Software Foul-Ups Bleeps and Blunders

Ernest Mnkandla

Department of Computer Science

University of South Africa

mnkane@unisa.ac.za

Abstract

Today computing software has improved our quality of life as it is found in all technologies we use. With the rise of the Internet over the past few decades, software engineering research focused on determining the quality of engineered software as opposed to hacking. This focus gave rise to software development methodologies such as solo software, open source software and agile. Despite such a focus, software disasters also increased. Software development training therefore grew to emphasise on software quality management (SQM), teaching a medley of craft and science to build programming skills, soft skills, teamwork and self-management skills. Poor training in some of these skills led to a number of software failures. Some of the software blunders have resulted in hospital accidents, war accidents, and losses in investments. This lecture presented an overview of software failures and blunders and possible solutions based on effective software quality management (SQM). As we enter the era of the Internet of Things (IoT), Fourth Industrial Revolution (4IR), and big data, a focus on SQM is inevitable to avoid software disasters.

Introduction

Today we live in a world where computing software has undoubtedly taken centre of every technology used by humans for anything. In the past few decades when it became clear that a lot of software would be developed for web-enabled systems, software engineering research took great interest in determining if such software was engineered or simply hacked. This period saw the rise of practices such as, solo software development, open source software development, and agile software development among others. Today it is clear that agile development has become a leading approach to the development of software. The need to develop software following engineering practices cannot be over emphasised. However, software engineers cannot ignore the fact that software development skills are not just technical but a medley of craft built on technical knowhow, soft skills and teamwork, and organisational and self-management skills. Developing quality software therefore, entails honing together all these skills and finding the sweet spot between relevant professional ethics, business needs, and customer requirements.

It is interesting to note that the history of software has a number of software failures of note, ranging from loss of life to loss of money and loss of comfort somewhere in between. This paper presents an overview of software failures and blunders with the aim of proposing possible solutions from the perspective of effective software quality management. This research is justified by the fact that we are moving into a world where almost any device used by humans is most likely to have software, for example in the cases of Internet of Things (IoT),

Fourth Industrial Revolution (4IR[1]), and big data related technologies, software disasters can result in disasters of large magnitudes.

Background

There can be no doubt that computing software has improved our quality of life as human beings as we know it today. Think of inventions such as social media, transport, industrial machinery, learning systems, household appliances, communication systems, war systems, security systems, and health to name but a few. Without software, human life would be very different from what it is today. Developing software requires skills in mathematics from which logical reasoning is attained together with problem-solving skills. Another requirement for software developers is an aptitude for programming languages, i.e. being able to develop in different languages. Software developers cannot be successful if they do not possess excellent organisational and time management skills and the ability to work in teams. Software development also requires skills in accuracy and attention to detail. A software developer must always understand the latest trends and apply self-development skills to keep up to date with fast-changing trends that are applicable to commercial environments.

Software development

Software engineering: Applying engineering principles to the design, development, testing, deployment and management of computing software.

Software development: A process with activities ranging from requirements elicitation, to design, coding, project management, testing and quality management.

In order for software developers to be effective today, develop quality products, and remain relevant in future, they need certain skills in a number of areas. Such skills are needed due to the complexity of the application needed for various devices and environments, ubiquity of software and security requirements. Besides all this, software developers need to observe professional ethics.

The current set of skills needed by software developers are (Redbytes, 2018):

- Mobile application development: be skilled in a few top programming languages such as BuildFire.js, Python, Java, PHP etc.
- Creative practices in UI/UX design: creating visually appealing interfaces with flowing navigation, creating a wow moment kind of user experience.
- Familiarity with AI and machine learning: knowledge of AI and ML will enable fast development of software that has intelligence.
- AR Apps design and development: application of augmented reality to education and training has raised its interest in mainstream development.
- Special exposure in data analytics and science: skills in analysing large quantities of data will make the developer very relevant to the current and future industry.
- Specialised knowledge in cybersecurity: software solutions must be designed to protect against cybersecurity vulnerabilities.

[1] An industry era that uses cloud computing, mobile technology and big data for production.

- Excellence in SaaS and cloud computing: such technology skills will differentiate developers who are able to apply today's technologies to their development work.
- Adoption of digital transformation and IoT: the ability to implement applications across various digital technologies is inevitable especially with the rise of the Internet of things.
- Enthusiasm for coding and engineering: Coding is very technical and specific and software engineering is about building high quality code, without passion, this can be a challenge.
- Flair for programming languages: master more than one language in order to be able to grow a career in software development.
- Functional communication: Software developers can only be effective in their projects if communication is the core the entire software lifecycle.

Software failures

Despite all the technical skills used to develop high quality software, failures have happened and may continue to happen. The most common software problems that result in faults are (Perry & Rice, 2003):

- Incorrect calculations: seen in mathematical functions used in cases such as financial and date calculations.
- Incorrect data edits: the software does not apply existing data edits correctly.
- Ineffective data edits: is when data edits are in place and working correctly, yet still fail to prevent incorrect data from being entered into the system.
- Incorrect coding/implementation of business rules: the mistakes that occur between what is intended to be developed or implemented and what is actually delivered.
- Inadequate software performance: slow system response times and transaction throughput rates.
- Confusing or misleading data: the data shown to users may be correct, but the users might not fully understand how to interpret the data.
- Software that is difficult to use: software that is cumbersome, difficult to navigate, and requires several steps to perform simple tasks.
- Obsolete software: no longer works due to new hardware or support software changes.
- Inconsistent processing: only works correctly in one environment.
- Difficult to maintain and understand: the ability of a programmer or developer to maintain the software.
- Unreliable results or performance: the software does not deliver consistently correct results or cannot be depended to work correctly each time it is used.
- Inadequate support of business needs or objectives: software that is inflexible to meeting business needs.
- No longer supported by the vendor: vendor has stopped to support a particular software product.
- Incorrect or inadequate interfaces with other systems: the software does not correctly accept input (data, control, parameters, etc.) from other systems or sends incorrect output (data, control, parameters, print, etc.) to other systems.
- Incorrect matching and merging of data: data is obtained from one source and matched or merged with data from another source.

- Data searches that yield incorrect results: a search retrieves incorrect data as the result of a search.
- Incorrect processing of data relationships: data relationships are not created or maintained correctly between one or more data elements.
- Incorrect file and data handling: software incorrectly retrieving data from files or tables.
- Inadequate security controls: unauthorized access to the system is not adequately controlled and detected.
- Inability to handle production data capacities: software's inability to process data at the level required by the organization.

Examples of software failures

Those who have studied computer science or at least software engineering are aware of the classic examples of software failure usually taught and sometimes even included in some textbooks. A good example is the Therac-25, a computer-controlled radiation therapy machine produced by Atomic Energy of Canada Limited (AECL) in 1982. In the early 80s, the AECL machine gave patients radiation doses that were many times above normal, which killed patients or resulted in serious injury (Leveson & Turner, 1993).

St. Mary's Mercy Medical Centre's Hospital Patient Management System had a glitch that sent messages saying each of the patients who had a procedure done between 25 October and 11 December 2002 had died. It was in fact a software blunder, those patients were alive.

Knight Capital Group (KCG), an organisation that had a reputation in its industry until 01 August 2012 when in 30 minutes their trading algorithms blundered and bought high and sold low on 150 different stocks. KCG lost $440 million on trades before the foul-up stopped. This resulted in a 62% drop on their stock price in a day, according to Bloomberg Businessweek.

On a cool autumn night of September 26 1983, the Soviet nuclear early warning system bleeped mistakenly reporting that the United States of America had launched an attack on the Soviet. This could have started world war III, because war between Russia and the US in those years would have meant war for everyone. Fortunately, the Air Defence officer on duty that night had a gut feel that the alarm was false and it was indeed false. Thanks to lieutenant colonel Stanislav Petrov the man who saved the world from nuclear war.

In 2012 Apple's iOS 6 update ditched the superior Google Maps platform to introduce the new Apple Maps. It was considered the most embarrassing least usable piece of software Apple had ever developed; their maps took people nowhere. A report by Talking Points Memo's IdeaLab the software was "missing entries for entire towns, incorrectly placed locations, incorrect locations were given for simple queries, satellite imagery was obscured by clouds etc."

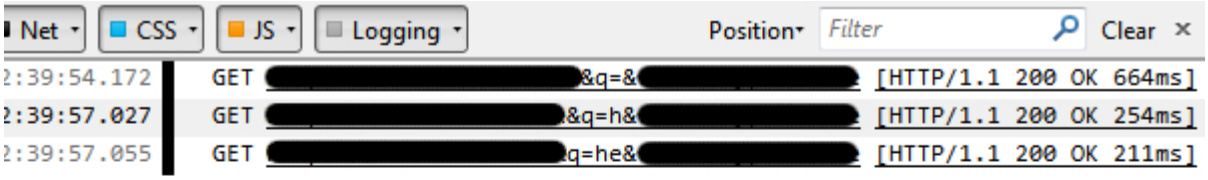
Michigan Department of Corrections granted early release to 23 prisoners due to a computer programming glitch. Prisoners were released 39 to 161 days earlier. The authorities consoled themselves by saying that it was not murderers, but non-violent criminals. However, the same cannot be said about California State who gave “non-revocable” preference to approximately 450 violent criminals due to a mistake in its computer program.

On August 14, 2003, a blackout across eight US states and Canada affected 50 million people. What caused it? A race condition bug (a condition that occurs when “two separate threads of a single operation use the same element of code.”) Without proper synchronization, the threads tangle and crash a system. The condition resulted in 256 power plants going offline and consequently there were disruptions in cellular communication. Guess who survived the communication blockage: a laptop using a dial-up modem.

Illustration of a race condition in AJAX

AJAX (**A**synchronous **Ja**vaScript and **X**ML). A technique for creating better, faster, and more interactive web applications using XML, HTML, CSS, and Java Script. https://bugs.jqueryui.com/ticket/8234

1. Remember: in digital systems: similar to race conditions in JK flip-flops; where if J=K=1 & clock is too long, (state toggles between 0-1, 1-0, etc.) the current output is uncertain, confusing. Solved using master-slave.
2. There is an artificial delay (500 ms) inserted when the search term is 1 character long.



"h" requested at 57.027, returned in 254 ms, @ 57.281
"he" requested at 57.055, returned in 211 ms, @ 57.266

"h" requested
"he" requested
"he" returned
"h" returned

UI shows "h" result set and "he" in the input field

We will not mention NASA’s Mars Climate Orbiter that was on its mission to Mars in 1998, it was lost in space after a sub-contractor on the engineering team failed to make a simple conversion from English units to metric. The result was the loss of the $125 million craft, it got fatally close to Mars’ surface after attempting to stabilise its orbit too low and eventually lost communication.

Those around my age or older would remember the news heading" *Patriot Missile System Timing Issue Leads to 28 Dead*"

This was tragic computer software blunder on our list occurred on February 25, 1991, during the Gulf War. While the Patriot Missile System was largely successful throughout the conflict, it failed to track and intercept a Scud missile that would strike an American barracks. The software had a delay and was not tracking the missile launch in real time, thus giving Iraq's missile the opportunity to break through and explode before anything could be done to stop it. Consequently 28 people died and more than 100 were injured.

Windows genuine bleep:

Very common even today, however, in 2007 this incorrect flagging of thousands of genuinely legal copies of Windows as pirated is said to have occurred after programmers inserted a glitch in the anti-piracy tool, Windows Genuine Advantage (WGA). According to Microsoft, the mistake arose after a member of the WGA team incorrectly uploaded bug-ridden pre-production software onto the company's servers on a Friday afternoon.

On June 4, 1996, an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. Ariane explosion The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million. The cause of the crash:

A software error in the inertial reference system. A 64-bit floating-point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than 32,767, the largest integer storable in a 16 bit signed integer, and thus the conversion failed.

Video Demos

1. Ariane spacecraft failure (https://www.youtube.com/watch?v=AGI371ht1N8)
2. TDD (https://www.youtube.com/watch?v=2Ekty7t621k)

The “what ifs” of software failures

Consequences of software failure

Human life

Environmental

Financial

Other assets

Liabilities in case of software failure

The issues surrounding legal implications resulting from losses or damages resulting from defaults in software have been debated since the 1980s. The liabilities depend on the laws of the country. In most cases, a customer may claim negligence of duty by software developer if the customer can establish that the developer’s actions fell short of what a reasonable software developer would have done to ensure their software is free from defects. This is obviously not an easy task considering that many users of software have little knowledge of what developers do.

In general, consumers in South Africa are protected under the Consumer Protection Act No 68 of 2008: (CPA). A consumer would need to be aware of the guarantee expected from services; these usually differ from the guarantees expected from goods.

Reading the terms and conditions of the software is very important but often neglected.

Solutions to Software Failure

Solutions are from two perspectives:

After the failure: where possible, recourses are considered as agreed in the terms and conditions or contracts where applicable. Recovery plans in case of organisations may help depending on the consequence of the failure.

Before the failure: this is favourable as it also costs less than the one after. This is usually done through effective software quality management. Software quality management is one of those activities that are performed from the start to the end of the software development process, i.e. from initiation to deployment of the software.

Software Quality Management (SQM)

SQM involves three processes:

- Quality assurance (QA), which are organisational processes followed to meet software development standards.
- Quality planning, which is the definition of quality attributes at project level that need to be implemented in order to reach the project goal.
- Quality control (QC) involves the tests and reviews of software done to ensure QA.

- SQ has two distinct but related aspects:
  - Functional quality: The software's fitness for purpose. How does it compares to competitors in the marketplace, is it a worthwhile product? It is the degree to which the correct software was produced. Tested using static tests such as software reviews.
  - Structural quality: It has a lot more to do with the degree to which the software works as needed.
- Functional quality: designed to meet functional requirements and is checked using dynamic testing such TDD.
- Structural quality: designed to meet non-functional requirements, e.g. robustness or maintainability and is checked using static testing such as inspections, code reviews etc.

I will explain and illustrate two dynamic testing techniques TDD and Juni.t

TDD as depicted by the name: is about writing better quality code by first thinking through how you want to test it, then you code (Testing drives the development):

The simple TDD process is:

- Write just enough code so it runs.
- Think through how to test it.
- To do this, you write test cases for it:
  - Test cases should fail at first because there is no implementation.
- Write code until test cases pass. You are done, that is TDD.

Let us see a video that shows you how to:

- Create interfaces for various kinds of arithmetic operations.
- Create a class to implement those interfaces.
- Create a JUnit test class to test the expected results.
- Write code until tests cases pass.

In general, good project management for software projects, use of known and proven industry standards, methodologies and practices such as agile methodologies, DevOps, Capability Maturity Model Integration (CMMI), ISO/IEC 9126:1991 etc. will contribute to a good quality product.

Finally, over the past decade, my students and I have contributed to the improvement of software quality in agile methodologies, recently in DevOps, and we have also developed machine learning algorithms to improve the quality of software.

Table 1: Selected Contributions in Software Quality

Authors	**Title**	**Journal /Conference**	**Year**
Hanslo, R. & Mnkandla, E.	Scrum Adoption Challenges Detection Model: SACDM	FedCSIS 2018	2018
Gaoussou, AKD. & Mnkandla, E.	Facilitating the management of Agile and DevOps activities: Implementation of an agile data consolidator	icABCD 2018	2018

Chiyangwa, T.B. & Mnkandla, E.	Modelling the critical success factors of agile software development projects in South Africa	EJISDC	2017
Sebega, Y. & Mnkandla, E.	Exploring Issues in Agile Requirements Engineering in the South African Software Industry.	EJISDC	2017
Teklemariam, M.A. & Mnkandla, E.	Software Project Risk Management Practice in Ethiopia.	EJISDC	2017
Mpofu, B & Mnkandla, E.	Software Defect Prediction on a Search Engine Software Using Process Metrics.	ICACCE 2016	2016
E. Mnkandla	A review of Communication Tools and Techniques for Successful ICT Projects.	AJIS	2014
Mwansa, G. & Mnkandla, E.	Migrating Agile Development into the Cloud Computing Environment	IEEE CLOUD 2014	2014
Hans, R.T. & Mnkandla, E.	Modeling Software Engineering Projects as a Business: A Business Intelligence Perspective	IEEE AFRICON 2013	2013
E. Mnkandla	Assessing a methodology's project risk management competence	JCM	2012
E. Mnkandla & C. Marnewick	Project management training: the root cause of project failures	JCM	2011
E. Mnkandla	About Software Engineering Frameworks and Methodologies.	IEEE AFRICON 2009	2009
E. Mnkandla, B. Dwolatzky	Defining agile software quality assurance	ICSEA'06	2006
Mnkandla, E. & Dwolatzky, B.	A Survey of Agile Methodologies	Transactions of the SAIEE	2004
Mnkandla, E. & Dwolatzky, B.	Balancing Human and Engineering Factors in Software Development.	IEEE AFRICON 2004	2004

My future plan is to do what professors do.

Conclusion

Let me end by recapping what you have learnt today:

- What software engineering is
- What is involved in software development
- What happens if quality is neglected or if sloppy code is used in critical systems
- Possible legal consequences of developing faulty code.
- How to ensure quality in your code (the TDD Junit example)

References

Casale, G., Chesta, C., Deussen, P., Di Nitto, E., Gouvas, P., Koussouris, S., Stankovskia, V., Symeonidisa, A., Vlassioua, V., Zafeiropoulos, A., & Zhao, Z. (2016). Current and future challenges of software engineering for services and applications. *Procedia Computer Science*, *97*, 34-42.

Hoda, R., Salleh, N., & Grundy, J. (2018). The Rise and Evolution of Agile Software Development. *IEEE Software*.

Leveson, N. (2001). Systemic factors in software-related spacecraft accidents. In *AIAA Space 2001 Conference and Exposition* (p. 4763).

Leveson, N. G., & Turner, C. S. (1993). An investigation of the Therac-25 accidents. *IEEE computer*, *26*(7), 18-41.

Perry, W. E., & Rice, R. W. (2003). Testing Dirty Systems.

Redbytes. (August 20, 2018). Top 11 Skills in Demand for Developers in 2018. REdbyte Career Blog. Retrieved from https://www.redbytes.in/software-developer-skills-2018/

Salonek, T. (2013, July 15). Top 15 Worst Computer Software Blunders. Intertech Blog. Retrieved from https://www.intertech.com/Blog/15-worst-computer-software-blunders/