Balancing Time-to-Market and Quality in Embedded Systems

Pieter van der Spek* and Chris Verhoef

Department of Computer Science, VU University Amsterdam, Amsterdam, The Netherlands

Received 5 May 2010; Revised 29 November 2012; Accepted 29 November 2012, after one or more revisions
Published online in Wiley Online Library (wileyonlinelibrary.com).
DOI 10.1002/sys.21261

ABSTRACT

Finding a balance between the time-to-market and quality of a delivered product is a daunting task. The optimal release moment is not easily found. We propose to use historical project data to monitor the progress of running projects. From the data we inferred a formula providing a rough indication of the number of defects given the effort spent thus far (effort-to-defect formula). Furthermore, we provide a worst case bound to the allowed number of residual defects at the end of a project in order to achieve the required level of quality. For this purpose we slightly modified a well-known reliability growth model by Bishop and Bloomfield. It turned out that the software in Philips’ MRI scanners has a defect rate of 1 per 1175 device-years of observation. This coincides with the second highest safety integrity level (SIL3) as defined in the IEC 61508 standard. The highest level (SIL4) is only attainable by applying redundancy. Finally, we combine the effort-to-defect formula with the reliability growth model to monitor the progress of a project and to determine when the required level of quality will be reached. We show that a common fault distribution, the Rayleigh model, is not necessarily the best model for predicting the number of residual defects in the system. Using a well-known data analysis approach called exploratory data analysis (EDA) we obtained an alternative model based on the Normal curve. We have evaluated the Rayleigh model and our model based on the Normal curve at Philips Healthcare MRI. The Normal curve predicts defects over time better than the Rayleigh model in the case of Philips Healthcare MRI. Furthermore, time series models (ARIMA) are also useful for accurately describing the defect trend, but are not suitable for long-term predictions. Finally, cost estimation models (COCOMO) lack the predictive capabilities of models fitted on the data using EDA. Their capabilities are limited compared to a model derived from data which reflects the constitutional knowledge of the actual realization of systems within a specific company. However, they can still be used to advantage when there is limited or no data available to use as a basis for lifting from gut feel to order of magnitude. © 2013 Wiley Periodicals, Inc. Syst Eng 17

Key words: time-to-market; quality; release readiness; effort estimation; failure rate; defect trend; Rayleigh model; Normal curve; time series; ARIMA; COCOMO; MRI; embedded systems

Contract grant sponsors: Dutch Ministry of Economic Affairs under the BSIK program. This research also received partial support from the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.004.405 EQUITY: Exploring Quantifiable Information Technology Yields and via contract 638.003.611 Symbiosis: Synergy of managing business-IT-alignment, IT-sourcing and offshoring success in society.

*Author to whom all correspondence should be addressed (e-mail: pvdspek@cs.vu.nl).
If we deliver on time, but the product has defects,
we have not delivered on time.

—Philip Crosby [1989]'

1. INTRODUCTION

Denver’s new international airport was to be the pride of the Rockies, a wonder of modern engineering, but software glitches in an automated baggage-handling system forced Denver International Airport to sit empty 9 months after airplanes were to fill these gateways and runways [Gibbs, 1994]. In 2003, a local outage that went undetected due to a “race condition” [Poulsen, 2004] in General Electric Energy’s monitoring software caused a power outage in North America, affecting 40 million people in the United States and 10 million people in Canada, with an estimated financial loss of $6 billion [Bindewald, 2004]. Finally, in another, infamous case, six known accidents involved massive radiation overdoses due to software-related accidents in a computerized radiation therapy machine called the Therac-25 [Leveson and Turner, 1993].

Due to the complex system context of embedded-software applications, defects can cause life-threatening situations, delays can create huge costs, and insufficient productivity can impact entire economies [Ebert and Jones, 2009]. Providing better estimates plays a pivotal role in releasing high-quality products on time. However, it is notoriously difficult to balance the time-to-market and quality of a delivered product [Rakitin, 1999]. In the case of the Therac-25 a combination of inadequate management, overconfidence in software qualities, less-than-acceptable software engineering practices, and unrealistic risk assessments culminated in the disastrous outcome. In the competitive commercial software market, companies feel compelled to release software the moment it is ready. The optimal release moment is not easily found. It treads the line between releasing poor-quality software early and high-quality software late. As Crosby already noted [Crosby, 1989], if software is released while it still contains significant defects, it has not been delivered on time. The task of delivering on time gets even more difficult when the company operates in a regulated market in which quality is of the utmost importance. Finding a sound answer to the question, “Is the software good enough to release now?” is critical to a company’s survival. That answer is sometimes based on the opinion of experts, but historical data can put this judgment on firmer footing.

Repositories such as time tracking systems, versioning systems, and defect tracking systems are useful to aid in managing software projects. Practitioners and researchers increasingly recognize the potential benefit of mining this information to monitor the performance of the project [D’Ambros et al., 2008]. As Myers stated in his seminal work The Art of Software Testing: “The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section” [Myers, 2004]. In other words, instead of using an existing model and trying to squeeze the data into the model, extracting information from the repositories available in a company and creating a model based on the data, would provide an invaluable source of information on various aspects of a system such as its reliability in the case that we are looking into the number of errors.

However, despite some heroic efforts from a small number of research centers and individuals [Fenton et al., 2007; Li et al., 2006; Nagappan et al., 2006; Ostrand and Weyuker, 2002; Schroeder and Gibson, 2006; Zimmermann et al., 2009], there continues to be a lack of published empirical data relating to the quality of realistic commercial systems [Ebert and Jones, 2009; Fenton and Ohlsson, 2000; McConnell, 1997]. An exception is a study in which the number and rate of recalls and safety alerts of automated external defibrillators (AED) [Shah and Maisel, 2006] are determined. They state that:

Although AEDs are complex medical devices designed to function during life threatening emergencies, little is known about their reliability.

So, also others observed that surprisingly not much is known about the reliability of life-critical systems. The AED study is based on publicly available information. They use information from the database of the Food and Drug Administration (FDA). The FDA data is limited compared to the data that we use in this article. Some argue even that the FDA provides only a small fraction of the actual numbers of failures [Gilk, 2009].

The results reported in this article are based on empirical data obtained from Philips Healthcare MRI, which develops large software-intensive systems for healthcare applications. These systems combine high-end hardware components with a large amount of software into a single, complex product. The complexity is not just in the sheer size of the physical machine or the amount of code required to operate it, but also in the multitude of connections and interactions between the two. The magnetic resonance imaging (MRI) system is not unique with respect to its complexity. A modern car, for instance, may contain dozens of microprocessors, with software controlling aspects such as engine ignition, pollution control, security, air-conditioning, car radio, or even the seat position [Stevens, 1998; Stoermer et al., 2006]. In fact, value creation in cars is primarily determined by embedded software, resulting not only in increased cost and complexity, but also in increased potential defects from embedded software [Ebert and Jones, 2009].

Another example of a complex system is a modern television [van Ommering, 2002]. A television consists of mechanics, electro-optics, electronics, and software. It typically takes 100 software engineers 2 years to build the software for a high-end television. Many more comparable complex systems exist, such as copiers, wafer steppers used in the semiconductor industry, airplanes, and military systems. MRI scanners, therefore, are a typical example of systems engineering in practice. These and other complex systems are also called software-intensive systems [Broy, 2006], and their development and evolution require interdisciplinary teams consisting of experts in hardware, software, and several other disciplines.
The historical boundaries between hardware and software are vanishing [Krikhaar et al., 2009]. Due to the increasing importance of software in these complex software-intensive systems, more and more failures are a result of problems in the software instead of the hardware of the system. The examples at the start of this article clearly show that software is the Achilles’ heel of software-intensive systems. Systems engineering in the twenty-first century is concerned with both hardware and software engineering. The software for Philips’ MRI scanners is not confined to the physician’s desktop, but is embedded in the system, installed on EPROMs, FPGAs, and so on, and controls every part of their behavior. Quality of the software embedded in these software-intensive systems is therefore a major concern and is entitled to our full and undivided attention, and is the main topic of this article.

In this article, we provide a contribution to the body of empirical knowledge by describing a number of results from a quantitative study of defect management data of a major commercial system. We inferred from the data that the measured defect rate of the software in Philips’ MRI scanners is once per 1175 device-years. In the IEC 61508 standard this failure rate is defined as Safety Integrity Level 3 (SIL3), the second highest possible safety level specified in this standard. The highest safety integrity level, SIL4, is only attainable when redundancy is built into the system, for instance, by duplicating critical components, by combining different safety measures, or by implementing multiple instances of the same system.

These results have been obtained using a well-known data analysis approach called exploratory data analysis (EDA) [Tukey, 1977]. EDA differs from classical analysis in that it takes all the available data as a starting point and infers models from this data. This is different from classical analysis which takes a model as a starting point and imposes the model on the data. The classical analysis approach runs the risk of ignoring characteristics of the underlying data as these might not have been considered when the model was first developed. In practice, if the underlying assumptions of the model are unknown or untested, the validity of the scientific conclusions becomes suspect [Filliben and Heckert, 2012]. Many EDA techniques, on the other hand, make little or no assumptions. They present and show all of the data as is. Obviously, this has the downside that the specific model obtained through EDA will most likely not be directly usable in a different context. The general approach, however, is reusable in different circumstances.

Thus, we make no claims about the generalization of the specific numbers mentioned in this article, but in time they can form part of a broader picture. The method described in this article, however, is more generally applicable and is relevant in other companies and for different types of projects. We hope that our study will stimulate others to also use a rich information set. In that way we believe that a clear picture of the level of safety and reliability ensues which is attainable for life-critical software-intensive systems. These kinds of numbers help in the communication between management and IT giving hands and feet to the general gut feeling that “less testing means less quality.” In that way they create a negotiation means between the two fields which is still not very common due to the inability of these two worlds to effectively communicate with one another.

In short, the main contributions of this work are as follows.

- We give a detailed description of the historical data from a defect tracking system and time tracking system for a large commercial embedded system.
- We propose a method showing how to use this data to monitor the progress of a project. Furthermore, we predict when a certain level of quality is met. For this purpose we did the following.
  - We inferred a formula providing a rough indication of the number of defects given the effort spent thus far (effort-to-defect formula).
  - We modified the reliability growth model by Bishop and Bloomfield [1996] to obtain a worst case bound on the number of residual defects in the source code.
  - We fit a model based on the Normal curve to predict when a certain level of quality, defined as the number of residual defects, is reached.
- We show that the often advocated Rayleigh model is not always the appropriate model to predict the trend with which defects are submitted during a project.
- We show that defect trends over time can be modeled by well-known time series models (ARIMA), but they cannot be used for predicting long-term deadlines.
- We show that cost estimation models (COCOMO) lack the predictive capabilities of models fitted on the data using EDA.

**Organization of the article**

In Section 2, we provide an overview of the systems engineering field of MRI. In Section 3, we describe the data we have used for our analysis. In Section 4, we start with the first step of our method and show how the data was used to estimate the expected number of defects that must be solved over the course of the project. Section 5 details the second step of our method and shows how the data was used to calculate the level of quality. In Section 6, we describe how we combine these two estimations and estimate the time-to-market. Section 7 discusses threats to validity. Section 8 provides a brief overview of related work. In Section 9, we summarize our method and provide concluding remarks.

**2. SYSTEMS ENGINEERING AND MRI**

Development of an MRI system requires a multidisciplinary design team with competences in areas such as physics, electronics, mechanics, material science, software engineering, and clinical science. All the disciplines have to work together effectively on many aspects of system design. Adding to the complexity is that the work is carried out in geographically spread locations and in different time zones. Figure 1a shows an example of a modern MRI scanner. In clinical practice, MRI is used to distinguish pathological tissue (such as a brain tumor) from normal tissue.
As noted in the introduction, many systems of systems such as cars and airplanes, or medical devices such as an MRI are growing more and more dependent on software to complete even basic tasks such as starting the engine of a car. As noted among others by Boehm [2006], as well as others [Gibson, 2005; Pandikow et al., 2000], in response to the increasing criticality of software within systems and the increasing demands being put onto twenty-first century systems, systems and software engineering processes will evolve significantly over the next two decades. Software and systems engineers will need to work closely together to make these software-intensive systems operate flawlessly and, in cases where human lives are involved, highly reliably.

The market for medical devices, such as MRI scanners, is a so-called “regulated market.” Various regulating agencies exist to formulate and enforce the rules in such markets and take appropriate actions when they are broken. For instance, the market for healthcare applications in the United States is regulated by the FDA. Users trust medical devices because of a belief that FDA regulations ensure safety. We have come a long way since the incidents surrounding the Therac-25. Medical devices are now subject to strict requirements and the companies manufacturing them are required to have a risk management system in place as detailed in ISO 14791 [International Organization for Standardization (ISO), 2007]. Furthermore, they are required to follow the requirements for a management system for the design and manufacture of medical devices as detailed in ISO 13485 [ISO, 2003].

If a manufacturer fails to comply, the consequences can be severe. Moreover, not only should the company create and maintain these documents, it should also make sure that the sources are stored properly and remain fully accessible for 15 years. In the regulations documents [Krikhaar et al., 2009; U.S. Food and Drug Administration (FDA), 1997], a case is described where a manufacturer did not possess the software source code of a part that behaved erratically in the field. The component supplier had withdrawn without granting the manufacturer the right to access the source code. Therefore, there was no practical way to maintain the software. In the end, the manufacturer was forced to recall the product, and all known units were collected and destroyed. The example illustrates the realities of failing to comply with the regulations. Maintaining a rigorous quality control process is, therefore, simply business-critical to many organizations. Such recalls have the potential to severely impact the sales of a company as illustrated by a series of recalls of automobiles by Toyota Motor Corporation which occurred at the end of 2009 and start of 2010. Soon thereafter, Toyota reported a nearly 10% drop [Carty, 2010] in their car sales in the United States.

Surprisingly, there are no strict quality requirements to which companies have to commit. Although international standards like IEC 61508 [International Electrotechnical Commission (IEC), 1999] are in place for “electrical/electronic/programmable electronic safety-related systems,” no such standards are enforced for medical devices. This standard defines four levels of safety for complex systems. Table 1 shows the safety integrity levels as described by this standard. The reader is referred to van der Spek [2010] for an example where this standard is applied to calculate the reliability of the Maeslantkering, a storm surge barrier in the Nieuwe Waterweg in the Netherlands.

Even though the reliability of hardware components is more or less understood to follow the well-known bathtub curve, researchers also recognize that failures in modern software-intensive systems can only partially be attributed to hardware components. For example, research into recalls of pacemakers has shown that between 1990 and 2000, firmware errors accounted for about 40% of the half million devices recalled [Ebert and Jones, 2009]. Other studies report soft-

Table 1. Safety Integrity Levels—Demand and Continuous Model

<table>
<thead>
<tr>
<th>Safety Integrity Level</th>
<th>Probability of dangerous failure on demand, average (Low Demand mode of operation)</th>
<th>Probability of dangerous failure per hour (Continuous mode of operation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIL4</td>
<td>$\geq 10^{-5}$ to $&lt; 10^{-4}$</td>
<td>$\geq 10^{-7}$ to $&lt; 10^{-8}$</td>
</tr>
<tr>
<td>SIL3</td>
<td>$\geq 10^{-4}$ to $&lt; 10^{-3}$</td>
<td>$\geq 10^{-8}$ to $&lt; 10^{-7}$</td>
</tr>
<tr>
<td>SIL2</td>
<td>$\geq 10^{-3}$ to $&lt; 10^{-2}$</td>
<td>$\geq 10^{-7}$ to $&lt; 10^{-6}$</td>
</tr>
<tr>
<td>SIL1</td>
<td>$\geq 10^{-2}$ to $&lt; 10^{-1}$</td>
<td>$\geq 10^{-6}$ to $&lt; 10^{-5}$</td>
</tr>
</tbody>
</table>

Figure 1. Impression of the Philips Healthcare MRI system. (top) Philips 3T Achieva system. The small opening (called the bore) is clearly visible. The casing around it holds the superconducting magnet as well as the liquid helium to cool it. (bottom) The result of a scan showing a patient’s neural system. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]
3.1. Characteristics of the Historical Data

Philips Healthcare MRI uses a single database to store problems discovered in the software both before and after release. We have therefore opted for using the term defects as defined by the SWEBOK and do not discriminate between whether the problem was detected before or after release.

Defects

We collected the defect information from eight projects as stored in ClearQuest. These defects were mostly submitted to the defect management system as part of test phases of the projects. Each project consists of several phases. The last phases of the project comprise different stages of testing including alpha, beta, and integration tests. Finally, once the product is deemed of sufficient quality it is installed on a limited set of systems for field testing. The bulk of the defects are submitted in these phases and are discovered as part of the rigorous testing procedure carried out by the test engineers of the system verification department. Defects discovered in earlier stages are usually resolved without submitting an official defect report.

The oldest defect in this dataset dates back to December 1995. The most recent defect dates are from November 2009. In total we collected a little over 28,000 defects for this period of nearly 15 years. It should be noted that this number does not provide a complete overview of all defects submitted over this period, as this number does not include defects from projects running in parallel to the eight projects we examined. Moreover, this number only refers to the work done on the software of the MRI scanners. The defect information for hardware-related development is tracked separately.

Version management

We also examined the data from the version management system, ClearCase. Each time an item is added to the version management system, or when it is updated or deleted, this is recorded. Such an action is more commonly referred to as a checkin. A single checkin usually consists of a combination of these actions carried out on a set of files. We tried to match this information with the projects we wanted to study. However, this proved impossible as development related to a specific project is sometimes done on a separate branch, sometimes on different branches (e.g., different subprojects on different branches), and sometimes multiple projects are carried out on a single branch. Although it is possible to separate the projects based on timestamps, the results of such an exercise are not very accurate.

Time registration

A third source of information was the data from the time management system. For recent projects we turned to the current time management system and exported the information for the projects we were investigating. For older projects the data was not readily available. In these cases, only an Excel sheet was available with an export of the data from an older time management system.

In order to compare the projects, we aggregated this data at the project level. Splitting the data at other levels, such as in...
groups for development, marketing, testing, and so on, proved impossible as this was not recorded for all projects. Moreover, over the past 10 years, the manner in which hours were recorded and rules on how to do this have changed frequently, making any kind of analysis on lower levels of granularity difficult.

Field problem reports
We also collected problems reported with systems in active use, called field problem reports (FPRs). An FPR records a problem reported with an MRI system in use by a hospital, clinic, or research facility. They are not tied specifically to an individual project, but are entered into the defect management system when they occur. For the most part they contain information similar to regular defects. The procedure with which they are handled, however, differs significantly as they first need to be examined before resulting in one or more defects which will be solved in a specific project.

As FPRs are not linked to a specific project, unlike defects, version management, and time registration, we extracted all FPRs from the repository. In total we collected 12,710 FPRs, the oldest of which dates from September 1994 and the most recent dates from December 2009. While this might seem like a large number, only a handful of these FPRs concern incidents which could potentially have harmed a patient. Most of them are not at all safety-related such as “popup window covers messages,” “print issue,” or translation errors. For our analysis we only used safety-related incident reports.

Installed base
Finally, we collected information on the installed base of Philips Healthcare MRI scanners which is recorded when they are shipped and which is updated when changes are made to the hardware or software configuration. The installed base of MRI scanners are all those scanners which are in active use since 1984. Our information on the installed base includes information on the shipping date, the original hardware and software release, the decommisioning date (if applicable), and many other data.

3.1.1. Defect Management Data
We will describe the data from the defect management system in a little more detail. We will also show its limitations. For managing the defects reported during the development process as well as for problems reported for systems being used by hospitals, Philips Healthcare MRI uses IBM Rational ClearQuest. This system allows one to register information on a defect and monitor the activities related to that defect. A typical defect report in ClearQuest contains at least the following elements.

- **Headline:** one-line description of the defect, written by the submitter.
- **Project:** the ClearQuest-project; this usually relates to one of the running or completed projects at Philips Healthcare MRI.
- **Problem Type:** either “Problem Report” or “Change Request.”
- **Responsible Group:** the (development) group that is responsible for the system-part in which the problem is found.
- **SubSegment:** for software: the Functional Cluster\(^1\) in which the problem occurred; for other groups “NA” (indicating this information is not available).
- **Version:** the version in which the defect occurred. This is either a released software version or a label of the current development stream.
- **Severity:** the impact of the defect on the product. In theory, the severity can only change when it was completed incorrectly or when a work-around is implemented which reduces the impact. There are five severity levels.
  1. **Critical:** must be fixed before product release; Safety, crashes, hang ups.
  2. **Major:** must be fixed before product release; Performance, memory.
  3. **Average:** should be fixed in this release, though does not block release.
  4. **Minor:** documentation.
  5. **Enhancement:** corresponds to a “Problem Type” of type “Change Request.”
- **Priority:** the priority of the defect. This field is used to indicate the importance of the defect for the project (see also Severity).

Moreover, for each action a log records who handled a defect and when this happened. Each defect goes through several steps, as shown in Figure 3. Starting from the state “Submitted” the defect will go through several phases before reaching an endstate like “Duplicate,” “Validated,” or “RejectAccepted.” An initial assessment of the defects’ description will result in an immediate reject, or an accept. In some cases the defect will be forwarded to another project, for instance when the related software is the responsibility of another depart-

**Figure 2.** Number of operational Philips Healthcare MRI scanners since 1984.

\(^1\)The Functional Clusters form a high-level decomposition of the software archive, similar to more well-known concepts such as subsystems.
ment. A rejected defect will always get a second opinion. If the second reviewer agrees, the rejection will be accepted. Otherwise, it will be resubmitted, as indicated by the arrow going from “Rejected” back to “Submitted.”

An accepted defect will be investigated in more detail as to what the cause of the problem might be. Once an underlying root-cause has been identified, it will be assigned to a developer. The developer will open the defect and implement a solution. After having resolved the problem, the software will undergo rigorous verification and validation to determine whether the problem has actually been solved.

Although this process is rigorously structured, the defect data confronted us with several problems with respect to its usability. Large organizations, such as Philips Healthcare MRI, adjust themselves to stay ahead of their competition and to be able to face new challenges. This means that they make changes in the organization as well as to the tools they use to support their (development) processes. In general, these kinds of changes make the analysis of historical data difficult or even impossible at least when trying to analyze the data from before the change.

In our case the effects were limited, although we had to make an effort to clean up the data so that comparisons over a longer period became possible. Between 2004 and 2005 Philips Healthcare MRI migrated from ClearDDTS to its successor ClearQuest. While all defect reports were imported into ClearQuest, their history could not be imported directly. Fortunately, as part of the transition, Philips added an attachment to each defect imported into ClearQuest containing the complete history from ClearDDTS. This enabled us to reconstruct the history of these reports. For our analysis we focused our attention on those parts of the data which were least subject to any organizational change and could be reliably reconstructed. In practice, this meant we used the severity information, the timestamps related to submission of a defect and state transitions, and information from the Project field so we could group reports by project.

3.2. Characteristics of the Projects

We have examined the historical data for eight software projects at Philips Healthcare MRI to illustrate how to balance time-to-market with quality. We took these eight projects since all of them primarily dealt with development of and updates to the user interface parts of the system.

The oldest project dates back to the start of this millennium and ran approximately from May 2000 until September 2004. This project included a major overhaul of the user interface. Subsequent projects added new clinical applications and updates. The most recent project in our dataset started in May 2009. Next to that, we also investigated the entire FPR database and the complete information on the installed base. These two repositories are not linked to a specific project.

During this time, the total amount of source code grew from around 3.5 million lines of code to nearly 9 million. The bulk of this code, around 2.5 million lines, has been written in C, since 2004 the share of C++ and C# code has been steadily increasing, while the total amount of C code has remained more or less stable. Furthermore, various other technologies such as Java and XML are used.

If we zoom in on the data of a single project, we get a picture like Figure 4. This figure shows the following information. Figure 4a shows the number of different people involved over the lifetime of the project. We counted the unique usernames occurring in the defect management data and version management history in order to create this graph. On average, about 300 different people are involved over the entire lifetime of a project from the start of gathering requirements to the end where the product is delivered. About 100 of them are actively involved in software development activities. Thus, this graph shows the portion of people involved in development and testing, but excludes other project roles such as management or people working on requirements and documentation.

Figures 4b and 4c show the number of checkins and the number of defects being submitted per month. Unlike the other three graphs, the graph displaying the number of checkins does not show a regular distribution. Although we cannot be sure as to why this is, there are various reasons for this particular pattern. The checkin behavior of developers usually is related to whatever developers are doing at that moment and is not necessarily related to the number of defects they are working on.
Furthermore, the number of checkins required for resolving a single defect varies greatly depending on the kind of changes which are required. Also, a refactoring results in an entirely different checkin pattern compared to a bugfix. As we are looking for a predictable pattern which can be captured in a theoretical model, the irregular distribution of the checkins over time does not provide a good starting point.

The number of defects being submitted per month shows a far more regular distribution compared to the checkins. There is only one potential outlier, notably around the 18th month. However, this is easily explained from the development practices at Philips Healthcare MRI. In this case, this was December and usually a lot of people will be on holiday during these last few weeks of the year, resulting in a drop in the number of defects being submitted. Thus, the outlier is presumably real data and not an artifact. Other than that, the information coming from ClearQuest seems a good candidate for capturing it in a theoretical distribution relating to fault behavior.

Figure 4d shows the number of hours spent over the entire life-cycle project as recorded in the time tracking software. As this is a separate system, it is very hard to link the two data sources on detailed levels such as individual people. At best, we can compare the number of hours spent to statistics collected from the defect tracking system and this is precisely what we have done as we will show in the next section.

4. ESTIMATING NUMBER OF DEFECTS

The most common basis used for estimating is the so-called “expert opinion” [Boehm and Fairley, 2000]. It seems to be a built-in human characteristic to prefer precision over accuracy. By this is meant that humans prefer a specific, single value that pretends to certainty (and which the estimator “knows” is probably wrong) over a range value that almost certainly includes the most probable value and hence is correct, but which retains some level of uncertainty. The results from our study allow for range estimates, which usually are far more valuable than a single point estimate [Boehm and Fairley, 2000].

To predict the defect trend and estimate when the required level of quality is reached, we first need a rough estimate of the total amount of defects which will be found in the software. It is likely that there will be a relation between the size of a project and the amount of defects that will materialize in the course of the project. Therefore, we used information on the project size from five of the selected projects. For the oldest project, as well as for projects running in other countries, this information was not available to us.

In particular, we collected the amount of time spent on these projects in terms of person-hours, and the amount of defects identified in those projects. The results are shown in Figure 5. On the vertical axis this figure shows the number of defects. The horizontal axis depicts the number of hours spent. The open circles show the number of defects submitted after spending a certain number of hours for five of the projects in our set of data. Subsequently, we investigated whether there is a relation between these two numbers.

Upfront, we suspected a polynomial relation between the two variables. In that case, it would be possible to describe the relation using the following formula. This formula describes the relation between the effort, measured in hours, and the number of defects found. In the remainder of this article, we will refer to this relation as the effort-to-defect formula.

\[ d = h^a \] (1)
The symbols used in this formula are defined as follows:

\( d \): the cumulative amount of defects up to a certain point in time
\( h \): the cumulative effort up to the same point in time
\( a \): a constant relating the number of hours to the number of defects.

On plotting the data, another possible relation between the two variables became apparent. The data shows a distinct S-like pattern characteristic for a logistic distribution, for which the following general formula holds:

\[
d = \frac{c}{1 + ae^{bh}}
\]

(2)

Here \( c \) represents the limiting value of the output; \( a \) is the number of times that the initial population must grow to reach \( c \); the coefficient \( b \) determines whether the function will be increasing or decreasing. The logistic equation is a common model for population growth. In our case, the logistic function is interpreted as describing the growth of the number of defects depending on the number of hours spent in a project.

We used the statistical programming environment R [R Development Core Team, 2008] to compute a fit for the available data. In both cases we were able to find a reasonable fit. A problem with the logistic distribution, however, is that it is dependent on the expected peak value represented by the value of \( c \). Unfortunately, this is also a value which we do not know yet, although we could choose an appropriate value by analogy, that is, we could set \( c \) to a value which we found for an older project of similar size. Moreover, even though we calculated a value for \( c \) by calculating the best fit for all of the projects in our dataset, the maximum value differs depending on the size of the project and so the logistic curve and its parameters differ significantly between different projects. Thus, despite the fact that the logistic model describes the data more appropriately, the predictive value of this model for a single project is less accurate.

The polynomial model, however, fits the data less accurately on the micro level but has more predictive power. One possible reason could be that the latter model only has one parameter, and the logistic model has three coefficients to fit. Overfitting might be in play here. We did not investigate this further as our goal was to find some rough predictive indicator that relates effort to defects.

Figure 5 depicts the value of the coefficient of the effort-to-defect relation over the lifetime of the five projects which were completed at the time of writing and for which the time management information was available. In particular, for each project we determined the number of hours spent after 1 month and the number of defects submitted in the first month. Together with Equation (1), this gives us a value for the coefficient of the effort-to-defect relation. Similarly, we determined the number of hours spent in the first 2 months and the number of defects submitted in these 2 months. We repeated this for each month of each project.

Unfortunately, we could not calculate this value for all of the projects for which we analyzed the defect data, as some of these projects were too old and the time management information was no longer available. Other projects are still under way, such that we cannot calculate this value for them either.

The three outliers, indicated by the open circles, invariably show the value of the coefficient of the effort-to-defect relation in the first month of the project. Thus, the relation between the number of hours and the number of submitted defects in the first month of a project is not typical for the rest of the project.

The red line shows the geometric mean which is 0.62. Thus, the average value for the coefficient of the effort-to-defect relation over these four projects is 0.62. We also determined a nonlinear least-squares estimate of \( a \) which is 0.70, shown by the dashed blue line. From these numbers we conclude that by using a coefficient of 0.6 and 0.7 for the effort-to-defect relation we get a lower and upper bound on the amount of defects that will be found. For example, if a project is estimated at 80,000 person-hours, based on these numbers, the amount of defects submitted will be somewhere between

\[ d = \ldots \]
80000^{0.6} = 874 and \( d = 80000^{0.7} = 2705 \). This range is used as best- and worst-case estimate in the subsequent steps of the approach.

In short, with the effort-to-defect relation it is possible to obtain a rough estimate on the total number of defects that will be found over the lifetime of a project, depending on the size of the project in terms of the number of hours expected to be spent over the lifetime of that project. This estimate provides an insight into the defects that will be found during the lifetime of that project, and monitors its progress as we will see in Section 6. However, first we quantify the required level of quality in the next section.

5. ESTIMATING SOFTWARE QUALITY

Once software is perfect, it does not wear out [Stevens, 1998]. At least in theory, software does not require maintenance, or support. However, this does not factor in hardware failures with which the software should be able to cope, or even faults in critical pieces of the software. Faults in the software tend to be immediate and total [Stevens, 1998]. While hardware usually groans before it comes to a grinding halt, the effects of a software defect are often immediate and catastrophic.

Software engineers for safety-critical systems have known for years that for a large, complex system to have a mean time to failure (MTTF) of 100,000 hours, it must be subject to at least that many hours of testing [Butler and Finelli, 1991]. This was first observed by Adams in 1984 in a study of the bug history of IBM mainframe operating systems [Adams, 1984], and has been confirmed by extensive empirical investigations since. The first theoretical model explaining it was published in 1996 by Bishop and Bloomfield [Bishop and Bloomfield, 1996, 2002], who proved that under standard assumptions this would be the worst-case behavior. Brady et al. [1999] tightened this result by showing that, up to a constant factor, it was also the expected behavior.

The formula for predicting the reliability of a system, presented by Bishop and Bloomfield, is

\[
\lambda \leq \frac{Nd}{eT} \quad (3)
\]

The symbols have the following meaning.

- \( \lambda \): the expected dangerous failure intensity of the whole program after time \( T \).
- \( N \): the total number of safety-critical faults in the software.
- \( d \): the number of failures that have occurred before a fault is fixed. In the case of safety-related failures, we expect all faults to be fixed immediately so we assume that \( d = 1 \) [Verhoef, 2007b].
- \( T \): the period over which the system has been used or changed without changes to the software or hardware. This includes changes such as the modification of the system’s configuration.
- \( e \): the exponential constant (2.718...).

Note that this formula gives a worst-case bound (for a given operational profile), but there are empirical arguments that the best-case defect rate could be no more than an order of magnitude better than the bound [Bishop, 2005]. However, we cannot use this formula as such. The formula calculates the dangerous fault intensity for a single system. Our data, however, originates from a variety of systems being used on a daily basis in hospitals and clinics around the world. Therefore, we need to determine the reliability level, \( N \), and quality level, \( \lambda \), based on the data at hand.

5.1. Determining the Reliability

The data available to us only records safety-related incidents since 1994, so we also limit the MRI scanners included in this calculation to MRI scanners which have been taken into use since that time. The total amount of MRI scanners in active use in 2010 is even larger as some scanners taken into operation before 1994 are still in use today.

From this data we calculated the failure intensity of the system. The failure intensity \( \lambda \) is alternatively described as the level of safety of a system. The lower the failure intensity is, the higher the level of safety of the system. Section 2 introduced the IEC 61508 standard. This standard defines four safety integrity levels. The highest level, SIL4, is only attainable when redundancy is built into the overall system, as is necessary in the case of the Maeslantkering to be able to reach the required SIL.

From the available data we can fairly easily compute the SIL of the MRI scanners. The SIL is defined as the number of safety-related incidents over the lifetime of the system. In our case, this is the total lifetime of all MRI scanners taken into operation since 1994 and the number of incidents reported since that time. About 7000 MRI scanners had a total operational lifetime of 47,000 device-years, during which 40 incidents were reported. This is equivalent to \( 9.715 \cdot 10^{-8} \) incident per device-hour. Overall, we can conclude that the software for the currently operational MRI scanners has a level of SIL3. This is the second highest band defined by the IEC 61508 standard and the highest attainable level of safety without building redundancy into the system.

Usually, determining the SIL of software is more difficult than the calculation that we carried out in retrospect. The IEC 61508 prescribes and recommends best practices to improve the chance that some SIL is reached. Of course, this is no guarantee that this level is indeed reached as turned out to be the case with the Maeslantkering. An important issue that we do not further discuss herein is therefore: how did Philips Healthcare MRI develop their software and systems? Apparently their practice led to these top-notch quality levels, so from a systems engineering viewpoint it is rather interesting to find out how. This paper is about how to balance time-to-market and the necessary quality levels and in the next section we deal with determining the latter.

5.2. Determining the Quality Level

The model by Bishop and Bloomfield helps us to determine an acceptable level of quality. Although there are no requirements on the defect rate of medical devices, we can safely assume that there should not be any safety-related incidents during the operational lifecycle of a medical device. As each safety-critical fault is a potential failure [Stevens, 1998], this
number directly relates to the SIL we calculated above. The formula devised by Bishop and Bloomfield provides us with a defect count in terms of a bound on the residual number of potentially dangerous faults that are allowed to remain in the hardware and software system without affecting the quality of the system. This formula provides a worst-case bound, while the best case is no more than an order of magnitude better [Bishop, 2005]. Rewriting the formula gives

\[ eT \lambda \leq N \leq 10eT \lambda \] (4)

It should be noted that there are several implicit assumptions in this formula. First, it is acceptable to leave several safety-related faults in the software unresolved as long as this does not affect the quality level. Thus, it is possible that these faults are carried over to new releases. As long as the number of safety-related faults remains at most \( N \), this is acceptable. However, this also assumes that all safety-related faults in the software are known, which is of course an unrealistic assumption. Thus, in order to attain a high level of quality in practice, it is advisable to resolve more safety-related faults in the software than would be required based on this formula. As we will see, this is precisely what Philips Healthcare MRI does and this philosophy is likely one of the pillars under the high level of quality of their systems.

The value for the quality level \( \lambda \) depends on the value we choose for \( T \). However, if we use these values directly, we would not be calculating the number of residual dangerous defects allowed in a single software-system, but for the total set of operational MRI scanners. We therefore need to compensate for this fact and adapt the formula by Bishop and Bloomfield. In particular, we need to adapt the formula to take into account that both the period \( T \) and the dangerous failure intensity \( \lambda \) are counted over the entire set of operational systems.

We already discussed how to calculate \( \lambda \) by dividing the sum of all dangerous failures, which occurred during a certain period, by the total number of systems in operation. Similarly, we calculate the operational period \( T \) by dividing the operational lifetime of all systems by the total number of systems in operation during that same period. This yields

\[ e \cdot \frac{O}{S} \cdot \frac{L}{S} \leq N \leq 10e \cdot \frac{O}{S} \cdot \frac{L}{S} \] (5)

\[ \frac{eOL}{S} \leq N \leq \frac{10eOL}{S} \]

The symbols are defined as follows.

- \( O \): the sum of all operational periods \( T \) for all MRI scanners
- \( L \): the sum of all failures over the period \( O \)
- \( S \): the number of MRI scanners which have been operational in this period.

In the remainder of this section, we will show how to determine the operational period of a system, in this case an MRI scanner. Due to the restrictions placed on the original formula by Bishop and Bloomfield, this is not a straightforward activity.

5.2.1. Determining the Operational Period

As mentioned in the previous section, our data goes back to 1994 in the case of the field problem reports, while the information on the installed base goes back even further. The formula of Bishop and Bloomfield defines \( \lambda \) as the expected dangerous failure intensity of the whole program after time \( T \). This is the period over which the system has been used or changed without changes to the functionality of the software or hardware. This period is more formally called the “operational profile” of a system. Musa [1993; Musa et al., 1996] defines an operational profile as follows: “An operational profile is a quantitative representation of how a system will be used.”

An operational profile is a quantitative description of how the system is being used in practice. It is based on the customer profile, the user profile, and the system mode profile. A customer is the individual, group, or organization that is purchasing the (software) system. A user may be different from the customer, and is the person, group, or institution that operates the system. Finally, the system mode is the way that a system can operate [Musa et al., 1996].

A stable operation profile, thus, implies that these three components do not change. While refactoring potentially affects the quality of the system, it does not change the way a system can operate, and so it does not alter the system’s operational profile. In the case of Philips Healthcare MRI, this means that over the period \( T \) the MRI scanner and its software stay the same and are used in the same way.

We can safely assume that the MRI scanners have been used in the same way. However, it is less obvious that the system itself has remained the same. In the case of calculating the SIL of the system, this is not an issue as this provides an overview of the overall reliability of the system and does not rely on an operational profile as defined by Musa. Determining the allowed number of residual faults, however, does require a closer look at what constitutes a reasonably constant operational profile.

As mentioned in Section 3.2, the software grew from 3.5 million lines of code in 2000 to 9 million in 2009. So the software more than doubled in size, but how does this affect the operational profile? As shown in Figure 7, the software did not grow gradually, but rather grew in various bursts. For example, between February 2003 and December 2004 the software grew from about 4 million lines of code to 6 million. It remained at that level until March 2006, when it grew to about 7 million lines, after which a series of smaller increments resulted in an overall size of about 8 million lines. This observation is important as it allows us to identify a period in which the functionality implemented in the software did not change much and thus exhibits a stable operational profile. It might still be the case that the software underwent some refactorings, but these do not change the functionality only how it has been implemented.

In 2004 the first version (release 1) of the current hardware and software platform was introduced; a major update (release 2) to this platform was released in 2006. For these two platforms a total of 10 failures have been reported while approximately 3000 systems were running on this hardware and software platform. So, we could make a choice and pick...
some time frame that presumably displays a reasonably constant operational profile. Maybe a slightly different time frame shows dramatically different results. Therefore, we opted for a simulation approach. In order to determine the allowable residual errors based on Equation (5), we randomly selected 10,000 periods that were at least a year apart. We took one year since the hardware and software of an MRI scanner is at least frozen for a year (and often much longer). For these 10,000 time frames we determined the defect rate and the number of residual defects based on Equation (5).

Thus, this experiment provides us with an insight of the impact of the operational period on the allowed number of residual safety-critical faults. While the software is being actively developed, the software in use on systems in hospitals will remain stable as required by the formula of Bishop and Bloomfield. As the data used for this formula relates to failures of systems in active use at hospitals and clinics, this experiment should show us precisely what is an acceptable operational period.

The results are shown in Figure 8. We used two plots for the two releases and one for a combined view. Each plot displays horizontally the number of safety-critical faults that Equation (5) predicts given the time frame, the amount of device-years within that time frame, and the amount of safety-related incidents within that same time frame. In plot 1 we observe that in a few cases between 0.14 and 1.4 residual faults are allowed in order to maintain the high level of quality. The median number of safety-critical faults for release 1 is between 0.05 and 0.5, or, in other words, less than 1 residual fault. In plot 2 we find that the boundaries are more strict with a median number of safety-critical faults between 0.01 and 0.1. In plot 3 we do not discriminate between the individual releases. However, from this figure we note that this does not change the median number of safety-critical faults which remains between 0.01 and 0.1.

Our simulation experiment shows that the number of allowable residual dangerous defects has the same order of magnitude for arbitrary time frames of at least a year. Therefore, for instance taking all device-years since 1994 as the operational period in Equation (5) does not yield a vastly different estimate than had we known the exact constant operational profiles of each individual MRI scanner. So even though the software has changed since 1994, if we aggregate over the entire set of MRI scanners, this change has been very gradual, resulting in a more or less stable operational profile.

5.2.2. Determining the Number of Residual Faults
From this assumption we calculate the allowable number of residual faults, \( N \), for the software of Philips’ MRI scanner.
The total number of device-years of operation since 1994 is 47,000 for a total of 7000 MRI scanners. In this time, 40 safety-related incidents have been reported. If we enter this data in the modified Bishop and Bloomfield formula, this gives us a bound on the residual number of potentially dangerous faults that are acceptable to reside in the code of the MRI scanners which are currently operational:

\[
\frac{e \cdot 47000 \cdot 40}{7000^2} \leq N \leq \frac{10 \cdot e \cdot 47000 \cdot 40}{7000^2}
\]

\[
0.1043 \leq N \leq 1.043
\]

So in this situation the allowable number of residual faults, which can potentially result in dangerous failures of the system, is at most 1. This conforms to the state of the practice at Philips Healthcare MRI where the software is not released until each and every critical and major defect is removed before the software is released. It should be noted that the total number of faults in the code can be higher than that. Bishop [2005] states that dangerous faults are only 3% of the total number of faults while Jones [2000] holds this to be 1%.

Using the effort-to-defect formula defined in Section 4, we estimate a bound on the total number of defects that will be found over the course of a project. A project of 80,000 person-hours will generate at worst 2705 defects. Taking the pessimistic view of Bishop, about 81 of these defects are dangerous. At the end of the project, no more than 1 of these should remain in order to end up with a product which has a sufficient level of quality. In other words, around 97% of the defects should be solved in a product with the desired quality. Around 97% of the defects should be solved in a product with the desired quality characteristics which means that this marks the date at which the product is ready to be shipped. So, only one thing remains now and that is how to balance this level of quality with the time-to-market. In the next section we will examine a model which helps to make this trade-off and help determine the date.

6. ESTIMATING TIME-TO-MARKET

There is some confusion about the business value of quality, also outside the software development context. On the one hand, there are those who believe that it is economical to maximize quality. On the other hand, there are those who say: “I’d rather have it wrong than have it late. We can always fix it later” [Slaughter et al., 1998]. While this is true, the cost of fixing a defect before release is much lower compared to fixing it after having released the product. Moreover, problems with a released product can have grave repercussions. In the case of Toyota, it comprised a 10% decline in market share. For medical devices, it means putting the life of a patient at stake and may be the difference between making a profit and bankruptcy. So it is an important question how to balance time-to-market with the quality of the system.

Firms frequently tend to ship a product as soon as it is good enough; similarly, given that fixing bugs takes time, they may fix only enough bugs for their product to keep up with the perceived competition. For example, Microsoft takes the perfectly pragmatic approach of prioritizing bugs by severity, and as the ship date approaches the bug categories are allowed to slip. More severe bugs are allowed through into the product if they are discovered at the last minute and if fixing them is nontrivial [Anderson, 2005]. Philips Healthcare MRI, on the other hand, only releases a new version of its software when all critical and major defects have been resolved.

However, the trend of bug discovery throughout a project aids in making the project more predictable such that we know when the product is actually good enough. As a result of the inherent complexity of the software development process, developers have plenty of opportunities to make errors [Putnam and Myers, 1991]. Most of these errors are removed before delivery through self-checking by analysts and programmers, design reviews, walkthroughs, inspections, module testing, and integration testing. However, finding and subsequently solving each and every bug is not feasible. At some point a trade-off must be made between the estimated number of defects which are likely to still remain in the code and the time when the product can be shipped. In other words, at some point we need to balance the time-to-market with the quality.

A possible way to estimate the number of defects still in the system is by examining the trend with which defects are found in a project. Motivated by engineering applications, Weibull [1951] suggested a distribution that has proved to be of seminal importance in reliability. The Weibull distribution has been widely used to model the failures of many materials, and in numerous other applications. Among others, Burgess and Leffey [2001] found that project data could be fitted well by Weibull distributions. Others, such as Pillai and Nair [1997], found the predictive capabilities of the Rayleigh model (a variant on the more general Weibull model) limited and instead proposed a gamma distribution. Putnam and Myers [1991] proposed to use the Rayleigh curves for IT projects. Thus, while the specific distribution which models the defect trend differs from case to case, there is no doubt that these distributions are useful for modeling the defect trend.

In the remainder of this section, we will first examine the Rayleigh model and its applicability to the data we obtained from Philips Healthcare MRI. Subsequently, we will try a different model, based on the Normal distribution, and compare the results. Next, we assess a time series approach which is advocated by some as an alternative to models such as the Rayleigh and Normal distributions. Finally, we evaluate a cost estimation model which is widely used in both academia and industry. The approaches evaluated in this study comply with the recognized need for predictive models that are computationally efficient, comprehensible, and easy to apply [Lessmann et al., 2008].

6.1. Rayleigh Model

Over the lifetime of a project, errors are committed, detected, and fixed. Only a few people are working in the early stages of the project and the number of defects submitted per month of the project as a whole is low. As more people are assigned to the project, the rate of committing errors will increase. Similarly, as the project nears completion, the number of people tails off, and the number of errors they make declines.
Another contributing factor is that near the end of the project the focus will shift from adding new features to bugfixing.

Various authors have suggested that this trend can be modeled by a Rayleigh model [Putnam and Myers, 1991]. Moreover, it has been suggested that this model can be used to predict the number of defects over the lifetime of a project and thus can be used to predict when for instance 97% of the total number of defects have been resolved. Thus, this model should help to determine when the desired level of quality, as depicted by the number of defects still present in the product, has been reached. An example of a defect rate and matching Rayleigh model can be seen in Figure 9. The line shows the number of defects submitted each month, while the gray surface represents the total number of defects submitted. Using the general trend of the curve and knowing the total surface that is supposed to be below the curve from Equation (1), we can estimate when 97% of the defects have been submitted.

### 6.1.1. Applying the Rayleigh Model

As the Rayleigh model should be generally applicable, we have used it to predict the behavior of the defect trend in the case of Philips Healthcare MRI as well. Although we ran into several problems, contrary to others [Staron and Meding, 2008], we think it can be a useful tool for monitoring the defect trend and determine when a certain quality level will be reached. However, predicting this moment at the start of the project seems to be infeasible. We will discuss these two issues below.

A Rayleigh curve is described using the formula

\[
 f(t) = at e^{-bt^2} \tag{6}
\]

The two parameters \(a\) and \(b\) are determined from historical data and are supposed to be more or less generally applicable throughout an organization. However, this assumes that all projects are more or less the same. Whether this is a realistic assumption is questionable. In fact, in the case of Philips Healthcare MRI we observed that this is not the case. Figure 10 shows Rayleigh curves modeled using the data of three different projects. It depicts the Rayleigh curve with month zero being the start of the project’s test phase. It is seen that the three projects behave in a completely different fashion even though they are primarily concerned with the same part of the software archive as explained in Section 3.2. The projects differ in their duration and the maximal amount of defects submitted in a month. In other words, each project is unique and a Rayleigh curve based on data from other projects cannot be used to predict the defect trend in another project.

Furthermore, this model assumes that defects will be submitted from the start of the project. At Philips Healthcare MRI, however, the process is different and defects will not be submitted until the start of the test phase of a project. Figure 11 shows the total number of defects being submitted per month from the start of a project. The Rayleigh curve (red) can only be fitted over the last part of the graph which relates to the start of the earliest test phase of the project. In other words, in the case of Philips Healthcare MRI, this model is only usable for predicting the test phase of a project.
Although the Rayleigh model can be used to monitor the progress of a project and provide insight into the quality of a software system, it is far from perfect. Figure 12 shows why this is so. It presents an \( f/a \) plot [Eveleens and Verhoef, 2009] of the forecasts from the Rayleigh model compared to the actual date of the completion of the projects. The vertical axis shows the deviation of the forecast in percentages of the actual. The gray band shows a 1% margin around the 1.0 line. This line corresponds to forecasts equal to the actual. The gray band helps to quickly see which forecasts are within 1% of the actual.

The horizontal axis shows the amount of historical data used to make the prediction. That is, at point 0.2 the defects submitted in the first 20% of the project were used to predict the completion date. The forecasts at this point differ from the actual completion date by 50% or more. This changes to an overestimation of the actual time to ship by 20% or more when more data is used. Although the forecasts converge to the actual, even when all the available data is used (i.e., when a forecast is calculated from the data of a completed project), the forecasts still are an overestimation of the actual date.

The Rayleigh model obviously grossly overestimates the duration of a project in the case of Philips Healthcare MRI. The assumption of Putnam and Myers and others [Burgess and Lefley, 2001; Putnam and Myers, 1991] that the Rayleigh model is adequate is thus proven wrong as it overestimates the remaining number of defects. In general, we think that the methodology of using a model such as the Rayleigh model is useful. However, the actual model might very well differ for different companies. In the case of Philips Healthcare MRI we found an alternative model which provides more accurate forecasts compared to the Rayleigh model as we will discuss next.

6.2. Normal Model

The Rayleigh model assumes a relatively steep upward trend early in the project, while the trend in the later stages of the project shows a similar decline in the number of submitted defects. This decline is usually much steeper than the one modeled by the Rayleigh model. In fact, there is a certain symmetry to the upward trend in the beginning of the project and the decline at the end of the project. The Rayleigh model, therefore, structurally underestimates the number of defects in the first half of the test phase. Subsequently, in the second half of the test phase, it overestimates the remaining number of defects.

Based on our observations from the available data, we suspected a different model would produce more accurate forecasts earlier in the project. The symmetric distribution of the defects over time might be better modeled by a model based on the Normal curve instead of one based on the Rayleigh curve. A Weibull curve could solve some of the problems of the Rayleigh curve since there are more degrees of freedom in that model. However, this more general model is also asymmetrical. Therefore, we opted for a symmetrical model.

In the past the Normal model was also called the distribution of errors [Lévy, 1924]. Although later on more sophisticated models were found to be more appropriate, as mentioned above, in some cases the Normal model is still more adequate than newer techniques. The general formula of a Normal curve is

\[
f(t) = e^{a^2+bte^c}
\]  

Starting from this equation, we statistically fitted the appropriate values for \( a \), \( b \), and \( c \) similar to the way we did this for the Rayleigh model. Figure 13 shows the results when we apply the Normal model to the same project as before. When we fit the Normal model over the last part, we see a much better fit to the available data compared to the fit of the Rayleigh model in Figure 11.

Also in this case, the same limitation applies as the Normal curves for different projects are very different from each other as shown in Figure 14. Thus, also in the case of the Normal model, it is not possible to come up with a single set of values
for \(a, b,\) and \(c\) which can be used equally well to model the behavior of older projects and predict the behavior of future projects.

Finally, the \(f/a\) plot in Figure 15 shows that the forecasts based on the Normal model are more accurate compared to the Rayleigh model, at least in later stages of the project when more data is available. Moreover, they converge to the actual more quickly, making this model more useful in practice.

### 6.3. Forecast Quality

Although the \(f/a\) plots give insight into the bias of the forecasts and the quality of the forecasts, it is difficult to compare different \(f/a\) plots to each other. Fortunately, it is also possible to quantify this difference. One such tool is the Estimating Quality Factor (EQF) [DeMarco, 1982].

DeMarco developed this tool to quantify the deviation between the projection of the estimators and the actual. The EQF, therefore, shows the quality of forecasts and allows for comparisons. A higher EQF means a better forecast. Verhoef [2007a] and Lister [2002] posit that an EQF value on the order of 10 is good, while the norm is about 4. This means that usually forecasts are about 1/4 or 25% off. The EQF and \(f/a\) plot together provide complete insight into the quality of the forecasts.

We computed the EQF for the various projects based on the forecasts provided by the Rayleigh model and those provided by the Normal model. We compared the median of these two collections of EQF values to each other as the distribution of the EQF values does not need to be symmetrical. Since the median value is not significantly influenced by large EQF values, it is, therefore, more robust than the mean value [Eveleens and Verhoef, 2009]. A median EQF value of 4 means that in 50% of the cases, the maximal deviation is 1/4 or 25%. The EQF is

\[
EQF = \frac{\text{Area under actual value}}{\text{Area between forecast and actual value}}
\]

\[
= \frac{\int_0^t a dt}{\int_0^t (a - e(t)) dt}
\]

\[
= \frac{\int_0^t 1 dt}{\int_0^t |1 - e(t)/a| dt}
\]

(8)

The variables in this equation are defined as follows.

- \(a\): the actual value, that is, the actual duration of a project
- \(ta\): the time when the actual is known
- \(e(t)\): the value of the forecast at time \(t\) (0 \(\leq t \leq ta\)).

Figure 16 shows an example EQF calculation. For this project, in total four forecasts were made: at \(t = 0.2, t = 0.4, t = 0.6,\) and \(t = 0.8\). Similar to the forecasts from our two models, we do not have a forecast at \(t = 0\). We assume that the forecast at that point is the same as the forecast at \(t = 0.2\). For
Table II. Median of EQF Values for Forecasts Using Rayleigh and Normal Model

<table>
<thead>
<tr>
<th></th>
<th>Rayleigh</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>using all forecasts</td>
<td>2.7</td>
<td>3.0</td>
</tr>
<tr>
<td>excluding forecasts based on only the first 20%</td>
<td>2.2</td>
<td>5.1</td>
</tr>
<tr>
<td>excluding forecasts based on only the first 40%</td>
<td>2.2</td>
<td>11.6</td>
</tr>
</tbody>
</table>

Each forecast, we calculate the area between the forecast and the actual. Of the first, we find the difference to be $1 - 0.5 = 0.5$. The duration of this forecast is $0.2 - 0 = 0.2$, which means that the area between the first forecast and the actual of this project is $0.5 \times 0.2 = 0.1$. We repeat the calculation for the other areas and find the sum of the areas to be $0.1 + 0.08 + 0.04 + 0.02 = 0.34$. As the area under the actual is 1, we find the EQF value $= 1/0.34 = 2.9$. The reader is referred to Evleens and Verhoef [2009] for a more extensive description of the EQF in combination with the $f/a$ plot. Table 2 shows the median EQF values for the two models. It is seen that the overall forecasting quality of both models is comparable, although the Normal model is slightly better. However, if we exclude forecasts based on only the first 20% or even the first 40% of the project data, the EQF values for the Normal model become much larger than those for the Rayleigh model. This confirms the picture provided by the $f/a$ plots. For 50% of the projects, the forecasts based on the Normal model have a maximal deviation of only $1/1.16 = 0.86$. In the case of the Rayleigh model, this is $1/2.2$ or $45.4\%$, over a factor 5 difference. Moreover, already at 40% the forecasts based on the Normal model are slightly better than the norm for EQF values. Even though these numbers are based on forecasts of only five projects, these EQF values suggest that while early forecasts based on either model are not good, the forecasts based on the Normal model become more accurate while the forecasts based on the Rayleigh model remain far from the actual date of completion and thus are not usable for predicting this date.

6.4. Monitoring Project Status

As has been shown in the previous section, the Normal model was used as a predictive tool although the early forecasts will be an underestimation compared to the actual moment when the required level of quality is reached. Moreover, we have shown that the model is very useful as a method for monitoring the progress of a project and determining when the minimum required quality level is reached as the forecasts will become more accurate as the project progresses and more data becomes available.

The model provides information on the estimated defects still present in the software system. A quality requirement might be that at least 97% of the defects must have been resolved before the product is ready to be released. At Philips Healthcare MRI the quality requirements even demand solving all defects found during development and testing marked as either a level 1 (critical) or 2 (major) defect (out of the five possible levels). Based on such requirements, it is possible to predict when this goal has been reached. Using the estimate of the total amount of defects, as described in Section 4, we calculated the moment in time when 97% of the defects have been resolved.

In Figure 17 we have depicted an example of using the Normal model to monitor a project’s progress. On the vertical axis the number of defects submitted per month are shown, while the horizontal axis represents the number of months passed since the start of the test phase of the project. The blue curve shows the Normal curve fitted on the data using the information from the first 5 months of the project’s test phase. As shown, this curve largely underestimates the time to ship. That is, using this curve, the prediction is that 97% of the defects will have been found after 6 months, while in fact this does not happen until after 19 months. The orange curve, which uses the data of the first 10 months, still underestimates the time to ship and puts this at 13 months. However, when more of the data is used, the curves start to converge and estimate the time to ship at 19.4 (using data from the first 10 months) and 19.5 (using data from the first 15 months) after the start of the project. These curves are already so close together that it is hard to distinguish them from each other in this figure and totally obscure the red curve, which is based on all of the available data. As soon as the estimates start to converge—usually halfway through the test phase—it is possible to provide a firm estimate of the moment at which the product can be shipped.

6.5. A Time Series Approach

Various researchers have investigated alternative approaches for estimating project durations. Recently, a time series approach has been described for making such estimates [Raja et al., 2009]. Therefore, in this section we will apply the same approach to determine its usefulness for making long-term predictions and to compare the results to those obtained using the Normal model.

The failures over the life cycle of hardware components follow a “bathtub curve” with high defect rates at the beginning and the end of the life cycle. In the early life of a product,
the defect rate is high but rapidly decreasing as defective components are identified and teething problems are resolved. At the opposite end of the curve is the wear-out region. Components have been functional for a long period of time and are beginning to experience failures due to physical wearing of electronic or mechanic components.

The defect rate for software products is often assumed to follow a different curve and drop over time as more defects are detected and removed. Using stochastic models such as Poisson, Weibull, Rayleigh, or (generalized) Gamma distributions for modeling defect rates in software is a common practice. These models accurately model the global life cycle of the defect rates showing a decreasing trend as the software matures. However, some researchers have questioned this approach. Fenton and Neil [1999] claim that software defect data is autocorrelated, meaning that defects are related over time. Although they do not provide any (statistical) evidence of their claim, software failures are the result of human mistakes, which indeed do not appear to be random in nature [Dick et al., 2007]. One defect can induce the next fault. The above models ignore the causal effects of programmers and designers. This observation calls into question the validity of general linear models, which assume the absence of autocorrelation.

Thus, while theoretical distributions are well suited to model global defect rates as we have shown in the previous sections, at the micro level models which take autocorrelation into account might be better suited. Examples of such models are autoregressive integrated moving average (ARIMA) processes [Box et al., 1994] and possibly also generalized autoregressive conditional heteroskedasticity (GARCH) [Bollerslev, 1986] processes. ARIMA models intend to describe the current behavior of variables in terms of linear relationships with their past values.

ARIMA modeling takes into account historical data and decomposes it into (1) an autoregressive (AR) process, where there is memory of past events (e.g., the number of defects submitted this month is related to the number of defects submitted last month); (2) an integrated (I) process, which accounts for stabilizing or making the data stationary, making it easier to forecast; and (3) a moving average (MA) of the forecast errors, such that the longer the historical data, the more accurate the forecasts will be, as it learns over time. ARIMA models therefore have three parameters, one for the AR(p) process, one for the I(d) process, and one for the MA(q) process, all combined and interacting with each other and recomposed into the ARIMA(p,d,q) process. The general equation of an ARIMA(p,d,q) model is

\[ \nabla^d y(t) = \mu + \phi_1 \nabla y(t-1) + \ldots + \phi_p \nabla^d y(t-p) + \epsilon_t - \theta_1 \epsilon(t-1) - \ldots - \theta_q \epsilon(t-q) \]

where \( y(t) \) is the time series of the random variable \( y \). In this equation, the constant term (here denoted by \( \mu \)) is the average difference in \( y \). The parameters of the autoregressive part are denoted by \( \phi_1, \ldots, \phi_p \). The defect term is denoted by \( \epsilon_t \), which is assumed to be an independent, identically distributed variable sampled from a normal distribution with zero mean. Finally, \( \Theta_1, \ldots, \Theta_q \) are the parameters of the moving average part associated with \( \epsilon_t \).

In order to model a time series by means of an ARIMA model, the data needs to be stationary. Nonstationarity means that parameters such as mean and variance change over time. To transform a nonstationary dataset to stationarity, the data is differenced. The first difference of a time series is the series of changes from one period to the next. If \( y(t) \) denotes the value of the time series \( y \) at period \( t \), then the first difference of \( y \) at period \( t \) is equal to \( y(t) = y(t-1) - \nabla^1 y(t-1) \). For example, an ARIMA(1,1,1) model, in which all components are represented, is written as

\[ \nabla^1 y(t) = \mu + \phi \nabla y(t-1) + \epsilon_t - \theta \epsilon(t-1) \]

Raja et al. [2009] found that software defects over time were modeled by ARIMA models. They based their research on the monthly defect statistics of eight open source software projects. Their primary goal was to describe the pattern with which defects are reported in these open source projects using time series models. They also tested whether the parameters fitted over roughly the first 95% of the available data also resulted in a good estimate of the number of defects in the last 4 months of the project (roughly equal to the last 5% of the investigated projects). In the case of Raja et al. [2009], the ARIMA(0,1,1) described all eight sampled projects. They did not investigate whether the model could be used for long-term predictions as is the goal of this paper.

In order to determine whether these models are useful to describe our data and predict long-term deadlines similar to the Normal model, we carried out an analysis similar to the investigation described in Raja et al. [2009]. Contrary to the results reported there, however, we found that the defect trend aggregated by month could not be modeled by means of an ARIMA model. The KPSS test [Kwiatkowski et al., 1992] showed that the basic data was not stationary, as required by the ARIMA model. As described above, it is possible to obtain a stationary time series by means of differencing. Indeed, the first-order difference of the data was found to be stationary. Subsequently, we carried out the Ljung–Box test [Ljung and Box, 1976] to verify whether the data was autocorrelated. However, we found that this was not the case. Similarly, the second- and third-order difference were stationary according to the KPSS test, but the Ljung–Box test showed no autocorrelation. Intuitively, this result is to be expected. Software defects are the result of human mistakes. Thus, if a programmer makes a mistake, it will result in a defect being reported the first or second day thereafter. Aggregating this process to the level of months, obscures this relation in our case.

This intuition also suggests that a defect trend based on the daily statistics might be suitable for modeling by means of an ARIMA model. Again, we carried out the KPSS test to test for stationarity of the data. While the basic data was not stationary, the first-order difference was. Furthermore, the Ljung–Box test showed that the first-order difference was autocorrelated. Thus, it should be possible to model the data by means of an ARIMA model. Indeed, we found
ARIMA(1,1,1) models in all cases and in some cases a slightly better fit was found with ARIMA(1,1,5) models.

Figure 18 shows the fit of an ARIMA(1,1,5) model onto the day-to-day defect data of one of the projects. The vertical axis shows the number of defects submitted per day, while the horizontal axis displays the number of days since the start of the project. The model trend (gray) closely follows the trend in the actual data (black). However, the upward and downward outliers are large compared to the actual data. This sometimes even results in a prediction of a negative amount of defects to be submitted. As this can be prevented by putting constraints on the possible values of the time series, the plot shows that an ARIMA(1,1,5) model describes that defect trend accurately.

Figure 19 shows the defect data for the same project as Figure 18. Figure 19a shows the same time series except that the ARIMA(1,1,5) model is not shown in this plot. The vertical axis displays the number of defects per day, while the horizontal axis shows the number of days which have passed since the start of the project. As we already know, the data closely resembles a Normal distribution. The raw data is not stationary. However, following the method described earlier, we have transformed the data to stationarity. The result of this process is shown in Figure 19b.

Figure 19c shows the autocorrelation function (ACF) values of $\nabla Y$, while Figure 19d depicts the partial autocorrelation function (PACF) values of $\nabla Y$. The ACF represents the correlation, at specific lags, between the residuals of the data. The ACF plot for this series indicates a cutoff at lag 1. The PACF may be interpreted as either a die out or a cutoff. Using the information from these plots, we estimate the parameters for the ARIMA model. The lag at which the values stay below the dotted line indicates at which point the data is no longer significant. This value indicates the order of the ARIMA model. As the ACF plot is significant for lag 1, while the PACF plot is significant up to lag 5, we have evaluated both ARIMA(1,1,1) and ARIMA(1,1,5) processes. The ARIMA(1,1,5) model proved to be slightly better. Furthermore, we have performed a similar analysis for the other projects as well. Also in these cases the ARIMA(1,1,5) proved to give the most accurate description of the data.
The ACF and PACF plots are also useful in determining the amount of differencing required in order to acquire a stationary series. As discussed in the seminal book on autoregression by Box et al. [1994], a basic tool to employ in the identification process is the cross correlation between input and output. As soon as the ACF does not damp out quickly, a degree of differencing of the time series is applied for which the ACF and/or PACF do damp out quickly. In practice, the degree of differencing is 0, 1, or 2. In our case, it turned out to be in all cases a differencing of 1.

An important question regarding the noise, as displayed in Figure 19b, is whether it is “white noise,” that is, does not contain any information and thus is entirely random lacking any further systematic properties. This is one of the most important questions one can ask about the residual time series that results from having fit a model. If we can say “yes,” then we have a useful indicator that our analysis has gone as deeply as it can go and the model indeed accurately describes the actual data. Figure 20 shows the residuals, similar to Figure 19b. Furthermore, it displays the ACF of the residuals and the results of the Ljung–Box test [Ljung and Box, 1976]. Since the ACF does not show any significant lags other than zero, there is presumably no significant lag for autocorrelation left. The Ljung–Box test analyzes the residuals of the data to see if it is autocorrelated up to a number of lags. If the results from this test are different from zero, then the data is not autocorrelated and is presumed to be random. The test is derived from the idea that the residuals of a “correctly specified” model are independently distributed. This means that the residuals no longer contain autocorrelation and thus the ARIMA(1,1,5) model extracted all of the nonrandom behavior and only white noise is left. Indeed, the results from the Ljung–Box test in Figure 20 are different from zero. Thus, the ARIMA(1,1,5) model is the best possible autoregression model to describe our data.

Although we could reproduce the findings of Raja et al. [2009] on day-by-day data, our goal is to use a model for predicting the time when a sufficient number of tests have been performed rather than describing the data and perform short-term predictions. In other words, we strive for a model that predicts as soon as possible what the end date of the project will be. Time series analysis is highly sensitive to the history length. Prediction accuracy tends to decrease as the number of predicted steps-ahead increases [Chatfield, 2003; Kenmei et al., 2008]. In particular, they tend to damp out to the overall mean [Chatfield, 2003]. Thus, ARIMA models only serve short-term predictions while with more time units to predict the models tend to damp to a long-term average.

In our case this is also true. Figure 21 shows the prediction for one of the studied projects. The vertical axis shows the number of defects submitted per day. The horizontal axis displays the number of days which have passed. The first half of the figure shows the actual defect trend in black based on the available data. Based on the ARIMA(1,1,5) model we predict the second half of the project (blue line). The black line, which quickly changes into a straight line. The yellow and orange shaded areas show the 80% and 95% prediction intervals. The dotted gray line shows the defect trend of the actual data. From this figure it is clear that the ARIMA(1,1,5) model is not suitable for making long-term predictions. The forecast does not model the actual data well at all. Moreover, the rapidly widening confidence intervals show that the ARIMA(1,1,5) model does not make accurate predictions of the future defect trend.

In short, although we also found that defects over time can be described by ARIMA models, as was reported on in Raja et al. [2009], we could not use these models to predict the time trend. The reason is that ARIMA models turn into a straight line already after a few predictive steps. These few predictive steps would suffice in the case of monthly data—as these would still describe several months. However, our analysis showed that it is not possible to describe monthly data using ARIMA models. ARIMA models are able to predict the trend.
describe defect trends based on daily statistics, but these models lack the capabilities for making long-term predictions.

### 6.6. COCOMO and COQUALMO

Another well-known model which might be used to make predictions regarding quality and duration of a project is COCOMO. COCOMO (Constructive Cost Model) [Boehm et al., 1995] and its successor COCOMO II [Boehm et al., 2000] are models for estimating software costs. Cost, schedule, and quality are highly correlated factors in software development. Beyond a certain point, it is difficult to increase the quality without increasing either the cost or schedule or both for the software under development. Similarly, development schedules cannot be drastically compressed without hampering the quality of the software product and/or increasing the cost of development. Recognizing this fact, a quality model was developed as an extension to COCOMO II, called COQUALMO [Chulani and Boehm, 1999]. COQUALMO is an estimation model that can be used for predicting the number of residual defects in a software product. It can also be used for estimating time-to-market in terms of lost income due to overspending on quality or loss of market share [Huang and Boehm, 2006; Madachy and Boehm, 2008].

While the goal of COQUALMO is similar to that of the model we have described, the method is very different. We have extracted a model from the available data. COCOMO together with COQUALMO, however, is a fixed model which needs to be tuned to the specific environment. There is a wide variety of parameters in various categories such as precedent-ness, team cohesion, product complexity, personnel continuity, and platform volatility. These parameters need to be ranked allowing for values such as “very low,” “low,” “nominal,” “high,” and “very high.” While these rankings have been defined in terms of project size and source lines of code, they leave room for interpretation. Moreover, taking lines of code as a starting point for such an analysis will be problematic when parts of the software development are outsourced or when a system is forked and subsequently merged. These activities will have a significant impact on the (changes in the) number of lines of code as shown in Kulk and Verhoef [2008]. When using the data from Philips we end up with a trend as shown in Figure 22. This figure reveals that the daily differences between the amount of lines of code can change drastically in a very short time period. This is caused by forking or merging between different development streams, by checking third-party software in, or by incorporating open source modules. In this case, on 57 different days the daily difference is at least 10KLOC. With a measurement period of 210 days, there are significant differences every 3.7 days. Thus, it will be very hard to update the predictions made by the COCOMO model during the project, as the amount of changed lines of code varies wildly and bears little predictive information as the amount of change is not directly related to the amount of work done, but to other external factors such as the decision to fork or merge development streams.

To illustrate our point, we applied the COQUALMO model to one of the projects used for this paper. We found that the model was not able to provide predictions in line with the actual data available to us. Moreover, varying only one or two parameters resulted in a profoundly different estimation of effort and quality. Differences of 1000 person-months were no exception. An example of two different predictions is shown in Figure 23. These figures have been created using the online tool3 which is provided by Raymond Madachy, one of the original authors of COCOMO II. In this case we varied only the value for the reliability and compared the estimates for high and very high reliability. The results show large differences in the staffing profile with the estimated effort varying between 8000 person-months when we chose high reliability and over 9100 person-months in the case of very high reliability.

---

Furthermore, if we zoom in on one of the differences in Figure 22 exceeding 10KLOC and use these two data points as input to the model, we end up with vastly different predictions. Before the merge activity, COQUALMO predicts an effort of over 2200 person-months, while directly after this activity the prediction for the remainder of the project is a little over 1000 person-months. This lack of accuracy is in line with earlier findings: “The resulting model produces estimates within 30% of the actuals 64% of the time for effort” [Clark et al., 1998]. Similarly, others found that “COCOMO II produces estimates within 30% of the actuals 75% of the time for effort” [Chulani et al., 1999].

The above examples illustrate that trying to fit the available data to the fixed COCOMO models yields results that are beyond reality. This is clearly illustrated by Figure 24. This shows the staffing profile required to change 10 lines out of 3.9MLOC. According to COCOMO this would require 1838 person-months. The difference with our approach is that, instead of trying to fit the data on a fixed model, we have extracted a model from the available data. Trying to fit data on a fixed model that does not fit is bad practice. Unfortunately, this practice is so endemic that it has a name, Procrustean statistics. Wikipedia provides a clear explanation4:

In a Procrustean solution in statistics, instead of finding the best fit line to a scatter plot of data, one first chooses the line one wants, then selects only the data that fits it, disregarding data that does not, so to “prove” some idea.

Thus, in the presence of detailed and accurate data it is ill-advised to use fixed models. A fundamental difference with our approach is that we let the data speak to form a model. This statistical handiwork is known as exploratory data analysis—this is the preferred approach when actual data is present. Of course, if next to data certain beliefs are present, a Bayesian approach should be kept in mind, too. For instance, if you believe that the mean time between failure before testing exceeds 250 hours, you can calculate that under that condition the time to test will be much lower than without using that belief. In any case, postulating a model and trying to fit data on it that does not fit is bad practice and more sophisticated analysis methods such as the one we used (EDA) are necessary.

7. THREATS TO VALIDITY

As stated by Basili et al. [1999], drawing general conclusions from empirical studies in software engineering is difficult because any process depends on a potentially large number of relevant context variables. For this reason, we cannot assume a priori that the results of a study generalize beyond the specific environment and projects for which it was conducted. However, in time we hope they can form part of a broader picture. Researchers become more confident when similar findings emerge in different contexts [Basili et al., 1999]. Furthermore, the number and distribution of the defects from the projects in our study will differ from those of other companies. This is often misunderstood as a criticism of empirical studies. However, while the specific numbers will be different for different companies, the method is generally applicable. This study shows useful results to build up a body of empirical evidence, which hopefully encourages more researchers to run similar studies and deepen the understanding of the field.

A problem with using historical data is the tenability of the results. For instance, more and more projects at Philips Healthcare MRI are organized in an agile fashion. The projects are carried out in small increments, each resulting in a work-in-progress release of the software which is used for testing. This results in a pattern different from what we have seen in projects carried out in the past following a traditional waterfall model as shown in Figure 25. The waterfall model shows a distinctive, mountainlike pattern which, as we have seen, fits well with a Normal curve. The agile model, however, looks different. Clearly visible are three peaks in the 21st, 27th, and 32nd months of the project. Prior to these months, a work-in-progress release was delivered to the testing department, resulting in an increase in the number of defects submitted in the month following the release.
As this way of developing software is relatively new within Philips Healthcare MRI, there is not yet sufficient data to determine the effects of these changes on the usability of the Rayleigh and Normal model, but it stands to reason that this will require some changes compared to the current implementation. The general principle will still be applicable, but the specific formulas and their coefficients need to be found and fitted.

8. RELATED WORK

There are various related areas of research which concentrate on parts of the question of how to balance time-to-market and quality. We will discuss some of these efforts in the remainder of this section.

Effort estimation

In this article we make use of historical information to determine a relation between the size of a project, in terms of person-hours, and the number of defects that will be found. Various other researchers [Bernstein et al., 2007; Khoshgoftaar et al., 2006; Nagappan et al., 2006; Turhan et al., 2009] have used historical data for this purpose as well. They have used information from various repositories such as version management systems and bug tracking systems to predict the number of defects in the software system. Other work focuses on estimating effort and cost based on analogies between projects [Shepperd and Schofield, 1997] or makes use of models such as COCOMO. A more complete overview of effort estimation models can be found in Briand and Wieczorek [2000].

Software quality

In this article we proposed a modified version of the formula devised by Bishop and Bloomfield to estimate the quality of a software system in terms of the number of dangerous faults that remain in the software. Fenton et al. [2007] and Tockey [2008] aim at a similar goal and use Bayesian nets to predict the number of software defects remaining undetected after testing. Others use information such as test coverage [Bishop, 2002] or individual estimates [Biffl, 2000; Lubashhevsky, 2002] to estimate the number of residual faults.

Defect trends

In order to model the defect trend in a software project, we have used the Rayleigh model as described by Putnam and Myers. We have proposed the Normal model as an alternative. Various researchers [Burgess and Lefley, 2001; Card, 2002; Pillai and Nair, 1997] have also used Weibull models or variants thereof to model defect trends in software projects. Others have taken different approaches and have used change data [Mockus et al., 2003], expert estimations [Bai et al., 2003], and project plannings [McConnell, 1996; Staron and Meding, 2008] to predict the trend with which defects will be discovered. Furthermore, various researchers have proposed to model this trend as time series [Dick et al., 2007; Herraiz et al., 2007; Kenmei et al., 2008; Raja et al., 2009]. In this article, we have shown that there is no generally applicable model that works in all circumstances. Our results show that the Rayleigh model does not work well in the case of Philips Healthcare MRI. Time series can be used to describe the data, but not for making predictions. We have shown that in our case it is better to use the Normal model instead.

Release readiness

The goal of the method described in this article is to predict the moment when the software system is ready for release. McConnell [1997] provides an overview of different techniques for determining software readiness based on defect tracking, defect pooling, defect seeding, defect modeling, and combinations thereof. More recently, various authors [Malaiya et al., 2002; Quah, 2009; Ware et al., 2008] have used testing time, defect tracking, and various other metrics to determine whether the software is ready to be released or not. The approach described in this article goes one step further by determining release readiness in terms of the required level of quality. We calculate this level by combining information from faults and failures. By using a defect trend modeled on historical information we make a prediction of the due date of the software and monitor the progress of a project.

Failure rates

Analysis of failure rates of life-critical software-intensive systems is paramount for our understanding of these types of systems and the level of reliability they can realistically achieve. There are various studies based on publicly available data such as a study on the reliability of medical devices [Maisel et al., 2001; Shah and Maisel, 2006] which uses data from the FDA website, and a study on the reliability of the Maeslantkering [Verhoef, 2007b]. A limitation of studies using publicly available data is that this data only represents a part of the actual accidents that occur [Gilk, 2009].
Some examples where real-world in-depth data is used are NASA which provided data on software failure processes [Finelli, 1991] and an analysis of a fuel cell system [Feitelberg et al., 2005]. Studies using reliable, real-world data not only aid our understanding of these types of systems in general, but also help those companies to obtain invaluable insights into the reliability of their own systems.

9. CONCLUDING REMARKS

In this article, we have used historical data obtained from multiple repositories and combined them in order to make estimates on the number of defects that will be found during a project based on the size of that project. Furthermore, we have used information on field problem reports and the installed base to come to a bound for the allowable number of residual defects as a measure for the quality of the end product. Together, these two estimates help us to predict when this level of quality is reached using the Normal model. Putting this in other words, the model allows us to determine when the software can be shipped with the required level of quality sooner.

In our case study, the Normal model provides an accurate description of the data and results in forecasts for the time-to-market with high EQF values. The EQF indicates the quality of a forecasting method. Of course, other theoretical models can be more suitable in other contexts. This does not change our approach as a whole.

It turned out that the commonly used Rayleigh model does not provide a good description of our data and thus is not suited for making any predictions. Time series models, in particular ARIMA models, do accurately describe the defect trends. However, long-term predictions based on these models do not result in useful forecasts as they tend to revert to the overall mean. Furthermore, cost estimation models like COCOMO are useful for providing order-of-magnitude estimates, but lack the predictive capabilities of a model based on the actual data compiled using EDA. In short, our analysis based on information from repositories of historical data supplements the expert’s opinion and provides numbers to reason about.

In summary, the method to balance time-to-market and quality comprises four steps. First, collect the data from the various repositories. Second, using the effort-to-defect formula, estimate the total number of defects based on the size of the project in person-hours to provide a range for the defect trend model. This provides an upper and lower bound to the predictions based on the Normal model. Third, calculate the required level of quality using the modified version of the Bishop and Bloomfield formula. Fourth, while the project progresses, collect the data on the actual rate of defect submission and compute a fit for the Normal model over this data while recomputing the fit as more data becomes available.

Using these four steps, we obtain more accurate estimates on the time-to-market as more data becomes available. We applied this method at Philips Healthcare MRI and our analysis showed that their development practices result in a high level of quality without compromising the time-to-market. Furthermore, it turned out that the measured defect rate of the software of Philips’ MRI scanners is once per thousand device-years. This is the second highest safety integrity level, SIL3, as defined in the IEC 61508 standard (SIL4 is higher but only achievable by introducing some form of redundancy).

We base our approach simply on the data available to us as opposed to applying an existing model and trying to fit that onto the data at hand. Our analysis showed that existing models did not apply to this data. While these models have proved their usefulness in different circumstances, they did not provide useful results here. This holds for the Rayleigh model, for time series such as ARIMA and the cost estimation models like COCOMO. We showed this by applying these models with less than desirable results. Using straightforward EDA we were able to identify a simple, symmetrical model which resulted in accurate estimates.

The analysis confirms in this case that Philips Healthcare MRI is doing the right thing. In other cases it will show you whether you are working too long on defect removal, and in practice most cases show that people are not working enough on defect removal for the necessary quality level. It helps in the communication between management and IT: less testing means less quality. In that way it creates a negotiation means between the two fields. Momentarily this is not common resulting in low-quality software since IT cannot express in quantitative terms when more testing is necessary and what it delivers in terms of quality. By applying this method, it is possible to provide a factual basis for the expert opinion which is repeatable and measurable. In other words, with this method it is possible to provide a target delivery date with a measurable level of confidence.

The important underlying contribution here is the utility of using in-process and product metrics from large software systems, namely, MRI scanners, to make early estimates of software quality in terms of failures or failure-proneness. We linked this data to estimations on project size and models for predicting defect trends in projects in order to balance the time-to-market and quality of the end product. We do not claim that specific numbers are relevant outside of Philips Healthcare MRI. However, they constitute a step toward a comprehensive picture in this field of study. Moreover, the method presented here is generally applicable and can be used in other companies.

ACKNOWLEDGMENTS

We thank Peter Kampstra (VU) for his help with the R calculations and visualizations and Laurenz Eveleens (Price-waterhouseCoopers) for his help with the EQF calculations; Piërre van de Laar (ESI) and Miha Fuderer (Philips Healthcare MRI) for comments on earlier versions of this article; those persons from Philips Healthcare MRI who helped with obtaining all of the data, for their invaluable input throughout various discussions and for their comments on earlier versions of this article; and the anonymous reviewers for their valuable comments.

This work has been carried out as part of the DARWIN project at Philips Healthcare under the responsibilities of the Embedded Systems Institute. This project is partially supported by the Dutch Ministry of Economic Affairs under the
BSIK program. This research also received partial support from the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.004.405 EQUITY: Exploring Quantifiable Information Technology Yields and via contract 638.003.611 Symbiosis: Synergy of managing business-IT-alignment, IT-sourcing and offshoring success in society.

REFERENCES


S.R. Rakitin, Balancing time to market and quality, Softw Qual Prof 1(3) (1999), 54–57.


M. Staron and W. Meding, Predicting weekly defect inflow in large software projects based on project planning and test status, Inf Softw Technol 50(7–8) (2008), 782–796.


Pieter van der Spek studied Technical Informatics at Delft University of Technology and was awarded a master’s degree in Software Engineering in 2004. Following his studies he worked for two years in industry as a software engineer before starting his PhD at the VU University Amsterdam. He participated in a collaborative research project together with researchers from several other universities, the Embedded Systems Institute, and Philips Healthcare. He received his PhD (Software Evolution in Embedded Systems) in 2010 and returned to industry where he is currently working as a software architect with a special focus on technical and embedded systems.

Chris Verhoef is a professor at the VU University Amsterdam. He served as Vice Chair of conferences of the IEEE Technical Council on Software Engineering, and held Steering Committee, General Chair, and Program Chair positions for many important juried software-related research conferences. He acted as an industrial consultant in several IT-intensive areas, notably hardware manufacturers, telecommunications companies, financial enterprises, government, and large service providers.