A Preliminary Taxonomy for Software Failure Impact

Simone Krug  
*Information Systems, University of Mannheim, Mannheim, Ba-Wue, Germany*, skrug@wifo3.uni-mannheim.de

Robert Nickerson  
*College of Business, San Francisco State University, State University, CA, United States*, rnick@sfsu.edu

Hugo Campidelli  
*Information Systems, University of Mannheim, Mannheim, Ba-Wue, Germany*, hcampide@mail.uni-mannheim.de

Follow this and additional works at: [http://aisel.aisnet.org/amcis2012](http://aisel.aisnet.org/amcis2012)

**Recommended Citation**  

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2012 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
A Preliminary Taxonomy for Software Failure Impact

Simone Krug  
University of Mannheim  
Simone.Krug@uni-mannheim.de

Robert C. Nickerson  
San Francisco State University  
RNick@sfsu.edu

Hugo Campidelli  
University of Mannheim  
H.Campidelli@googlemail.com

ABSTRACT

Abstract Software quality is of strategic importance to decision makers in enterprises. When software fails, the resulting impact on a business can be severe. So far, software failure has been mainly studied from the perspective of the developer by examining error prevention and correction through software testing and debugging. Even with a robust development process, software is almost never free of errors and thus failures will occur after the initial development process is completed and the software is installed and in use in an enterprise. Understanding the impact of such production software failures on enterprises, however, is complex and little research has been done on it. This paper starts the process of understanding the enterprise impact of software failure by developing a categorization, or taxonomy, of software failure impacts on the enterprise. We show how the resulting taxonomy can be applied to support practitioners and researchers in their efforts to gain a deeper understanding of the enterprise impact of software failures.

Keywords

Software Failure Impact, Software Failure, Taxonomy.

INTRODUCTION

In today's enterprises, software supports a wide range of processes, including those critical to the business. Alignment of IT and business strategies as a key success factor has long been recognized by the literature (Henderson and Venkatraman, 1990). The decision about the acquisition of a major software system has clearly to be made at a strategic level, especially regarding the importance of software quality for the productivity once it is employed (Fitzpatrick, 1996). When software fails, core processes, such as production and service operations, can come to a complete standstill. As software systems grow larger, system failures gain in their impact. A good example of a high impact software failure is the havoc caused at Heathrow airport in 2008 (Carrentals.co.uk, 2008). A malfunction in the baggage handling software led to massive interruptions, hindering passengers at the newly build Terminal 5 checking in with their luggage, and forcing employees of British Airways to manage the luggage by hand. Reportedly more than 500 flights were cancelled and 23,000 bags lost resulting in cost of £16m for British Airways (Thomson, 2008). In 2002, an NIST report estimated a total cost of $38.3 billion for users due to inadequate software testing infrastructure in the United States alone (National Institute of Standards & Technology, 2002). Despite these numbers, we are aware of no theoretical approach to help decision makers analyze software failure impact in their enterprise.

Significant research has been conducted into prevention of software faults, identification of errors through testing, removal of errors by debugging, and estimation of the cost to the software developer. While the purpose of this effort is to improve software quality, and it might eventually become of interest to the enterprise employing the software, the focus very clearly is from the software developer’s perspective. To the best of our knowledge, however, no research has examined software failures from the enterprise's point of view. The overall goal of this research is to examine software failure impact after a software system has been installed and is in use in an enterprise. The purpose of the current paper is to begin this investigation by gaining a better understanding of software failure through an exploration of different ways that software failure can impact the enterprise. Specifically, this paper develops an initial taxonomy of software failure impact in enterprises and then examines how it can be employed to gain a better understanding for software failure impact. We stress
that we do not take the impact of failures on the developer into account but only focus on the customer perspective, i.e., the enterprise as the user of a software system.

This paper is organized as follows. The next section presents related work on the topic of software failures. In section three, we develop a taxonomy for software failure impact, which we extend for an overall view on software failures in section four. Section five shows how the extended taxonomy can be used to examine software failures, while section six outlines the limitations of our approach. Finally, the paper concludes in section seven.

**RELATED WORK**

In order to establish a clear vocabulary, we need to distinguish between software faults and failures. We will follow the widely used notion of the IEEE standard (Radatz, 1990). A fault is defined as an incorrect process step, process, or data definition. Hence faults appear in the source code of a software system and must not be confused with a failure. A failure is an incorrect result arising from a fault. A failure impact is everything following the failure that is relevant to the enterprise. Figure 1 visualizes the timeline of faults, failures, and failure impacts.

![Figure 1. Time line of faults, failures and failure impact](image)

An overview of different software fault categorizations is given in Ploski, Rohr, Schwenkenberg and Hasselbring (2007). The work presents a variety of approaches that were developed to serve different purposes such as educating developers, testing, and debugging. While this fault-focused approach caters well to the needs of the software developer, it falls short of providing methods to bridge the gap between failures and the faults they result from.

As early as 1991, Chillarege, Kao and Condit (1991) explored the relationship between defects and what they called symptom groups, i.e., the causes of the defect (comparable to our understanding of faults). Symptom groups consist of multiple individual symptoms, which differ on their impact on reliability. Li, Tan, Wang, Lu, Zhou, and Zhai (2006) explicitly examine software failures and how they relate to the faults causing them. Their aim is to analyze changing occurrence rates of failure types over time. They identify five different failure types (which they call impact): hang, crash, data corruption, performance degradation, and incorrect functionality.

Recently, Li, Li and Zhang (2010) presented a categorizing of software defect conditions and defect results. Their notion of defect condition is again comparable to our understanding of faults, while defect results can be seen as failures. This second classification somewhat overlaps their first, yet differentiates further between different kinds of incorrect functionality. Examples of the classification by Li et al. (2010) include “Abnormal graphic interface (no response, cannot display, etc.)” and “Data content error in reading or writing”. Although these classifications are useful to distinguish between failures, they do not provide deeper insight into software failure impact.

**DEVELOPING A TAXONOMY OF SOFTWARE FAILURE IMPACT**

Developing a taxonomy in the field of Information Systems is not a new idea. Examples span a wide range of research areas, recently, for example mobile applications (Nickerson, Varshney, Muntermann and Isaac, 2009) and crowdsourcing (Rouse, 2010; Geiger, Seedorf, Schulze, Nickerson and Schader, 2011). Nickerson, Muntermann, and Varshney (2010) provide an approach to developing such taxonomies. Following their taxonomy development method, we started by examining a subset of software failures. We selected some well-known incidents:

- **Air-Traffic Control System at the LA Airport**: After the shutdown of LA Airport’s main voice communications system, voice contact to 400 airplanes was lost. A backup system that should replace it also shut down after five minutes and left the flight controllers without contact to the airplanes. The problem was caused by an internal counter that counts down milliseconds from 232. When it reached 0 the system stopped functioning. Normally the systems would be reset every 30 days in order to reset the counter to 232, which did not happen in this case (Geppert, 2004). While the passengers were not directly impacted in this case, the potential impact of such an incident is catastrophic.
Continuing the empirical-to-deductive steps of Nickerson et al.'s (2009) approach, we identified the general characteristics of enterprise and affecting, for example, the customers of the enterprise. If, for example, a customer is not able to use his or her mobile phone anymore, as in the Windows mobile phone update failure, the enterprise does not only have to replace or repair the device but may also have to pay compensation to the customer for the time the phone was not usable. Other software failures only impact the enterprise itself and can be seen as an internal issue. The differentiation between internal and external impact is therefore our first dimension of our taxonomy, describing the locality of the failure impact.

The second characteristic is the financial consequences. Software failures may lead to catastrophic financial impact. The list above is somewhat biased here: software failures with very high financial impact are more likely to be reported by the media.

- **Northeast Blackout**: Programming errors in the software managing the electrical network led to a power failure. When multiple systems tried to access the same information simultaneously, the software reacted by sending a busy signal back. Having no emergency management system in place, the system operators did not notice the degradation of the electrical system until it shut down completely (Jesdanun, 2004). The cost of this incident was estimated between $7 and $10 billion (ICF Consulting, 2003). The customers experienced a power outage, which most certainly brought production of goods and services to a halt in almost every case, thus leading to a catastrophic impact from their view.

- **NASA Mars Climate Orbiter**: In November 1999, the Mars Climate Orbiter, part of NASA's Mars exploration program, crashed onto the planet's surface. The incident was triggered by a mathematical mismatch. The probe sent its data to NASA in metric form, while the control center replied in non-metric form. The cost was estimated at $125 million and the spacecraft was lost (Recer, 1999). As the Mars Climate Orbiter was primarily a research mission, no customers were impacted.

- **Ariane 5 Explosion**: 39 seconds after its launch, the self-destruction mechanism of the Ariane 5 rocket was initiated. The rocket worth $7 billion in development cost by ESA (European Space Agency) was destroyed along with the satellites it was carrying. The cause of this was a conversion error. The system tried to fit a 64-bit value into a 16-bit variable and the resulting overflow error could not be handled, which lead to the shutdown of the guidance system. The backup system was confronted with the same problem and shut down too, which lead to the self-destruction (Gleick, 1996). Unlike the Mars Climate Orbiter, Ariane 5 was a commercial mission as it was carrying a television broadcast satellite. The impact for the customer was catastrophic.

- **Lufthansa Check-in System Outages**: The outage of Lufthansa’s central Check-in System caused lags in the check-in process, as well as delays of flights, which ultimately lead to the cancellation of several flights. Since the automated Check-in did not work, passengers had to be checked in manually. Aside from the inconvenience for passengers, Lufthansa had to deploy more than 100 employees to deal with the problems (Oe24.at, 2009).

- **Heathrow - Baggage Tracking Software Incident**: When Heathrow’s baggage handling system crashed, thousands of airline passengers were left without their checked baggage and had to determine what happened to their baggage (Currentals.co.uk, 2008). As described in the introduction, customers suffered from flight cancellations, and the financial impact for the airline was catastrophic. In dollars the loss was $31.2m (using the conversion rate at the date of the report).

- **PC-Virus in Integral Energy's Network**: In October 2009 an attack by a computer virus caused chaos in Integral Energy’s computer network. The software in all desktop computers had to be rebuilt, and the virus spread to the machines controlling the power grid. Fortunately the malware did not affect the power supply, and the virus could be contained within a reasonable time span (Moses, 2009). As the failure was internally confined, it did not lead to any impact on the customers.

- **Update-Failure for Windows Mobile Phones**: In February 2011 Microsoft decided to roll out an update for their mobile phone operating system Windows Mobile. This caused 10% of the affected phones to cease working. Some phones froze during the update process, others couldn’t be restarted afterwards. The phones were left unusable and users had a hard time restoring functionality to their devices (Warren, 2011). While mobile phones represent only one of many areas of operation for Microsoft, for the affected customers it meant that they needed to put in effort and time to get their phones working again.

- **1&1 Servers Down**: In August 2011 all servers of 1&1 – one of the largest server hosting providers in Germany – were down due to an error in a middleware subsystem used to manage the network connection settings of their servers. Servers were down for about six hours and the errors had to be corrected manually by the administrators until a permanent solution could be found (Heise Online, 2011). During the crash, the websites and other services hosted on the servers were unavailable – impacting customers and their potential customers at the same time. For example, an online merchant was unable to receive orders, because his customers could not access the website.

Continuing the empirical-to-deductive steps of Nickerson et al.’s (2009) approach, we identified the general characteristics of these software failures and their impacts. We realized that some failures have an impact traversing the boundaries of the enterprise and affecting, for example, the customers of the enterprise. If, for example, a customer is not able to use his or her mobile phone anymore, as in the Windows mobile phone update failure, the enterprise does not only have to replace or repair the device but may also have to pay compensation to the customer for the time the phone was not usable. Other software failures only impact the enterprise itself and can be seen as an internal issue. The differentiation between internal and external impact is therefore our first dimension of our taxonomy, describing the locality of the failure impact.

The second characteristic is the financial consequences. Software failures may lead to catastrophic financial impact. The list above is somewhat biased here: software failures with very high financial impact are more likely to be reported by the media.
However, we argue that enterprises experience a wide range of financial impacts. For example fixing a system – like the Integral Energy's network – after it has been infected by a virus may be carried out by the staff of the enterprise and thus not lead to additional costs. One might think that external failure always lead to catastrophic financial impact. However, this is not necessarily true as the 1&1 Hosting incident shows. Even though many customers were affected, the outage was within the downtime laid out in the terms of service so no compensation had to be paid. Therefore financial consequences might be insignificant, i.e., leading to cost near zero. The more information that is available, the better one will be able to assess the software failure’s impact. If detailed financial information is accessible, we suggest looking at the ratio of the costs of the failure to the annual revenue of the enterprise. We consider an amount exceeding one year’s revenue catastrophic; everything else with significant cost is considered medium.

Finally, software failure impact can be considered long-term, as it affects the enterprise even after a week. An example illustrating this is the destruction of resources which are hard to replace in the Mars Climate Orbiter case. Other incidents are resolved quickly, as in the failure of the air-traffic control system at the LA Airport.

We include these three dimensions in our taxonomy of software failure impact as shown in figure 2.

Next, following the deductive-to-empirical steps, we searched for new dimensions and characteristics. Reputation plays a role in many of these incidents reported by the media. For example external impacts like the Microsoft mobile phone failure also entail severe consequences for the enterprise’s reputation and thus for future business. However, reputation is highly intangible and a rather subjective measure. We therefore decided not to regard reputation in our taxonomy. Figure 2 is our final taxonomy of software failure impact.

**EXTENDING THE TAXONOMY**

Keeping in mind the triad of fault, failure and failure impact as described in section 2, we felt that the antecedents of failure impact should not be neglected. There are two main questions to be answered: (1) Where does the failure come from? (2) How does the failure express itself in the system’s functionality?

Business owners have a strong need to locate where the failure comes from, i.e., to trace an impact back to the root cause, because they cannot risk that the same fault occurs again in the future. Also, as software becomes less monolithic and more modular, the malfunctioning of one component might not only lead to one specific failure but can appear in different forms of failure. However, from the view of the enterprise, the nature of the fault itself (i.e., the problem in the source code) is less interesting than the responsibility for it. That is, we want to know whether the fault was introduced by an external vendor (“supplier”) of the software or inside the enterprise (“in-house”). The latter applies to very large enterprises with an in-house development division. Usually these enterprises are in need of highly specific software like NASA. However, most
enterprises will – at least for some parts of their operation – also employ software by external vendors. Thus, determining the responsibility for the root cause will help the enterprise to hold their suppliers liable once a failure occurs.

To determine how the failure expresses itself, we look into the system’s behavior during the incident. This is especially important for communications inside the enterprise as well as with customers of the enterprise impacted. This is well illustrated by the last example given above: the customers of the server hosting providers will demand some kind of information on the incident. Out of the different failure classifications shown in section 2, none served our needs. We therefore propose a high-level view, which is sufficiently specific yet concise, based on the classification of Li et al. (2006). During hang and crash the software is unavailable to the user so we subsume these failures into one category called “hang/crash”. Data corruption and incorrect functionality both lead to an “incorrect result” and therefore can also be merged. Performance degradation is a characteristic of its own and therefore we give it its own category “performance impairment”. Figure 3 illustrates our extended taxonomy.

As stipulated by Nickerson, Muntermann, and Varshney (2010), each dimension in the taxonomy contains characteristics that are mutually exclusive and collectively exhaustive. The taxonomy is parsimonious yet contains enough dimensions to provide useful understanding of software failures. Our taxonomy is intended to be used by decision makers in enterprises to determine the impact of software failures. The purpose is to compare software failures, including the preceding error, the failure and the following impact to previous incidents, to help assess the immediate impact, and – if possible – anticipate the long-term impact.

USING THE TAXONOMY TO EXAMINE SOFTWARE FAILURES

We can now use our taxonomy to classify the examples presented above. See table 1. The decisions on how each characteristic is identified are based on the information publicly available. In order to classify the financial consequences of the software failure impact, we also looked at the annual reports of the enterprises for all incident reports that stated a financial impact. No report for the year 2003 was available for FirstEnergy Corp., the regional transmission organization responsible for the northeast power blackout. The oldest available figure was $13.6m revenue in 2008. NASA and ESA are both government agencies, so we looked at the budget rather than at the revenue. In 1999, NASA had a budget of $14b, while the ESA budget for 1996 was not available. In 2010 ESA’s budget was €3.7b ($5b). For Lufthansa in 2009 the total revenue was reported at €595m ($852m) and £9 billion ($13.1b) for British Airways in 2008. Microsoft reported a total revenue of $69.9m in 2011. We used this data to calculate the financial consequences for the incidents, where an impact estimation was available. Although no such estimation was given for the Lufthansa and the Microsoft incidents, we assume that some significant financial consequences occurred but not on a catastrophic level.
The data set in table 1 is quite limited so it is hard to reach conclusions from it. To demonstrate the type of analysis that might be performed we informally looked for groups and relationships in the data. Informal analysis of the impact data showed that internal, insignificant, short duration impact formed a small group, and that external, medium, long term impact formed another small group. These results are extremely tentative, however, since the data set is so small. Informal analysis of the relationship between the failure type and impact indicated that in-house responsibility tended to lead to long term impact and supplier responsibility tended to lead to short term impact. Crash/halt behavior resulted in mainly external impact, while other behaviors did not tend to result in specific impacts. Again, the very small size of the data set makes these results extremely tentative. With a larger data set we may find meaningful clusters and relationships. We leave this analysis for future research.

The list of incidents presented is far from exhaustive. Every week new software failures become public through press reports or enterprise statements. These additional cases can be analyzed using the taxonomy, which can then be employed by researchers as well as enterprises to assess the dimensions of an incident. For enterprises this may be especially helpful for communication with stakeholders such as customers and suppliers (if applicable). Rather than looking at individual incidents, researchers might be interested in a meta-level analysis, i.e., comparing and clustering different incidents or cumulating the impact of multiple cases for an overview.

**LIMITATIONS**

Two limitations have to be considered: the sampling bias and aspects we purposely left out of our taxonomy. First, sampling bias includes two different aspects. On the one hand, the list of examples presented in this paper is far from complete and
only represents a subsample of failures which have been covered by news websites. We would like to see further work from systematic collection of data on software failure impact, which could be a case study inside a single enterprise. The second aspect of the sampling bias is that the examples presented above are all singular events causing extremely high impact. However, there is a second, more complex way in which software failure impact becomes significant: a reoccurring failure that has a small impact by itself but has a large frequency. Each singular occurrence is negligible, yet in sum the impact accumulates to a significant figure.

Small failures that could have a cumulative impact are often overlooked. In the enterprises daily routine, fast failure recovery has the highest priority. In our collaboration in a German manufacturing plant we learned that there is even a term for this: AEG (Ausschalten-Einschalten-Gut, which can roughly be translated into: turn it off-turn it on-everything is fine again). On the production floor, there usually is no time to search for the reason a machine is not running. Also, such failures are considered normal and usually not reported to higher management levels. To restart such a machine, however, about 15 minutes of production time are lost. During our visit, we discovered that such AEG incidents occur every day. When the management learned how much production time is lost over a week, they realized that they needed to direct their attention to these seemingly small failures. From a theoretical perspective, the significance of a software failure impact can be postulated as the relationship between its impact and frequency.

As depicted in figure 4, there are two types of failure impact that can be considered to be of high significance: either a singular large impact incident or multiple negligible incidents. By themselves negligible impacts with low occurrence numbers will not entail significant impact. On the other hand, large impact failures will not occur more than once, as their impact is of such importance to the enterprise, that the underlying cause will be identified and removed from the system. They will also raise high awareness, i.e., preventing similar errors will be important to all stakeholders.

The second limitation of our taxonomy is two aspects not included in our taxonomy: the impact of software failures on reputation and the impact related to security issues. First, although it is an interesting and important aspect of software failure impact, we decided not to regard the impact of software failures on the reputation for a methodological reason, specifically, that the impact on reputation varies greatly depending on the customer of the enterprise. Therefore the loss of reputation is bound to the relationship between two enterprises or of the enterprise with an individual end customer and should not be included in the taxonomy.

The second aspect we intentionally did not include is the impact of security related issues. We did present the example of a virus, but we are aware that data theft is not easily assessed. Security issues may or may not be based on a fault. On the one hand, malicious attacks may exploit vulnerability of the software. On the other hand, accessing data with a stolen password can usually not be prevented by the software. Also, the impact of a security breach may not be visible to the enterprise. Thus assessing the impact of security issues is hard to achieve and out of the focus of our work.

CONCLUSION

In this paper, we showed how faults, failures and their impact relate to each other. We then presented a list of incidents demonstrating software failure impact. Subsequently we created a taxonomy for software failure impact in an empirical-to-deductive way. In order to establish the link between the antecedents of software failure impact and the impact, we extended the taxonomy to include failure type. Two different aspects of software failure were identified: the type of the software failure, described by its root cause location and system behavior, and the impact of the software failure, described by the impact location, financial consequences and the duration. Finally, we showed how this extended taxonomy can be applied to a number of different cases.
We find two important implications of our work. First, software failures and their impact are important to enterprises on a strategic level. So far there has been no approach for analyzing them in a structured way. The taxonomy presented can serve as a tool for this. Second, this work only represents a starting point for future research. Although we proposed some relationships that might exist between the failure type and impact dimensions, we are aware that these can only be seen as a suggestion for hypothesis building. A larger sample size is needed to establish reliable results and cluster the data in a meaningful way. Furthermore, empirical studies are needed to explore the cumulative impact of small software failures.

REFERENCES


