Eds.), *Internat. Conf. on Software Maintenance*, 1999, Printed in the short paper appendix.
- [38] H.L. Gantt, *Organizing for Work*, Harcourt, Brace & Howe, New York, 1919.
- [39] D. Garmus, D. Herron, *Function Point Analysis—Measurement Practices for Successful Software Projects*, Addison-Wesley, Reading, MA, 2001.
- [40] P.R. Garvey, *Probability Methods for Cost Uncertainty Analysis—A Systems Engineering Perspective*, Marcel Dekker Inc., New York, 2000.
- [41] R.L. Glass, *Computing Calamities—Lessons Learned From Products, Projects, and Companies that Failed*, Prentice-Hall, Englewood Cliffs, NJ, 1998.
- [42] R.L. Glass, *Software Runaways—Lessons Learned from Massive Software Project Failures*, Prentice-Hall, Englewood Cliffs, NJ, 1998.
- [43] R.L. Glass, A new answer to how important is mathematics to the software practitioner? *IEEE Software* 17 (6) (2000) 135–136.
- [44] R.L. Glass, *ComputingFailure.com—War Stories from the Electronic Revolution*, Prentice-Hall, Englewood Cliffs, NJ, 2001.
- [45] United States Government, *Clinger Cohen Act of 1996 and Related Documents*, 1996. Available via: www.c3i.osd.mil/org/cio/doc/CCA-Book-Final.pdf.
- [46] The Standish Group, *CHAOS*, 1995. Retrievable via: standishgroup.com/visitor/chaos.htm (Current February 2001).
- [47] The Standish Group, *CHAOS: A Recipe for Success*, 1999. Retrievable via: www.pm2go.com/sample_research/chaos1998.pdf.
- [48] The Standish Group, *EXTREME CHAOS*, 2001. Purchase via: <https://secure.standishgroup.com/reports/reports.php>.
- [49] H.W. Hager, L.J. Bain, Inferential procedures for the generalized gamma distribution, *J. Amer. Statist. Assoc.* 65 (1970) 1601–1609.
- [50] B. Hall, Year 2000 tools and services, in: *Symp./ITxpo 96, The IT Revolution Continues: Managing Diversity in the 21st century*. GartnerGroup, 1996.
- [51] M. Hanna, Maintenance burden begging for a remedy, *Datamation*, April 1993, pp. 53–63.
- [52] H.L. Harter, Maximum-likelihood estimation of the parameters of a four-parameter generalized gamma population for complete and censored samples, *Technometrics* 9 (1967) 159–165.
- [53] G. Hector, *Breaking the Bank—the Decline of BankAmerica*, Little, Brown & Company, Boston, MA, 1988.
- [54] S. Huet, M.-A. Gruet, *Stastical Tools for Nonlinear Regression—A Practical Guide with S-Plus Examples*, Springer, Berlin, 1996.
- [55] IBM Corporation, *Transaction Processing Facility—General Information*, 4.1 Edition, 1993. Available via: www-3.ibm.com/software/ts/tpf/images/gtpgim00.pdf.
- [56] W. Jarvis, New approaches to phasing of resources, in the *Proc. 34th Annual Department of Defense Cost Analysis Symp.* 2001. Available via: www.ra.pae.osd.mil/adodcas/Presentations%202001/RSCAllocation.PDF.
- [57] W. Jarvis, E. Pohl, Program execution efficiency and resource phasing, in the *Proc. 32nd Annual Department of Defense Cost Analysis Symp.*, 1999. Available via: www.ra.pae.osd.mil/adodcas/slides/jarvis33.pdf.

- [58] J. Johnson, Chaos: the dollar drain of IT project failures, *Appl. Dev. Trends* 2 (1) (1995) 41–47.
- [59] C. Jones, *Programming Productivity*, McGraw-Hill, New York, 1986.
- [60] C. Jones, *Assessment and Control of Software Risks*, Prentice-Hall, Englewood Cliffs, NJ, 1994.
- [61] C. Jones, *Applied Software Measurement: Assuring Productivity and Quality*, 2nd Edition, McGraw-Hill, New York, 1996.
- [62] C. Jones, *Patterns of Software Systems Failure and Success*, International Thomsom Computer Press, Boston, MA, 1996.
- [63] C. Jones, *Estimating Software Costs*, McGraw-Hill, New York, 1998.
- [64] C. Jones, *The Year 2000 Software Problem—Quantifying the Costs and Assessing the Consequences*, Addison-Wesley, Reading, MA, 1998.
- [65] C. Jones, *Software Assessments, Benchmarks, and Best Practices*, Information Technology Series, Addison-Wesley, Reading, MA, 2000.
- [66] N. Jones, Year 2000 market overview, Technical Report, GartnerGroup, Stamford, CT, USA, 1998.
- [67] J.D. Kalbfleisch, R.L. Prentice, *The Statistical Analysis of Failure Time Data*, Wiley, New York, 1980.
- [68] C.F. Kemerer, Reliability of function points measurement—a field experiment, *Comm. ACM* 36 (2) (1993) 85–97.
- [69] C.F. Kemerer, B.S. Porter, Improving the reliability of function point measurement: an empirical study, *IEEE Trans. Software Eng.* SE-18 (11) (1992) 1011–1024.
- [70] T.A. Kirkpatrick, Research: CIOs speak on ROI, *CIO Insight*, 1(11) (2002). Available via: www.ciainsight.com, results of questionnaire available via: common.ziffdavisinternet.com/download/0/1396/0110_rio_research.pdf.
- [71] D. Kodlin, A new response time distribution, *Biometrics* 23 (1967) 227–239.
- [72] A. Krause, M. Olson, *Basics of S and S-Plus*, 2nd Edition, Springer, Berlin, 2000.
- [73] L. Lamport, How to Tell a Program from an Automobile, in: J. Tromp (Ed.), *A Dynamic and Quick Intellect—Liber Amicorum in honor of Paul Vitanyi's 25-year jubilee*, CWI, 1996, pp. 77–79. Available via: research.microsoft.com/users/lamport/pubs/automobile.pdf.
- [74] J.F. Lawless, Inference in the generalized gamma and log gamma distributions, *Technometrics* 22 (1980) 409–419.
- [75] J.F. Lawless, *Statistical Models and Methods for Lifetime Data*, Wiley, New York, 1982.
- [76] T.C. Lethbridge, Priorities for the education and training of software engineers, *J. Systems Software* 53 (1) (2000) 53–71.
- [77] B.P. Lientz, E.B. Swanson, *Software Maintenance Management—A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*, Addison-Wesley, Reading, MA, 1980.
- [78] B. Londeix, *Cost Estimation for Software Development*, Addison-Wesley, Reading, MA, 1987.
- [79] H.M. Markowitz, *Portfolio Selection—Efficient Diversification of Investments*, Cowles Foundation For Research in Economics and Yale University, vol. 16, Wiley, New York, 1967 (3rd printing).
- [80] W.L. Martinez, A.R. Martinez, *Computational Statistics Handbook with MATLAB*, CRC Press, Boca Raton, FL, 2001.
- [81] S. McConnell, *Rapid Development*, Microsoft Press, Redmond, WA, 1996.
- [82] F.W. McFarlan, Portfolio approach to information systems, *Harvard Business Rev.* 59 (5) (1981) 142–150.
- [83] H. Mills, *Software Productivity*, Little Brown, 1983.
- [84] S.N. Mohanty, Software cost estimation: present and future, *Software—Practice Exper.* 11 (1981) 103–121.
- [85] M. Mustatevic, Automation of function point counting, Master's Thesis, University of Amsterdam, Programming Research Group, 2000 (in Dutch).
- [86] P.V. Norden, Curve fitting for a model of applied research and development scheduling, *IBM J. Res. Dev.* 2 (3) (1958).
- [87] P.V. Norden, Useful tools for project management, in: B.V. Dean (Ed.), *Operations Research in Research and Development* Wiley, New York, 1963.
- [88] P.V. Norden, Useful tools for project management, in: M.K. Starr (Ed.), *Management of Production*, Penguin Books, 1970, pp. 71–101.

- [89] United States General Accounting Office, Information Technology Investment Management—A Framework for Assessing and Improving Process Maturity, 2000. Available via: www.gao.gov/special.pubs/10.1.23.pdf.
- [90] F.N. Parr, An alternative to the Rayleigh curve model for software development effort, *IEEE Trans. Software Eng.* SE-6 (3) (1980) 291–296.
- [91] V.B. Parr, J.T. Webster, A method for discriminating between failure density functions used in reliability predictions, *Technometrics* 7 (1965) 1–10.
- [92] M.C. Paulk, C.V. Weber, B. Curtis, M.B. Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley Publishing Company, Reading, MA, 1995.
- [93] P.Y. Peng, GFCURE—An S-PLUS Package for Parametric Analysis of Survival Data with a Possible Cured Fraction, 1999. Available via: www.math.mun.ca/~ypeng/research/gfcure/.
- [94] Y. Peng, K.B.G. Dear, J.W. Denham, A generalized F mixture model for cure rate estimation, *Statist. Med.* 17 (1998) 813–830.
- [95] J.C. Pinheiro, D.M. Bates, *Mixed-Effects Models in S and S-PLUS*, Springer, Berlin, 2000.
- [96] M.E. Porter, *Competitive Strategy—Techniques for Analyzing Industries and Competitors*, The Free Press, New York, 1980.
- [97] P.H. Porter, Revising R&D program budgets when considering funding curtailment with a Weibull Model, Master's Thesis, Air University, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, USA, March 2001.
- [98] R.L. Prentice, A log gamma model and its maximum likelihood estimation, *Biometrika* 61 (1974) 539–544.
- [99] R.L. Prentice, Discrimination among some parametric models, *Biometrika* 62 (3) (1975) 607–614.
- [100] L.H. Putnam, A macro-estimation methodology for software development, in the Proc. IEEE COMPCON 76 Fall, IEEE Computer Society Press, Silver Spring, MD, 1976, pp. 138–143.
- [101] L.H. Putnam, A general empirical solution to the macro software sizing and estimating problem, *IEEE Trans. Software Eng.* SE-4 (4) (1978) 345–361.
- [102] L.H. Putnam, W. Myers, *Measures for Excellence—Reliable Software on Time, Within Budget*, Yourdon Press Computing Series, 1992.
- [103] L.H. Putnam, R.W. Wolverton, Quantitative management: Software cost estimating, in the Proc. IEEE Computer Society First Computer Software and Applications Conf. (COMPSAC 77), IEEE Computer Society Press, Silver Spring, MD, 1977, pp. 8–11.
- [104] A.W. Rathe (Ed.), *Gantt on Management: Guidelines for Today's Executive*, American Management Association, New York, 1961.
- [105] ReliaSoft Corporation, *Life Data Analysis Reference*, rev. 2001 Edition, 2001. www.weibull.com/lifedatawebcontents.htm.
- [106] J. Reutter, Maintenance is a management problem and a programmer's opportunity, in: A. Orden, M. Evens (Eds.), 1981 National Computer Conf., AFIPS Conf. Proc., vol. 50, AFIPS Press, Arlington, VA, 1981, pp. 343–347.
- [107] P. Rosin, E. Rammler, The laws governing the fineness of powdered coal, *J. Inst. Fuel* 7 (31) (1933) 29–36.
- [108] H. Rubin, *Worldwide Benchmark Report for 1997*, Pound Ridge, NY; Rubin Systems, Inc., 1997.
- [109] H. Rubin, M. Johnson, What's going on in IT?—summary of results from the worldwide IT trends and benchmark report, 2002, Technical Report, Metagroup, 2002. Available via: metricnet.com/pdf/whatIT.pdf.
- [110] H.W. Sackman, W.J. Erikson, E.E. Grant, Exploratory experimental studies comparing online and offline programming performance, *Comm. ACM* 11 (1) (1968) 3–11.
- [111] SAS Institute Inc., *SAS/QC User's Guide*, Version 8, 8.2 Edition, 2000.
- [112] Q. Shao, Estimation for hazardous concentrations based on NOEC toxicity data: an alternative approach, *Environmetrics* 11 (5) (2000) 583–595.
- [113] H.M. Sneed, C. Verhoef, Business process reengineering, in: J.J. Marciniak (Ed.), *Encyclopedia of Software Engineering*, 2nd Edition, Wiley, New York, 2001, pp. 83–95.
- [114] SPSS. *SPSS 10.0 Syntax Reference Guide*, 1st Edition, Prentice-Hall PTR, Englewood Cliffs, NJ, 1999.
- [115] E.W. Stacy, A generalization of the gamma distribution, *Ann. Math. Statist.* 33 (3) (1962) 1187–1192.

- [116] E.W. Stacy, Quasimaximum likelihood estimators for two-parameter gamma distributions, *IBM J. Res. Dev.* 17 (1973) 115–124.
- [117] E.W. Stacy, G.A. Mihram, Parameter estimation for a generalized gamma distribution, *Technometrics* 7 (1965) 349–358.
- [118] T.D. Steiner, D.B. Teixeira, *Technology in Banking—Creating Value and Destroying Profits*, Irwin/McGraw-Hill, New York, 1990.
- [119] P.A. Strassmann, *Information Payoff—The Transformation of Work in the Electronic Age*, The Information Economics Press, New Canaan, Connecticut, USA, 1985.
- [120] P.A. Strassmann, *The Business Value of Computers—An Executive’s Guide*, The Information Economics Press, New Canaan, Connecticut, USA, 1990.
- [121] P.A. Strassmann, The policies and realities of CIM—lessons learned, in the Proc. 4th Armed Forces Communications and Electronics Association Conf., AFCEA, Fairfax, VA, USA, 1993, pp. 1–19.
- [122] P.A. Strassmann, *The Politics of Information Management—Policy Guidelines*, The Information Economics Press, New Canaan, Connecticut, USA, 1995.
- [123] P.A. Strassmann, *The Squandered Computer—Evaluating the Business Alignment of Information Technologies*, The Information Economics Press, New Canaan, Connecticut, USA, 1997.
- [124] P.A. Strassmann, *Information Productivity—Assessing the Information Management Costs of US Industrial Corporations*, The Information Economics Press, New Canaan, Connecticut, USA, 1999.
- [125] P. Tate, The big spenders, *Information Strategy*, December–January 1998–1999, pp. 30–37.
- [126] A.A. Terekhov, C. Verhoef, The realities of language conversions, *IEEE Software* 17(6) (2000) 111–124. Available at <http://http://www.cs.vu.nl/~x/cnv/s6.pdf>.
- [127] W.L. Trainor, Software: from satan to saviour, in the Proc. National Aerospace and Electronics Conf., 1973.
- [128] J. Valett, F.E. McGarry, A summary of software measurement experiences in the software engineering laboratory, *J. Systems Software* 9 (2) (1989) 137–148.
- [129] W.N. Venables, B.D. Ripley, *Modern Applied Statistics with S-PLUS*, 3rd Edition, Springer, Berlin, 1999.
- [130] C. Verhoef, Towards automated modification of legacy assets, *Ann. Software Eng.* 9 (2000) 315–336. Available at <http://www.cs.vu.nl/~x/ase/ase.html>.
- [131] W. Weibull, A statistical theory of the strength of materials, in: Proc. No. 151. The Royal Swedish Institute of Engineering Research, Stockholm, 1939.
- [132] S. Wolfram, *Mathematica Book*, 4th Edition, Cambridge University Press, Cambridge, 1999.
- [133] J.Y. Wong, Simultaneously estimating the three parameters of the generalized gamma distribution, *Microelectron. Reliability* 33 (15) (1993) 2225–2232.
- [134] J.Y. Wong, Simultaneously estimating with ease the three parameters of the generalized gamma distribution, *Microelectron. Reliability* 33 (15) (1993) 2233–2242.
- [135] F. Wright, *Computing with MAPLE*, Chapman & Hall/CRC, London, 2001.
- [136] E. Yourdon, *Death March—The Complete Software Developer’s Guide to Surviving ‘Mission Impossible’ Projects*, Prentice-Hall, Englewood Cliffs, NJ, 1997.