Failure Analysis in the field of Computer Science

Charles Richardson

Herbert Wertheim College of Engineering, University of Florida, Gainesville, Florida

crichardson5@ufl.edu

**TABLE OF CONTENTS**

[**INTRODUCTION**](#_kthuqbblah0k) **1**

[**COMMON FAILURES IN COMPUTER SCIENCE**](#_va3ydqhjwi94) **2**

[Types](#_mcs9juaxyk0l) 2

[Causes](#_grs42y6bqgw) 2

[**FAILURE ANALYSIS METHODS**](#_ummc78rfu99w) **3**

[**CASE ANALYSIS**](#_410ozgbvaz08) **4**

[Description](#_8ob5wgpdtphl) 4

[Investigations](#_71z16y3z2sa5) 5

[Recommendations](#_3pfyz4jvo2ij) 7

### INTRODUCTION

As digital reality continues to overtake the status quo, it is becoming increasingly important to be selective over what we digitize. Overestimating computational systems and their developers has been an issue in the software development industry for over 2 decades. Technical limitations are rarely the root cause, but rather the poor execution of these technical processes. Fortunately, many individuals and organizations devote energy to ensure they execute their tasks correctly, helping avoid bigger hurdles later down the line. This is commonly done through plenty of testing and failure analysis.

### COMMON FAILURES IN COMPUTER SCIENCE

Computers need clear instructions, zero ambiguity, and often contain many interdependencies. All of these variables present plenty of room for error. Mistakes can occur in any step of the development cycle of a software project, they can even happen before the project begins.

#### Types

The most common type of failure in a computational system is called a **bug**. A bug is a slight imperfection in the code. It is thought that approximately 35-50 percent of a developer's time is spent debugging - removing bugs - accounting for 50-75 percent of the total budget of software development projects [2,3]. Matters are made worse when errors are ignored in development and not documented. After growing the project a bit more, the error resurfaces, this time hidden by all the added complexity, requiring more time to find and solve.

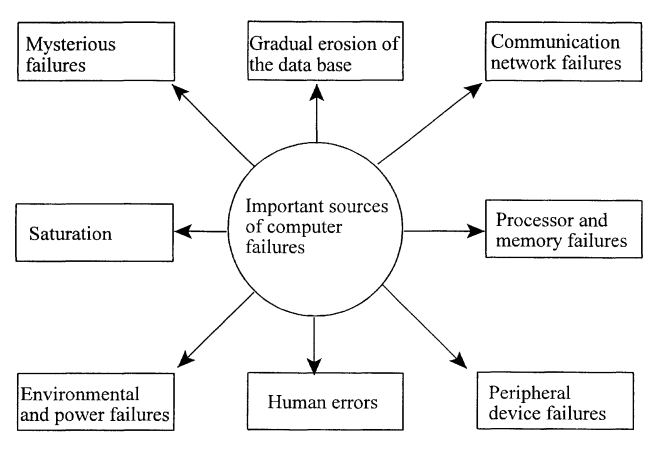
Design errors are the second leading failure to plague developers. Often, they do not reveal themselves until the systems are in use and serving tens of thousands of clients. Design errors are the ones that can make a business go bankrupt, since they are put in an impossible position of spending time starting a new product from scratch, or spending more money replacing big chunks of the system. On top of this decision, they are forced to handle their public reputation [3].

#### Causes

Software projects are often new ideas, thus n0t standardized. Excitement is high and predictability is low for outsiders. Therefore, the manager can inflate the capabilities of the project, pull back deadlines, and lie about budgets to make the project sound better than it really is [3]. Inaccurate projections can lead to poorly defined systems of requirements, inconsistent data definitions, input data ranges and inadequate validity checking [1]. Problems at the beginning of project development are the most dangerous, because they pull the most resources into a faulty project that may not appear as a failure until the resources have been exhausted.

Communication is essential to keep the project moving in the right direction for everyone. As obstacles appear during the development of the project, everyone must communicate with each other. The managers, developers and the users must all be in correspondence with one another over priorities, bugs and requirements of the project. Sloppy development practices, the use of immature technology, and the inability to handle the complexity of the project are all issues that can be lessened through proper communication.

A final, and more pressing cause of failure in a project, is stakeholder politics [1]. Those that are not participating in the project with their time, but rather with their money. These individuals and organizations oftentimes do not understand the project management and development process and see it as a cost rather than an investment. This can lead to cut corners, bad culture and other issues that can hinder the completion of a project, resulting in failure.

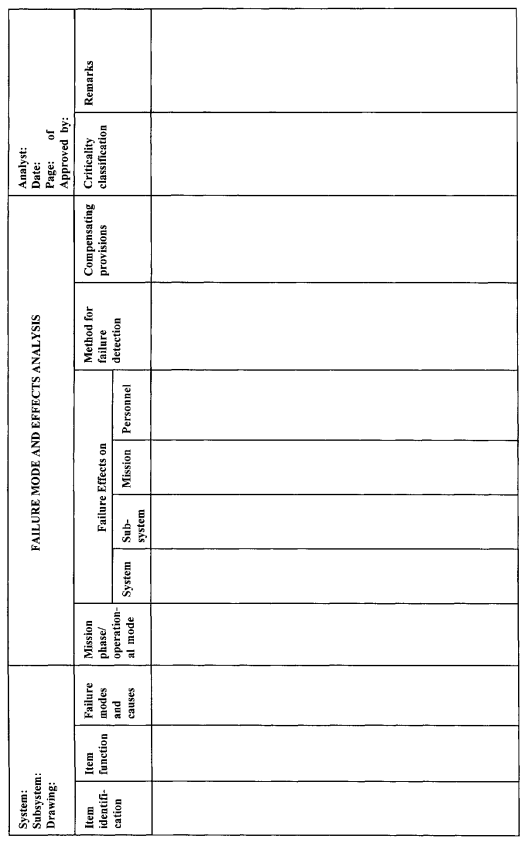


**Figure 1** A general diagram representing sources of computer failures from 2005.

### FAILURE ANALYSIS METHODS

Among all the types and causes of failures, there are common themes that can be analyzed, specifically in regard to software development. Preventative and forensic methods blend into the same tools and methodologies, they are just performed at different times depending on when they are necessary. There are hundreds of ways to test, analyze and detect failures and flaws. Additionally, more are always being added. A few of the most popular are:

* Integration Testing [4]
  + Software modules are tested as a group. Usually done by the testing team.
* System Testing [4]
  + Inspecting both the software and the hardware to verify that the system as a whole meets its specified requirements. Conducted by both the development and testing team.
* FMEA - Failure Modes and Effect Analysis [1]
  + Analyze each potential failure mode to examine the results of the failures on the systems. Can be broken down into 2 levels for software development:
    - Design - Intended to validate the design options for a functional performance requirement
    - System - Similar to system testing, are carried out to ensure the risk of functional failure is at its lowest possible level.
* FTA - Fault Tree Analysis [1]
  + Logical Representation of the relationship of primary events that lead to a specified undesirable event called the ‘top event’.

****

**Figure 2** A FMEA Template [1]

### CASE ANALYSIS

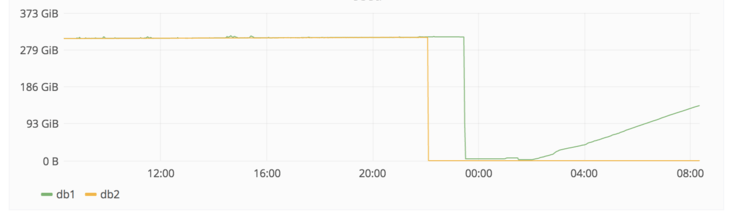
Popular code collaboration service, GitLab.com, with over thirty million registered users [5] faced a combination of a bug, a design error and a human error that sent their world back six hours.

#### Description

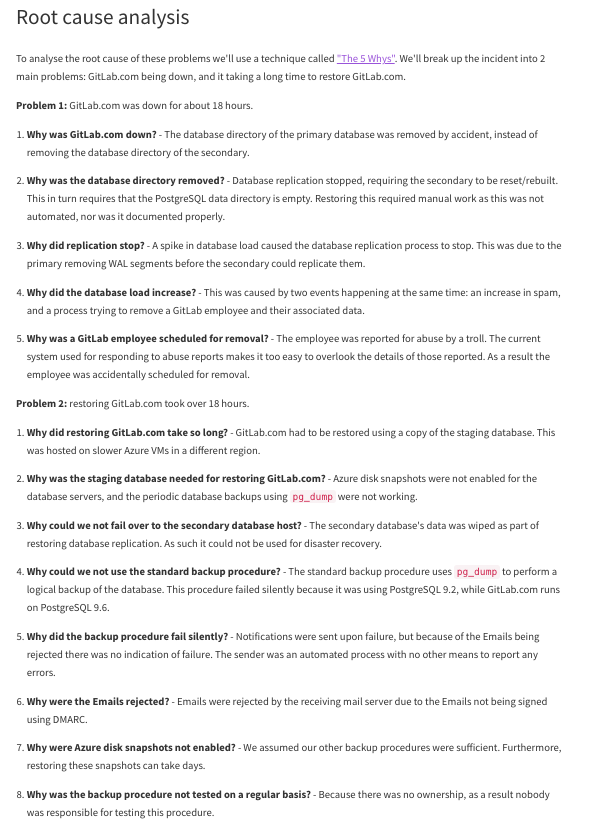
Issues began at 10pm, there was an alert that their database replication stopped. This was due to an excessive amount of writes to the database that were not processed on time by the secondary database, causing it to pause. The secondary database was only behind by 4GB at this point, but it was giving the engineers issues, so they deleted it to ensure a clean replication. After more troubleshooting and clock ticking, the problem was not solved, and the engineers were beginning to get frustrated. Particularly, the one engineer who was working through most of these issues.

Some people have a habit of not stopping a job until they finish it, and it turns out that the frustrated engineer mentioned earlier is one of these people. At 11pm, he considered a solution involving removing the replicated database, even though it was empty. A moment after executing the fix, he realized that he removed the primary database instead of the secondary one. This forced the engineers to shut down the site and begin searching for backups.

Upon the hunt for a backup, the engineers came across more problems. First, LVM snapshots (automatic backups) occur once every 24 hours (one was done manually six hours prior). Second, regular backups also happen every 24 hours, but were not working. Third, disk snapshots (backups) were enabled for the NFS server, but not the database server. Fourth, the replication procedure is fragile, prone to error and badly documented. Fifth, backups to S3 (Cloud storage) were not working. Out of the five backup methods in place, zero were up to date and functional.

**Figure 3** Graph of database storage volume over the day, including the mistaken deletion and the beginning of the reboot.

#### Investigations

Gitlab used a method called “The 5 Whys”, an iterative interrogative technique used to explore the cause-and-effect relationship underlying a particular problem [7]. The strategy of this method is to continue asking why until the root cause is reached.

**Figure 4:**  Gitlabs Failure Analysis [8]

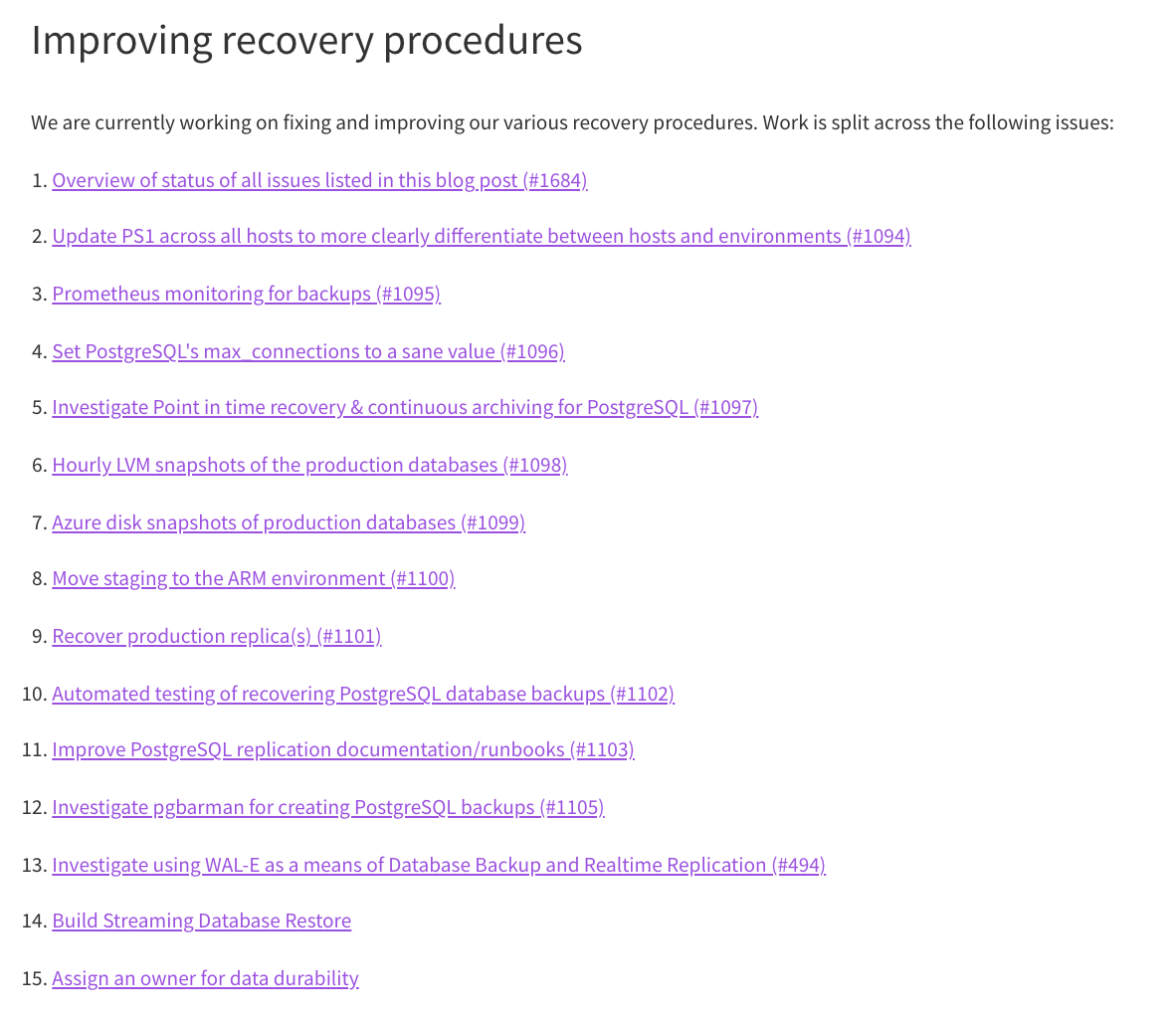
No tests had to be performed to reach the root cause, since they were answered by asking questions. An obvious accident caused a problem that unrooted many other problems behind it, which are now in the light and the company is better because of it.

The failure caused 6 hours of data to be lost, forcing many users to re-work things that they had already done. Further, the GitLab repository is hosted on GitLab, which slowed company progress down as well, given they lost 6 hours of progress. Fortunately, the engineer that made this mistake made a backup. Without that save, even more than 6 hours of data would be lost, which would have extended into a typical 9-5 working day, a time where plenty of contributions are made to repositories.

GitLab kept the public up to date, sharing all the information know at the time, keeping a Google Doc live with updates, and running a live steam on Youtube for the last few fix steps [6, 8]. This reduced any backlash from the news that would put the company in long-term jeopardy. Next, they laid out recovery procedures to ensure the mistake would not occur again. They split work into sections to make it manageable, and assigned it to employees.

#### Recommendations

GitLab showed that failure does not need to have such a poor image. Dozens of media sites made stories on this, and they were for the most part all relating to the transparency of the company with this issue. Given the amount of information this company provided, there is not much more to advise this company. Instead, other companies should follow GitLabs’ footsteps, be transparent about failures and support the community through issues.



**Figure 5:**  GitLabs personal action plan following the disaster

**References**

1. B. S. Dhillon, in *Design reliability: fundamentals and applications*, Boca Raton, Florida: CRC Press, 1999, pp. 126–187.
2. D. H. O'Dell, “The Debugging Mindset,” *Queue*, vol. 15, no. 1, pp. 71–90, 2017.
3. R. N. Charette, “Why software fails [software failure,” *IEEE Spectrum*, vol. 42, no. 9, pp. 42–49, Sep. 2005.
4. Guru, “Types of Software Testing: 100 Examples of Different Testing Types,” *Guru99*, 2020. [Online]. Available: https://www.guru99.com/types-of-software-testing.html. [Accessed: 12-Jul-2021].
5. GitLab Team, “Is it any good?,” *GitLab*, 2021. [Online]. Available: https://about.gitlab.com/is-it-any-good/#:~:text=Tools%20Of%202021-,GitLab%20is%20the%20most%20used%20DevOps%20tool,30%20million%20estimated%20registered%20users! [Accessed: 12-Jul-2021].
6. GitLab Team, “GitLab.com database incident,” *GitLab*, Feb-2017. [Online]. Available: https://about.gitlab.com/blog/2017/02/01/gitlab-dot-com-database-incident/. [Accessed: 12-Jul-2021].
7. Public Source, Ed., “Five whys,” *Wikipedia*, 05-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Five\_whys. [Accessed: 12-Jul-2021].
8. GitLab Team, “Postmortem of database outage of January 31,” *GitLab*, Feb-2017. [Online]. Available: https://about.gitlab.com/blog/2017/02/10/postmortem-of-database-outage-of-january-31/. [Accessed: 12-Jul-2021].