Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 264 (2010) 3–17
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Combining Architecture-based Software Reliability Predictions with Financial Impact Calculations

Franz Brosch [1]

Forschungszentrum Informatik (FZI) Karlsruhe
76137 Karlsruhe, Germany

Ralf Gitzel[2], Heiko Koziolek[3]

ABB Corporate Research
68526 Ladenburg, Germany

Simone Krug[4]

Chair in Information Systems III, University of Mannheim
68131 Mannheim, Germany

Abstract

Software failures can lead to substantial costs for the user. Existing models for software reliability prediction do not provide much insight into this financial impact. Our approach presents a first step towards the integration of reliability prediction from the IT perspective and the business perspective. We show that failure impact should be taken into account not only at their date of occurrence but already in the design stage of the development. First we model cost relevant business processes as well as the associated IT layer and then connect them to failure probabilities. Based on this we conduct a reliability and cost estimation. The method is illustrated by a case study.

Keywords: Software failure, Reliability, Cost.

1 Introduction

Our approach targets financial risks of failing software systems. There are numerous examples where software defects had major financial impact. For example, a

[1] Email: brosch@fzi.de
[2] Email: ralf.gitzel@de.abb.com
[3] Email: heiko.koziolek@de.abb.com
[4] Email: skrug@wifo.uni-mannheim.de

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2010.07.002

debt product was incorrectly awarded a prime rating by a major agency due to a coding error in their rating model [4]. Lufthansa, one of the largest airway carriers worldwide, suffered from several outages of their check-in system, leading to flight cancellations and delays for passengers [20] [9]. British Airways reported an estimated loss of GBP 16'000'000 in the first five days of operation after the opening of Heathrow's Terminal 5. It was caused by the incorrect recognition of messages received and the incapability of the new baggage handling system to cope with the amount of messages generated [6].

Forecasting such extreme failure induced impact is hard. Software reliability prediction requires knowledge about how a system is used, which includes the business process supported by the software. The financial impact of a systems's failure has to be considered according to the frequency of its use. Functionality with small cost impact but high occurrence rates can lead to higher overall impact than the one rarely used.

Our approach presents a method to bridge the gap between the reliability prediction calculated from the model of the IT layer and the cost impact modeled as a business case. The objective is to support business analysts in comparing different system alternatives during design. Our method is also of use for the software developer, as it allows focusing on the customers' needs and helps to assess possible compensation costs.

Applicability of our method is demonstrated in a case study modeling an industrial maintenance management system from ABB. We support our case study with the Palladio Component Model (PCM) [2] [14]. A distinctive feature of the PCM, compared to other reliability prediction approaches, is its combined consideration of software failures and hardware unavailability, as well as the support for service usage propagation through parameterized behavioral specifications [14]. Its relevant inputs are the usage profiles and failure probabilities of individual software components. The PCM is well suited for our approach, since it explicitly takes these parameters into account.

An inherent aspect of our approach is the shift from the software provider's perspective to the view from the customer's side to decide how severe a failure is. This paper is organized as follows. Section 2 introduces the basic concepts of our approach, describing the steps taken to establish the link between the business process and the IT layer. Section 3 describes the case study to illustrate the application of the approach, followed by a discussion of limitations in Section 4. Section 5 discusses work related to our approach. Finally, section 6 concludes the paper.

2 Method

We propose a two-phase method. First, a model of the business processes (called business layer) and its links to the hardware/software system (called IT layer) is created. In a second phase, additional reliability and cost information is added to the layers to determine various forms of cost information.

There are business processes that occur during regular operation and some that

will be triggered by failures. Based on the modeled business processes, we determine which parts of the computer systems have to be represented in the IT layer. As shown in figure 1, not all business process steps are connected to the IT layer. Later, these steps of the different business processes have to be associated with cost data.

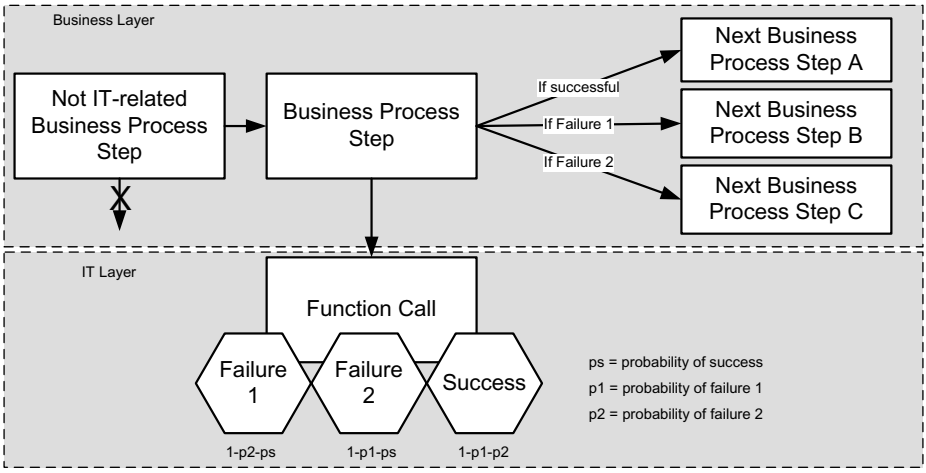

Fig. 1. Connection between Business Layer and IT Layer

Our approach models the IT layer using the PCM notation as a source of failure information for the business layer. Therefore, failure probabilities for software components and availability properties for hardware resources are added to the IT layer. The addition of this data finally allows a calculation of probabilities for the different process variants based on the system reliability.

In the next step, the system reliability (i.e., the probability of failure on demand for an IT service) is calculated based on the transformation of the PCM model into a discrete-time Markov chain (DTMC). The results include a distinction between failure modes.

Selected failures along with their calculated probability of occurrence can be combined with the cost information to predict the overall expected cost. The expected cost can be used for comparing design alternatives. In the following subsections, we will further elaborate the different steps of our method.

2.1 Modeling the business layer

The cost of a failure for the end user depends on the business case the software supports. The business case can be described as a combination of the business processes triggering the hardware/software system as well as cost information for each of the steps. The business steps using the IT layer form the connection between the two layers as shown in figure 1. Each business process step can make at most one call to the IT layer. The call can be either successful or lead to a failure with a certain probability. On success, the planned business process continues. In the case of failure, the business process will deviate from the original plan, incurring additional cost for the necessary extra steps.

The Object Management Group has developed a standard for business process modeling, the Business Process Modeling Notation (BPMN). We aim to use this language and augment it with cost information that we attach to each step of the process. Because of space restrictions, we will not explain the details of BPMN [19]. We use the term process step synonymously for the BPMN object *activity*.

Due to the possible failures, the business layer contains not only a model for the planned business case but also for the deviant processes incurred by a failure. For example, if a server crashes, a member of the IT staff might have to restart the machine. Since there is an almost unlimited number of failure scenarios, a preselection has to be made in order to focus on the most relevant options. As our example shows, some failures (i.e. the cost drivers) have a dominant cost impact while others are irrelevant.

The cost deviation is calculated as the difference between the total cost of the standard and the deviant process. For each process step the required time for rework, consumption of material, cost of opportunity, and other costs (such as fees for using software as a service) can be considered. We focus on the factors most comprehensible and highly influential in industrial settings: downtime related costs and additional work labor costs. The first includes lost profits and penalty costs for late delivery to the customer. Additional labor costs cover expenses related to tasks which have to be executed in order to correct the system's faulty state and to perform rework of certain process steps which were disrupted during first execution. Labor costs can be calculated by the estimated work effort in time units multiplied with the estimated labor cost per time unit. The analysis of deviant costs should be conducted in close cooperation with the accounting department. The costs of consequences in safety-critical systems, such as the harming of human beings or the environment, is beyond our scope.

2.2 Modeling the IT layer

In our approach, we use the Palladio Component Model (PCM) for modeling the IT layer. The PCM is specifically designed to model component-based software architectures and evaluate their non-functional properties, in particular performance and reliability. We describe the PCM here very briefly; for a more comprehensive description see [2].

The PCM captures the important aspects of IT service reliability within four architectural views: The *structural view* describes service components, service interfaces, and their composition into an overall IT system. The *behavioral view* specifies probabilistic data and control flow through the system in terms of sequences of actions, including branches and loops. The *deployment view* specifies the hardware and communication links that form the execution environment for the IT system, as well as the allocation of service components to the hardware. Finally, the *usage view* describes how the system's services are used in terms of number of users, service invocation sequences, and service input parameters.

Several sources of failure are considered when evaluating IT service reliability with the PCM. *Software failures* are caused by faults within the software com-

ponents, which are eventually activated depending on the system state and the input parameters of the actual service invocations. The modeler specifies the likelihood of such failures by annotating certain actions within the control flow with a failure probability. *Hardware unavailability* may lead to service failure, if a hardware device is unavailable just when service execution needs it. The modeler specifies the Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR) of a physical resource, which is used to determine its steady-state availability $A = MTTF/(MTTF + MTTR)$, enabling the analysis of unavailability effects. Furthermore, the PCM considers the possibility of *communication link failures*, namely failures in transmission of a service invocation or return message sent over a network communication link. Comprehensive tool support exists for specifying all model parts and evaluating system reliability (see Section 2.4).

A planned key feature that will be added to the PCM in the future is a distinction between failure modes. So far, the PCM in essence only calculates the probability of any failure affecting the system as a whole. Effectively, for all the relevant points of failure in the *business layer*, a failure probability has to be calculated. In the context of this motivational paper, we do the calculations by hand, following the basic principles of the PCM as described above.

2.3 Estimating failure probabilities

With the business processes defined and connected to the components of the IT layer, it now becomes necessary to annotate the software components in the IT layer with the failure probabilities to run the calculation. Estimating failure probabilities is an important step of our method, as they serve as an input for the subsequent reliability calculation. We rely on estimations of domain experts. Limitations of expert estimations will be discussed in Section 4.

2.4 Calculating the reliability

Our approach uses the PCM to calculate the reliability of an IT service invoked from the business layer. Thereby, reliability is expressed through the probability of success of the service invocation, which means: the probability that no software and communication link failures occur during service execution, and none of the used hardware devices is unavailable (see Section 2.2).

For the reliability evaluation, a fully specified PCM instance is transformed into an absorbing DTMC, that represents all possible execution paths through the architecture, together with their probabilities. This transformation requires the propagation of service input parameters throughout the architecture in order to determine the execution path probabilities [14]. Three special states in the Markov chain mark the beginning, failure and success of the execution. Applying Markov theory [23] on the DTMC allows for calculation of the probability that the success state is reached from the execution start (and not the failure state).

2.5 Calculating the costs

After the probabilities for the failure scenarios have been determined with Markov models, it is trivial to calculate the expected cost of the failure scenarios. As described in Section2.1, the cost for each failure scenario is the difference to the cost of the baseline scenario. The overall expected cost is therefore the sum of all deviating costs in scenario i:

$$C = \sum_{i=1}^{n} p_i * (c_i + d_i),$$

where p_i denotes the failure probability as calculated in Section 2.4, c_i additional labor costs and d_i downtime induced costs as calculated in 2.1.

3 Case Study

This section describes a case study to illustrate our approach to integrate reliability predictions with financial impact calculations. Initially the system under study is introduced (see Section 3.1). Afterwards, the method outlined in Chapter 2 is applied step by step: first the business layer (Section 3.3) and IT layer (Section 3.2) are modeled and then annotated with costs (Section 3.4) in order to calculate the reliability (Section 3.5). Based on this the total estimated deviating cost are calculated for the example (Section 3.6).

3.1 System under study

Our example case examines the failure cost of a software system used in a manufacturing plant that produces paper. The plant has a number of machines (pumps, valves, motors) that require maintenance and repair by a crew of workers (*Technical staff*). Often a problem can be recognized before it becomes critical and repairs can be conducted before production downtime occurs. Key challenges are the assignment of work orders and the traceability of problems. In order to support repair tasks, a Computerized Maintenance Management System (*CMMS*) is used. The CMMS is essentially a database application tailored to the problem domain of production plant maintenance.

The Technical staff takes its repair orders from the CMMS. This application contains a data structure with all assets/machines in the plant. To these, *Defect reports* can be attached. The *Supervisor* is notified automatically by the CMMS about problems that have been detected in the plant and defines repair tasks that are put into the *Staff schedule*. The Technical staff receives a message (in whatever form) when a new task has been put into their schedule. They will execute the repairs described at the scheduled time. For this purpose, they use information from the CMMS (from the Defect report) and then fix the problem according to what is required. After the repair, they will write a *Repair report* to be attached to the defect report in the CMMS. The Repair report contains information about the actual vs. the planned times and whether the initial problem assessment was correct. This information can be used for statistical analysis or mundane book keeping.

A special case occurs when there are defects that require immediate attention because they cause the production process to stop. The Technical staff detecting the problem (or being informed by the operators) will make an immediate repair and then create a report in the CMMS describing the incident. This report is the same as the normal Defect report complete with attached Repair report. The difference is that all information is added at once.

3.2 Modeling the CMMS business layer

For the purpose of the example calculation, we focus on a single procedure that is used to submit a defect report and notify the supervisor of the new report. The procedure has two failure modes that can prevent a successful execution. The first problem is a *non-trivial downtime of the server*, i.e. the CMMS is down when the Technical staff tries to submit a Defect report. By non-trivial downtime we mean a problem that does not rectify itself after a moment but requires significant action by *IT staff*. The second scenario is the *unnoticed loss of a message to the Supervisor*. In it, the message sent to the Supervisor after a defect report has been created does not reach its recipient. Additionally, there is nothing that indicates this loss to any of the actors in the process. All other failure modes are considered to be *trivial failures* which are small nuisance problems that will slightly delay execution but do not have a relevant consequence on the execution of the business process.

In the case of no failure or a trivial failure, the process corresponds to the description given above. It is shown in figure 2. The different actors (human and computer system) are represented as so-called "swimlanes". The process starts at the first round shape, passes through different actors and ends at the bold round shape. There is one point of decision where the further actions depend on whether a problem in the production plant requires immediate action or whether it can safely be scheduled for a future time.

Figure 3 describes the first failure scenario. In it, the server crashes just before or during the creation of a defect report. As a result, a member of the IT staff has to restart the server for the work to resume. This additional step adds extra cost to the scenario.

Figure 4 shows the process that occurs when the message to the supervisor is lost. As the Supervisor does not realize that there is a defect, he cannot assign an appropriate job to members of the Technical staff. Consequently, the failure will remain unrepaired. For simplicity's sake, we assume that the failure will only be rediscovered when it has become critical. The result are the process steps and costs associated with a critical failure.

In order to compare the cost effects of failures, monetary values have to be added to the process steps in the diagrams. For the baseline case of no failure, these are:

- Create Defect report: 10 minutes of Technical staff work time
- Supervisor Evaluates Defect report: 5 minutes of Supervisor work time
- Define Repair Tasks: 5 minutes of Supervisor work time

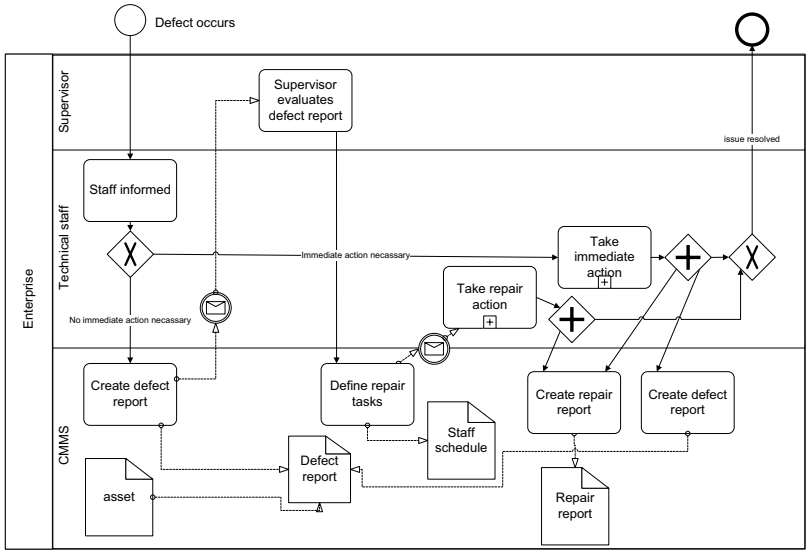

Fig. 2. Business Process for Successful Execution

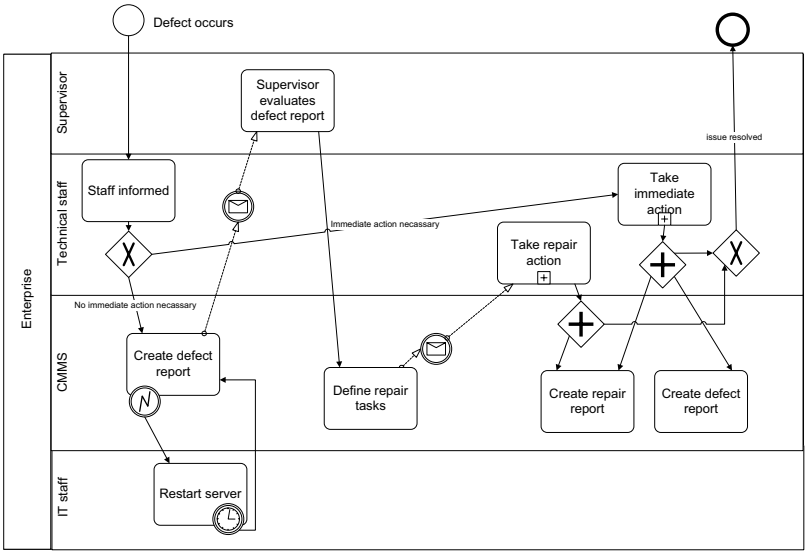

Fig. 3. Business Process with a Server Crash

- Take Repair Action: on average 1 hour of Technical staff work time
- Take Immediate Action: on average 1 hour of Technical staff work time
- Create Repair report: 10 minutes of Technical staff work time

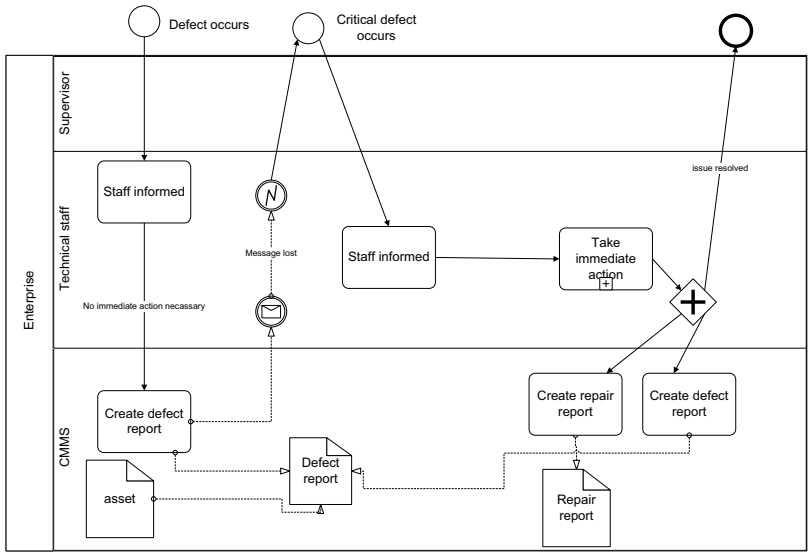

Fig. 4. Business Process with a Lost Message

- Downtime: if immediate action is necessary (see first decision point in the process), the time between defect occurs and the end of the process step Take Immediate Action is downtime. With a downtime rate of 10'000 Euro per hour and an average time span of 3 hours, this means 30'000 Euro of downtime cost.

The first failure scenario adds the following process steps:

- Restart server: 15 minutes of IT staff work time
- Rewrite defect report: 10 minutes of Technical staff work time

The difference to the original scenario are 15 minutes of IT staff work time and 10 minutes of Technical staff work time, which in this example both correspond to 100 respectively 60 Euro per hour. Thus deviant costs are 25 + 10 = 35 Euro.

The second scenario only differs from the baseline when a message is lost and the failure resurfaces as a critical failure later. This means downtime cost and cost for the process steps *Take Immediate Action* and *Create Defect report*. No longer relevant is the cost for *Take Repair Action*. This means downtime cost of 30'000 Euro and the comparatively marginal additional labor cost of 10 Euro for the 10 minutes spent on the second defect report like in the first scenario.

3.3 Modeling the CMMS IT layer

We have created a PCM instance for the CMMS in order to evaluate the reliability of the IT service operations invoked by the business layer, namely the process step calling *createDefectReport*. Figure 5 displays the service components and interfaces that are involved in the process. As the figure shows, all three invoked service

operations are part of the *ReportingEngine* service, which in turn uses a *Database* service for storage, update and retrieval of any information, and a *Notificator* service for sending notifications to staff and supervisors. In the context of the paper, we limit our calculations to the failure modes of createDefectReport.

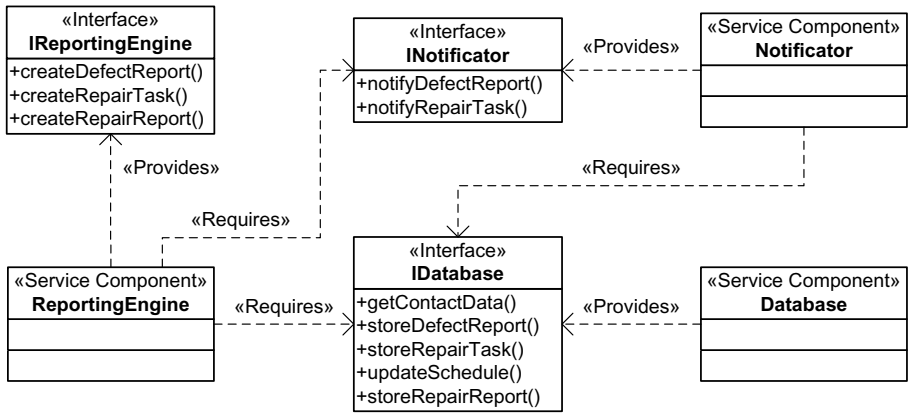

Fig. 5. CMMS Service Components and Interfaces

We have modeled the control and data flow through the components in PCM for createDefectReport. When a staff member creates a defect report, the *ReportingEngine* stores it in the Database, and issues a request to the Notificator to send an e-mail to the corresponding supervisor.

Regarding the execution environment, we have modeled a *Main server* and a *Database server*, and assumed that the ReportingEngine and Notificator are allocated to the main server, while the database is separated and runs on the Database server.

Finally, we have completed the PCM instance by creation of an own usage scenario for the scenario createDefectReport. Together with the failure probability annotations (see Section 3.4), all required information for the evaluation of IT service reliability is present.

3.4 Estimating failure probabilities and deviant costs

In our example, we estimate the failure probabilities of the CMMS system to be:

- Main server unavailable (0,01%)
- Database server unavailable (0,001%)
- Communication link between servers failed (0,01%)
- Storing of report in database failed due to software failure (0,01%)
- Retrieval of Supervisor contact data from database failed due to software failure (0,01%)
- Notification of Supervisor failed due to software failure (0,02%)

3.5 Calculating the reliability

As an example, we have evaluated the reliability (that is, probability of success) of the service operation createDefectReport, which is invoked by the business layer in the step *Create Defect report* (see Section 3.3). The DTMC for the createDefectReport operation is shown in figure 6. The DTMC represents the sequence of actions involved in the service execution, and determines the risk of failure for each action. The service operation's reliability is the product of the individual success probabilities of all actions, namely $R = (1 - 10^{-4})^7 * (1 - 10^{-5}) * (1 - 2 * 10^{-4}) \approx 0.99909$.

The probability of the first failure mode (server failure) can also be calculated from the DTMC. It is approx. 0.00011, which is the possibility that the call fails due to a server problem (but not due to anything else). The probability of a lost message is the probability of a failure in the last step (but in none of the steps before), which is approx. 0.0002.

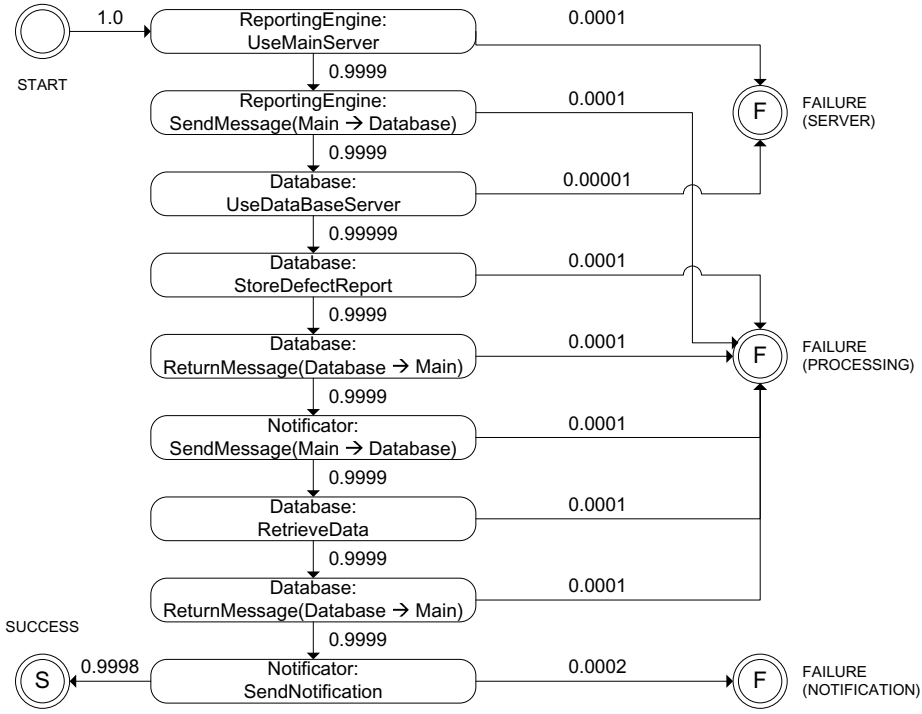

Fig. 6. DTMC for the CreateDefectReport Operation

3.6 Results

With the numbers given in sections 3.2 and 3.5 we can now calculate the cumulated costs for the two scenarios:

$$C = (0.00011 * (35 + 0)) + (0.0002 * (10 + 30'000)) \approx 6$$

Even though the numbers appear very suggestive, they should be handled with care. These numbers describe only the average case for a single execution for the

process. Given a single installation and a short time span, the actual average cost per call can differ vastly. However, the numbers can be used to compare this software with design alternatives or to evaluate potential improvements. For example, the decision maker can support the choice for one particular system with the advantages due to better reliability properties expressed in monetary values. He can also decide to substitute an old software component responsible for the execution of a process step with high failure costs. In our example server, crashes are irrelevant when compared to the problem of lost messages. Thus the latter has to be scrutinized in more detail.

4 Assumptions and Limitations

Observation of systems at run-time is most suited to represent the perspective of the business user who is confronted with the software product after the development phase. A software system used in a business environment cannot be regarded to remain unchanged over a longer period of time. Adding or exchanging one of the components represents a not to be underestimated factor in the calculation of downtime induced by failures. Unlike other authors (e.g., [17]), we do not incorporate repair actions and their implications on the failure rate. As bug fixing and the introduction of new faults by system extensions are usually not within the area of control of the end user, we do not consider this aspect. Also the application of fault tolerance mechanisms, e.g. integrated into components or one component taking over the functionality of a faulty one, are out of our scope.

Estimation of the failure probabilities is one of the most critical assumptions. We take a parameterized approach, which relies on estimations of domain experts. The main drawback here is that the assessment of such a domain expert is strongly subjective and an expert may not be available [5]. Even though these findings refer to a different problem domain, expert judgment is valuable for estimation, as recent work in the field of software development work effort forecasting indicates [11] [12]. Therefore, even if we rely on rough estimates, our approach still is able to contribute to the estimation of cost impact and especially for comparing two design alternatives.

For the sake of simplicity our approach does not provide a mechanism to deal with the occurrence of multiple failures at the same time. Since failure probabilities are very low, we consider concurrent appearance to be too unlikely to have a noticeable impact on the end result. We only take a small subset of failures into account, where we focus on a single consequence of a failure. Also we only regard a limited number of scenarios, as there are too many possible failures and reactions to them. However, important cases are covered by our approach to get a good estimate and to allow comparisons between two different software solutions.

We also do not provide any insight on the time needed for a component in order to propagate its defective state to another, i.e. the delay between a failure in one component leading to a failure in one or more components that depend on the input of the first one. Markov chains are a well established method to model

the transitions between components [5] [7]. For a detailed discussion of limitations related to reliability in component-based architectures please refer to [14].

So far we have no conclusive method to rate the quality of a scenario, i.e. how well it represents different kinds of impact in practice. A more sophisticated analysis of software failure impact as well as a classification is needed and part of further research.

5 Related Work

On the IT layer, reliability growth models observe decreasing failure rates during testing and continuous fault-fixing in order to evaluate and predict testing effort or the failure rate to expect at the end of testing [1] [10] [16]. In contrast, our approach looks at a stable system at run-time, and evaluates the system-level failure rates based on knowledge about failure rates in individual components.

Component-based approaches mainly rely on probabilistic models to determine reliability. One recent example presented a probabilistic model checking tool for the analysis of performance and reliability properties of system, incorporating different failure probability measures [15]. However no account is given for the consideration of different user profiles. [8] explicitly stress the importance of usage profiles as a precondition to software reliability calculation but employ random testing for the reliability calculation without specifying the dependencies of transition probabilities. Our approach explicitly models the call propagation trough the components in the IT layer, supported by the PCM.

On the business layer, widely known cost estimation approaches such as COCOMO do not even consider software failures as one of the cost drivers for a running system, since the focus lies only on the development stage. Even the extension of the model [3], spanning the whole software life-cycle, falls short in this aspect. Other approaches explicitly take these costs into account but cannot provide any insights on the financial implications for the end user. [21] offers an overview on Cost of Quality models which cover many aspects of costs but neither do so for the customer nor establish any link with the IT layer. The second point is achieved by [24], who combines efficiency metrics for defect-detection techniques with a model for software quality costs. His model aims to support management decisions by analyzing the relation of costs and benefits of a software project's investment prior to its start.

Karg and Beckhaus (2007) propose a framework that covers the investments for the removal of defects during testing. They stress the point that there is a general lack of empirical validation for all cost estimation models in this sector but not provide one themselves. The approach is based on cost calculation function. They do not offer a conclusive way to arrive at this function though [13]. [18] addressed cost of failures during the actual use of the system. He applied a random testing method to a black box view of the system. The implications drawn were used to optimize testing effort. Recently [22] proposed a framework for the estimation of software maintenance effort, again seen from the software provider's point of view.

In summary, none of the approaches has arrived at bridging the gap between the reliability of a system and software failure induced costs. They are targeted at the costs on the side of the software developer but do not satisfy the needs of the business user. Weyuker addresses this problem by estimating the cost of a software failure and the probability of its occurrence [25]. She emphasizes the importance of testing by measuring risk as a function of tests that have been run and also points out that different usage patterns may cause a significant difference in the behavior of the software. However, she does not elaborate on how these patters might look like. Our approach incorporates the business layer as one layer of IT service reliability. We show how reliability prediction on the IT layer can be associated with costs for the end user, introducing a novel view on cost estimation based on established methods.

6 Conclusions and Future Work

We presented a novel approach to the integration of reliability on the IT layer of a software system with costing aspects on the business side implied by the first. By demonstrating how business process steps can be associated with the estimated downtime in case of a defect, we offer a method to calculate software failure induced costs. Reliability analysis is supported by an advanced tool, widening the spectrum of parameters to employ. We can therefore consider concrete component structures on the technical level and diverse process scenarios on the business level.

The method allows users to anticipate the software failure related financial impact of a given software system thus enabling a comparison between design alternatives. It also helps software developers at design time to identify business process steps which are highly influenced by the reliability values of the underlying IT layer. Business experts can profit from the breakdown of cost structures to channel optimization efforts.

Future work includes the inspection of automated generation of failure scenarios as an approach to accelerate analysis for problems with a known resolution both on the business and the technical level. At the moment our approach is still at an early stage and we have yet to add further formalization to the cost assignment. We plan to integrate more sophisticated measures, such as for example Value at Risk and other financial key performance indicators into our model. Furthermore, a way to handle decisions in the business process is needed.

References

[1] Almering, V., M. van Genuchten, G. Cloudt and P. Sonnemans, *Using software reliability growth models in practice*, IEEE Software **24** (2007), pp. 82–88.

[2] Becker, S., H. Koziolek and R. Reussner, *The Palladio component model for model-driven performance prediction*, Journal of Systems and Software **82** (2009), pp. 3–22.
URL http://dx.doi.org/10.1016/j.jss.2008.03.066

[3] Boehm, B., A. W. Brown, R. Madachy and Y. Yang, *A software product line life cycle cost estimation model*, in: *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering* (2004), pp. 156–164.

[4] Charette, R., *Moody's software rating bug gives credit where credit isn't due* (2008).
URL http://spectrum.ieee.org/blog/computing/it/riskfactor/moodys_rating_bug_gives_credit

[5] Cheung, L., R. Roshandel, N. Medvidovic and L. Golubchik, *Early prediction of software component reliability*, in: *ICSE '08: Proceedings of the 30th international conference on Software engineering* (2008), pp. 111–120.

[6] Computerweekly, *Update: lack of software testing to blame for terminal 5 fiasco* (2008).
URL http://www.computerweekly.com/Articles/2008/05/09/230629/update-lack-of-software-testing-to-blame-for-terminal-5-fiasco-ba-executive-tells.htm

[7] Gokhale, S. S., *Architecture-based software reliability analysis: Overview and limitations*, Dependable and Secure Computing, IEEE Transactions on **4** (2007), pp. 32–40.

[8] Hamlet, D., D. Mason and D. Woit, *Theory of software reliability based on components*, International Conference on Software Engineering (2001), pp. 361–370.

[9] Heise, *Computerprobleme legen check-in-system der lufthansa lahm* (2009).
URL http://www.heise.de/newsticker/meldung/Computerprobleme-legen-Check-in-System-der-Lufthansa-lahm-798193.html

[10] Inoue, S. and S. Yamada, *Generalized discrete software reliability modeling with effect of program size*, Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on **37** (2007), pp. 170–179.

[11] Jørgensen, M., *Estimation of software development work effort: Evidence on expert judgment and formal models*, , **23**, 2007, pp. 449–462.

[12] Jørgensen, M., B. Boehm and S. Rifkin, *Software development effort estimation: Formal models or expert judgment?*, IEEE Software **26** (2009), pp. 14–19.

[13] Karg, L. M., and A. Beckhaus, *Modelling software quality costs by adapting established methodologies of mature industries*, in: *Proc. IEEE International Conference on Industrial Engineering and Engineering Management (IEEM'07)*, 2007.

[14] Koziolek, H. and F. Brosch, *Parameter dependencies for component reliability specifications*, in: *Proc. 6th Int. Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA'09)*, ENTCS **253** (2009), pp. 23–38.

[15] Kwiatkowska, M., G. Norman and D. Parker, *Prism: probabilistic model checking for performance and reliability analysis*, SIGMETRICS Perform. Eval. Rev. **36** (2009), pp. 40–45.

[16] Lyu, M., *Software reliability engineering: A roadmap*, in: *Future of Software Engineering, 2007. FOSE '07*, 2007, pp. 153–170.

[17] Musa, J. D. and K. Okumoto, *A logarithmic poisson execution time model for software reliability measurement*, in: *Proceedings of the 7th international conference on Software engineering*, 1984.

[18] Ntafos, S. and V. Poceciun-Benson, *Improved testing using failure cost and intensity profiles*, in: *ASSET '00: Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'00)* (2000), p. 143.

[19] Object Management Group, "Business Process Model and Notation (BPMN)," Object Management Group, 2009.

[20] Reuters, *Lufthansa cancels flights after computer failure* (2004).
URL http://www.usatoday.com/travel/news/2004-09-24-lufthansa-cancellations_x.htm

[21] Schiffauerova, A. and V. Thomson, *A review of research on cost of quality models and best practices*, International Journal of Quality and Reliability Management **23** (2006), pp. 647–669.

[22] Shukla, R. and A. Misra, *Ai based framework for dynamic modeling of software maintenance effort estimation*, Computer and Automation Engineering, International Conference on **0** (2009), pp. 313–317.

[23] Trivedi, K. S., "Probability and statistics with reliability, queuing and computer science applications," John Wiley and Sons Ltd., Chichester, UK, 2002.

[24] Wagner, S., S. Xie, M. Rubel-Otterbach and B. Sell, *Profitability estimation of software projects: A combined framework*, in: *The First International Workshop on Software Productivity Analysis and Cost Estimation (SPACE'07)*, 2007.

[25] Weyuker, E. J., *Difficulties measuring software risk in an industrial environment*, in: *DSN '01: Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)* (2001), pp. 15–24.