Why Do Software Applications Fail and What Can Software Engineers Do About It? A Case Study

Peter Kueng, Credit Suisse, IT Architecture, 8070 Zurich, Switzerland; E-mail: pkueng@acm.org
Heinrich Krause, Credit Suisse, IT Architecture, 8070 Zurich, Switzerland

ABSTRACT

In many of today's companies application software has become a vital resource to successfully run the business. Since outages of application software may lead to operational and financial difficulties, companies have a vested interest to ensure high availability of their application software. In this paper, data from Credit Suisse, a large commercial bank operating internationally is examined. As a first step, the main causes that led to outages are analyzed. The analysis shows that administration-related tasks are the largest cause for application software outages. On the other hand, the analysis reveals that outages due to hardware failure plays an almost negligible role. In a second step, selected approaches on how availability of applications can be improved are presented. One of the most important means to reduce outages in our case is to address availability not only with system and platform engineers, but with software engineers in particular, since their awareness concerning availability has proven to be limited. Based on that, a set of recommended practices to be addressed by software engineers has been developed; a subset of them are presented here.

1. INTRODUCTION

More and more companies, in particular in the service industry, rely substantially on application software. The product creation, sales and after-sales processes depend heavily on up-to-date application software. Not only do company employees depend on enterprise-owned application software intensively, but external stakeholders such as private and corporate clients, providers, vendors and contractors also depend on such applications. This also implies that in most companies the number of software applications has increased over the years. In addition to that we have to consider that companies today operate in different times zones as their partner and branches are geographically dispersed. This in turn means that time periods for maintenance work have become smaller over time. Furthermore, in the era of the Internet, expectations in terms of availability, timeliness and response time have increased. All of these aspects lead to higher, more ambitious requirements in terms of availability. In short, many of today's applications have to run in a 7 day by 24h mode, the number of outages must be minimal to non-existent, and the time period to repair failures must be shortened.

The rest of the paper is structured as follows: Section 2 shortly describes the company this paper is based on. Section 3 analyses the outages that occurred within the last reporting period and shows the main causes. Section 4 presents some possible approaches on how availability can be improved. Section 5 presents selected software engineering-related approaches the company has chosen to improve availability of its custom-built software. Finally, section 6 summarizes the main points and states the main conclusions.

2. COMPANY PROFILE AND IT INFRASTRUCTURE

The empirical data presented in this case study stem from Credit Suisse, a global bank, operating in over 50 countries and headquartered in Zurich. The two main lines of business are private/retail and investment banking. The company employs over 40,000 people worldwide. The main geographic areas of employment are Asia, USA, and Europe.

To support the many employees, clients, third parties and the underlying business processes, hundreds of applications are in place and running today. Most of the banking-related applications are built and supported by the company's own IT department. The internal IT department not only provides application development and maintenance, but also systems engineering activities such as configuration of servers, middleware, components and networks. All application software runs in the internal data centre.

The number of applications we are concerned about in this study numbers approximately 300. The size of the application software is quite remarkable as it sums up to more than 30 million lines of code. The programming languages used are mainly PL/1 and Java. Some of the COTS applications are based on C and other languages. The runtime platforms are Sun (Solaris) and IBM (z/OS).

3. OUTAGES OF APPLICATION SOFTWARE

In order to improve availability of application software it is essential to know current statistics. For example: how many outages occurred over a certain period of time? What were the causes?

To properly understand the numbers given in Table 1, the measurement approach applied at Credit Suisse is described below:

- Data sources used: To gather the outages during the last reporting period, three different data sources were used: (a) outages reported by users; (b) outages reported via robots (artificial users); (c) outages reported via systems management software (in our case, Tivoli Enterprise Console is used)
- In scope: main banking applications (approximately 300)
- Out of scope: (a) industry-neutral support applications such as Microsoft Outlook; (b) applications used by small number of people (<50).

How is an outage defined? In this paper, the terms *outage* and *unavailability* are used interchangeably. Application unavailability or application outage means – in accordance to IEEE (2002) – that an application is not accessible when it is required for use.[1] 'When required for use' implies two things, for each application it must be known (specified) when the application is required for use and, second, an outage of an application is handled as an outage only if the outage falls into the period of time the application should be available. To practice the first aspect, the SLA[2] levels are used. At Credit Suisse every application must belong to one of three SLA levels. The most demanding SLA level is called 7x24. It means that the application should be available at all times.[3] The second-highest SLA level requires that applications are available from 6am to 11 pm; the third (lowest) SLA level requires that applications are available from 6am to 7 pm. Outages of applications that do not fall into these required up-times are not considered here.

Table 1 shows that a total of 222 outages have been registered during the last reporting period. To identify the hot spots, these 222 outages have been grouped into five categories: administration, software, hardware, environment, and unknown causes. These categories have been used by Gray (1985), who published the well-known causes of failures of Tandem systems. The use of the same categories makes it possible to compare the results.

- **Administration**: 36 percent of outages were caused by administration-related activities. Within this category the main cause for outages lies in operations

Table 1: Reported outages at Credit Suisse and Tandem systems

	Case Study		Results from Gray (1985)
Cause	Number of outages	Percent of outages	Percent of outages
Administration	80	36%	42%
Software	72	32%	25%
Hardware	4	2%	18%
Environment	18	8%	14%
Unknown cause	48	22%	3%
Total	222	100%	103% [1]

and configuration activities. To give a better idea of what is meant by operations and configuration-related causes, some examples of causes – taken from the outage reports – are listed: (a) "Database administration runs a job which exclusively reserved a pointer checker"; (b) "Database xyz was not available because of database reorganization"; (c) "Database reload locked some DB2 tables"; (d) "Change of certificate was not accomplished properly"; (e) "Volume copy action blocks transactions"; (f) "Web server was suspended"; (g) "Role for database access was removed"; (h) "Database files were overwritten due to incorrect software distribution"; (i) "The power for the technology centre wrongly switched off by construction worker!". Another subcategory of administration was labeled "unsatisfactory monitoring". Through a more intensive, and real-time monitoring, some outages of application could have been prevented. "Low on disk space", "too high CPU load", "queues were full" are typical examples that led to outages of applications.

- **Software**: According to our outage reports, errors in software represent the second largest cause that led to unavailable applications. Out of the 72 software-related outages, 37 were caused by erroneous application software whereas most of this software was custom-built. 35 outages were caused by incorrect system software. The rather high 'contribution' of system software is surprising as this category of software runs in hundreds or thousands of companies. What are examples of application software-related causes leading to outages? A few examples are listed here: (a) "Data sharing caused deadlocks"; (b) "An order without an amount was processed. This caused the crash of the program"; (c) "Duplication of a row caused a SQL-811 error"; (d) "Transaction xyz fails due to deadlocks"; (e) "Functional error in application". As mentioned earlier, not only flaws in application software cause outages but also erroneous system software. To illustrate this point again, a few examples are given: (a) "Error in load balancer"; (b) "Deadlocks in DB2"; (c) "Bug in operating system"; (d) "Database block is not working"; (e) "Malfunction of router".
- **Hardware**: Hardware-related outages were very rare. On the one hand, hardware over the years has become more and more reliable. On the other hand, monitoring of hardware components has become more common. In the case of Credit Suisse, only four outages (2 percent) were caused by hardware. In one instance, a broken fan led to an outage of the server, in another case the memory of a server was defective, and in a third case the power supply accumulator was damaged.
- **Environment**: The term 'environment' is vague. It has however been used to make the results comparable to the ones published by Gray (1985). In the case of Credit Suisse outages in the category 'environment' are mostly related to communication problems with company-external systems. A few examples which illustrate this are: (a) "No connection between Reuters and Credit Suisse"; (b) "Wide area network broken"; (c) "Network switch lost connection"; (d) "Fibre channel has been damaged".
- **Unknown**: According to Table 1, some 48 cases (22 percent of all outages) were not able to have been categorized properly. In a considerable number of these cases, it was not possible to identify the root cause of the outage. Many of the so-called unknown causes are transient and could not be traced. Examples are: (a) Server was down; (b) Memory leak; (c) "Hanging process"; (d) "Application not responding".

When comparing the data gathered from Credit Suisse with those published by Gray (1985), some interesting parallels arise. Firstly, both studies identify administration-related work as the primary cause for outages (36 percent vs. 42 percent).[4] The second most important 'contributor' to outages is the category of the software. While 32 percent of the outages are caused by software, this category accounts for 25 percent in the study of Gray (1985). However, Gray does mention in his paper, that this category is probably under-reported.

According to Gray, hardware represents the third category of causes as they caused 18 percent of outages at Tandem systems. In the case of Credit Suisse, the numbers are quite different as hardware-related problems caused only 2 percent of the outages. More than twenty years ago Gray wrote: "In the future, hardware will be even more reliable due to better design, …" (1985, p. 12). Based on our empirical data he was absolutely right.

4. HOW TO IMPROVE AVAILABILITY?

Over the years a large number of approaches, methods, and techniques have been suggested to improve availability and reliability[5] of systems and software. It is *not* the aim of this paper to present the sheer endless pallet of instruments suggested by scholars and practitioners,[6] instead, the aim is to illustrate the *range* of possible approaches, with some (rather accidental) approaches being mentioned in this section.

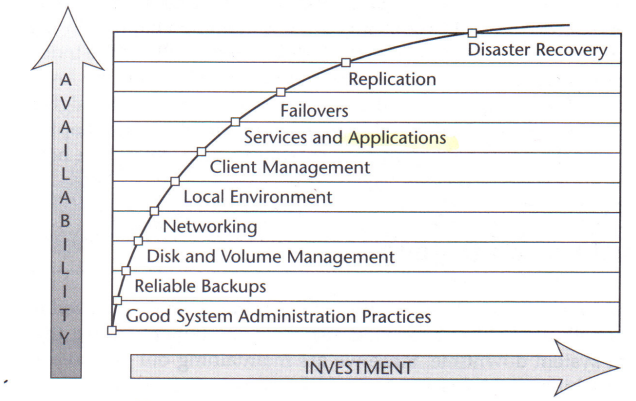
Probably the most commonly recommended approach is the use of *N-Version Programming (NVP)* described by Avižienis/Chen (1977). This approach implies that N-independent programs are executed in parallel on identical input, and the results are obtained by voting upon the outputs from individual programs. In order to ensure the development of independent program versions, different algorithms, programming languages, environments and tools must be used. Theoretical as well as empirical investigations indicate a positive effect of N-version programming (see Cai/Lyu/Vouk (2005)).

According to Viega/Voas (2000), aspect-oriented programming helps to improve availability and reliability as it offers powerful mechanisms for exception handling. Widmaier et al. (2000) propose the use of formal specifications. Green (1997) recommends to improve availability through the use of commercial off-the-shelf components. Candea/Fox (2001) emphasize the importance of recursively restartable systems as they make it possible to restart collections of components/subsystems with little or no advance warning.

By reading the various views that address availability and reliability engineering, one gets the impression that "you just have to do *this*" and everything will work properly. According to Meyer (1999) each community defines "this" differently. Meyer writes:

- *the management school*, which holds that all that really matters is better approaches in management;
- *the formal specification school*, which suggests we won't achieve anything unless we specify everything mathematically—and then we won't need testing at all;
- *the testing school*, which views formal specifications as an academic pastime and believes that the only meaningful solution is to devise systematic testing strategies;

Figure 1. Ten availability technologies (taken from Marcus/Stern, 2003, p.51)



- *the metrics school*, which focuses on assessing everything quantitatively;
- *the open source community*, which believe that only by extensive public scrutiny can we successfully develop reliable software".

The message put forward by Meyer is clear: Although the different communities claim to know what to do in order to improve availability there is no one best approach. Instead, problem and domain-specific approaches are more appropriate.

The richness of possible mechanisms to increase availability is well illustrated in the book by Marcus/Stern (2003); see Fig. 1. This model is conceptual in nature and does not attempt to graph particular levels of availability against specific amounts of investments. However, Marcus and Stern recommend to start with 'technologies' (to use their term) situated at the bottom and to gradually increase.

5. WHAT CAN SOFTWARE ENGINEERS DO TO IMPROVE AVAILABILITY?

As the previous section shows, there is no shortage of potential mechanisms and approaches to improve availability of application software. However, the mechanisms and techniques to be applied depend on various factors; e.g. on the causes that led to outages in the past, on the service or availability levels to be achieved, on the infrastructure in place, on the current software development processes, on awareness of software developers, etc.. Generally speaking, availability can be influenced by two groups of stakeholders, the *systems engineers* who provide the application platform the application runs on, and the *software engineers* who develop the application software; see Fig. 2. In this paper we discuss approaches to be applied be the software engineers.

To improve availability by means of software engineering we formulated twelve so-called 'recommended practices'. To give an idea how the issue is addressed at Credit Suisse some examples of recommended practices are presented here.

Two aspects have to be noted: (a) The recommended practices have not been deduced formally. They are based on both a gap analysis and best practices applied in selected areas. (b) The recommended practices are not universally valid. They fit to our environment.

A. Anticipate Outages of Subsystems and Components

Rationale: Applications are composed of subsystems and components. An outage of such a part should not lead to a full outage of the application. The impact to the user should be minimal.

Tasks for the Software Engineers:

- Ensure that an outage of a subsystem or a component (which are part of the application being built) has no or minimal impact on the use of the application.

Example: In an online-banking application for instance, an outage of the payments module should not impact the functions provided for brokerage.

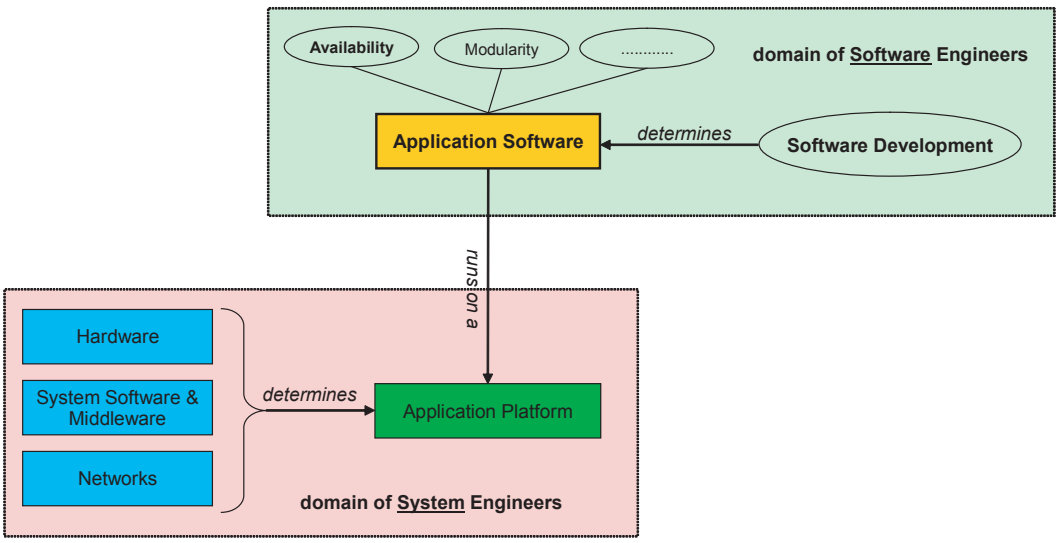
B. Minimize Exclusive Use of Shared Data in Terms of Scope and Time

Rationale: If large portions of a database are exclusively used by a single application, or if small portions of a database are locked for long periods of time, other applications are negatively impacted in terms of availability. Therefore the portion of data exclusively used by one application at a certain moment in time should be as small as possible.

Tasks for the Software Engineers:

- Minimize exclusive use of shared data in terms of scope and time.
- Keep the Logical Unit of Work[7] short.

Figure 2. Leveraging availability by system and software engineers



Example: An end of day batch job must not exclusively reserve tables of an online application.

C. Select Appropriate Processing Mode

Rationale: In terms of processing modes, there are two extremes: *real-time processing* (e.g. on-line transaction processing) and *batch processing* (e.g. end of day processing). In between there are two other options: *asynchronous processing* and *asynchronous processing plus latency*.

Synchronous processing (real-time processing) requires that all components involved are able to execute the requests immediately (i.e. in the same logical unit of work). This may in turn negatively impact availability and robustness of the applications being built. From an availability point of view, the synchronous processing mode should be implemented only when absolutely necessary; i.e. when immediate response and timeliness of data are critical.

Tasks for the Software Engineers:

- Use asynchronous processing as much as possible.

Example: To create a statistical report, synchronous processing is not required. From an availability point of view, asynchronous processing plus latency is more appropriate.

D. Test Behavior of Application when Selected Components and Interfaces are 'Switched Off'

Rationale: It must be guaranteed that mechanisms which have been implemented to treat unavailable or malfunctioning components/interfaces function as planned. Failures of one or several parts of a system must be observed. The planned utilization of imperfect parts (components, interfaces, etc.) is sometimes called "fault injection testing".

Tasks for the Software Engineers:

- Identify the application-*internal* and application-*external* availability-critical components and interfaces that may fail. Test the behavior (robustness) of the application by 'switching-off" (or not providing) selected components and interfaces.
- Specify the required test environment and test cases.

Example: When an online-banking application is to be tested, the interface providing the current exchange rates should be switched off in order to check whether the application's behavior is compliant to the design.

E. Build Applications that Require No Manual Administration

Rationale: Since human error is a leading cause of downtime, one important way to improve availability is to reduce the number of mistakes that humans (administrators) can make on critical systems. By making systems simpler you do just that. Simpler systems require less administrative attention, so there is less chance for human error which can lead to an outage of the application, see Marcus/Stern (2003), p. 103.

Tasks for the Software Engineers:

- Build applications that require no manual administration or intervention.
- If manual administration cannot be made obsolete, incorporate a mechanism that checks the outcome of tasks performed immediately. It must not be the case that an incorrect execution is detected with delay.
- Design all applications in such a way that they can be run in a 7x24h mode.

Example: If an application has been designed in such a way that a periodical reorganization of the database has to be initiated manually, then the principle mentioned above is not met.

6. CONCLUSIONS

The analysis of 222 outages of application software shows that (a) administration-related task are the largest cause of application software unavailability, (b) imperfect software (custom-built application software and system software) represents the second largest contributors, (c) environment-related causes 'occupy' the third place, and (d) outages of applications caused by defective hardware is very rare in the company considered.

As there is no best approach to improve availability, the various options have to be considered. They include *systems management-related approaches* such as disk and volume management and the use of redundant hardware as well as more *application software-related approaches*. In our case the focus has been put on the second category, since the potential to improve availability has been considered as being most effective in this area.

Where do we stand today at Credit Suisse? Twelve so-called recommended practices have been defined and communicated to software engineers. None of these recommended practices require additional infrastructure in order to be implemented. The enforcement of these practices takes place on the one hand via company-internal education. On the other hand, software engineers have to demonstrate just how the recommended practices have been applied in their software development projects. The initial feedback from software engineers has been quite positive. They now see the software development discipline broader than before (as many of them have not spent a lot of attention to availability issues in the past) and, not of lesser importance, the teamwork between system engineers and software engineers has become closer.

The primary conclusion is as follows: Every company who builds and uses application software to a larger extent should analyze the causes of outages carefully. Based on both the cause analysis and the mechanisms already in place, company-specific measures can be taken. If these two steps are omitted, inadequate or even useless actions might be implemented, which in turn, lead to unnecessary costs. Our impression is that the power of *system and hardware-related mechanisms* to improve availability is overestimated, while *software engineering-based approaches* are not adequately rated. One reason behind that behavior might be that some mechanisms addressing the first category can be bought, while the second category has to be addressed mainly via awareness, motivation, education, inspiration, and knowledge.

Finally, our literature research has revealed that the issue of availability of application software has not been intensively addressed up to now. In particular, software-engineering related approaches, to ensure high availability of large sets of applications, have been treated only fragmentarily. From our perspective, this in turn means that it would be fruitful for the software engineering community to further consider this topic in future research and teaching activities.

7. REFERENCES

Avižienis/Chen (1977). Avižienis, Algirdas; Chen, L.: On the Implementation of N-Version Programming for Software Fault Tolerance During Execution. *Proceedings of the Computer Software and Applications Conference, Annual International (COMPSAC'97)*, IEEE, pp. 149-155.

Candea/Fox (2001). Candea, George; Fox, Armando: Designing for High Availability and Measurability. *Proceedings of the 1st Workshop on Evaluating and Architecting System Dependability (EASY'01), IEEE.*

Cai/Lyu/Vouk (2005). Cai, Xia Cai; Lyu, Michael R. Lyu; Vouk, Mladen: An Experimental Evaluation on Reliability Features of N-Version Programming. *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISRE'05),* pp. 161-170.

Gray (1985). Gray, Jim: *Why Do Computers Stop and What Can Be Done About It?* Tandem Computers, Technical Report 85.7, June 1985.

Green (1997). Green, Paul. The Art of Creating Reliable Software-based Systems using Off-the Shelf Software Components. *Proceedings of the 16th Symposium on Reliable Distributed Systems (SRDS'97),* IEEE, pp.118-120.

IEEE (2002). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990(R2002).

Lyu (1996). Lyu, Michael: *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill, 1996.

Marcus/Stern (2003). Marcus, Van; Stern, Hal: *Blueprints for High Availability*. Wiley Publishing, Indianapolis, 2003.

Meyer (1999). Meyer, Bertrand: Every Little Bit Counts – Toward More Reliable Software. *IEEE Computer*, Vol. 32, Issue 11, Nov 1999, p. 131-135.

Oppenheimer et al. (2003). Oppenheimer, David; Ganapathi, Archana; Patterson, David: Why Do Internet Services Fail, and What Can Be Done About It? *Proceedings of the Usenix Symposium on Internet Technologies and Systems (USITS'03)*, pp. 1-16.

Viega/Voas (2000). Viega, John; Voas, Jeffrey: Can Aspect-Oriented Programming Lead to More Reliable Software? *IEEE Software*, Vol. 17, Issue 6, Nov/Dec 2000, pp. 19-21.

Widmaier/Smidts/Huang (2000). Widmaier, James; Smidts, Carol; Huang, Xin. Producing More Reliable Software – Mature Software Engineering Process vs. State-of-the-Art Technology? *Proceedings of the 22nd International Conference on Software Engineering (ICSE),* ACM Press, 2000, pp. 88-93.

ENDNOTE

[1] Availability is defined as follows: "The degree to which a system or component is operational and accessible when required for use." IEEE (2002).

[2] SLA stands for *S*ervice *L*evel *A*greement

[3] To be precise, even in a 7 by 24h operation mode short periods of time for maintenance work (e.g. deployment of new versions) are inevitable. These periods of unavailability depend on the type of software. E.g. for the software that controls the cash machines the interruptions should be below one minute.

[4] The relevance of administrator-related failures is further supported by the empirical study of Oppenheimer et al., (2003).

[5] "Reliability. The ability of a system or component to perform its required functions under stated conditions for a specified period of time." (IEEE, 2000).

[6] A good overview can be found in Lyu (1996)

[7] A logical unit of work is a set of transactions where either all are successfully applied against the database, or none have any impact on the database.

[8] This number (103%) has been taken from Gray (1985).