Knights Trading Loss

How a company lost $440 million in 45 minutes



Hack, D. (300364504), Kapadia, K. (300358741), Kumar, D.(300364481), Pearson, W. (300378621), & Yang, M.(300364234)

SWEN223 Report

Lecturer: McKay, J.

## 

## 

## Introduction

This report is about the Knights Trading Loss fiasco that occurred on August 1st, 2012, which lead to the trading firm losing $440 million in under an hour[[1]](#footnote-1). This event caused the Knight Capital Group share price to drop by over 70%, taking the company from being the largest US equities trading company in 2012 to being known as one of the most catastrophic software engineering failures ever to affect the stock market. This report will focus on what the failure was, why the software failure occurred and how the failure may have been avoided. To further develop these ideas topics discussed will include:

* What was the underlying cause?
* How did it result in the failure?
* Why was the fault not discovered earlier?
* Which violation of software engineering practices led to the introduction of the failure?
* If there could have been a way to catch the problem before the damage propagated?

## What was the failure?

The New York Stock Exchange (NYSE) announced a new program called the Retail Liquidity Program (RLP). This new program was to begin on August 1st, 20121 meaning that Knight Capital Group had to prepare new code to allow its customers to take part in this new program. This meant that Knight Capital Group had to write and deploy code before the launch of RPL meaning the deadline could not be extended resulting in rushed development and deployment with little testing.

A significant change was implemented in Knights systems to accommodate for RLP by adding new code to SMARS – a high-speed algorithmic router which would send stock orders to the market. This change was intended to replace code that had previously allowed for a functionality called Power Peg. Furthermore, Power Peg had been discontinued for nine years, which meant that no issues should have occurred as the functionality was not used by any of the system's core protocols. Instead of completely overwriting Power Peg with the new RPL code, a flag for activating Power Peg was repurposed to be used in SMARS without any integration tests.

Once these changes had been integrated, the updated software was deployed to the eight SMARS servers. However, the update was only deployed to seven of them, which meant that while seven of the servers had the correct RLP code, one of them still contained the old Power Peg functionality. As a result, the flag that was repurposed for the new RLP code would activate the Power Peg code on the unmodified server.

Once the market opened, the system began to buy and sell at a rapid rate, capable of sending 8 million orders to market per minute[[2]](#footnote-2). The seven servers with the correct code operated as intended - unlike the server that ran the old Power Peg code. This was unknown to the technician who deployed the new RLP code to the servers, due to a lack of peer assessment, testing and documentation. Power Peg was meant to track child orders against a larger parent order and keep sending orders until the parent order was fulfilled. However, in this new context the method dissolved into an endless loop, sending out millions of inaccurate orders into the market. It took 45 minutes for Knights technicians to shut off the system from the market. During the 45 minutes, the old Power Peg code was pushed back to the servers. This had the unintended effect of drastically worsening the situation as the old flag was still calling the Power Peg code.

Over the 45 minutes, Knights made a loss of $460 million resulting from transactions from 397 million shares. Knights at the time had $365 million in liquid assets, going from the largest trading company in the NYSE to bankrupt. Knights Capital Group still exists today after a $400 million investment from creditors and was acquired by Getco LLC2.

## Why the failure occurred?

Knights did not have systems in place to ensure that human error could be minimised. For instance, after the development of new code, one of Knight's technicians did not deploy the new RLP code onto one of the eight servers. As well as this, the team was not aware that the old Power Peg code had not been overwritten on this server - since there was no review or testing procedure to confirm that the lone technician had done the job properly.

This Power Peg code was discontinued, however, it remained on the servers and was callable based on a flag. The RLP code reused the flag which previously activated the Power Peg code. As the team assumed the old Power Peg code would be removed, they did not expect this code to be triggered once the flag was called. The eighth server however still contained the old Power Peg code meaning that when the flag was called Power Peg was executed.

Another reason that the failure occurred was that the correct warnings were not in place in case of a disaster or fault in the code. For instance, the internal system sent out 97 automated emails that identified an error message titled as "Power Peg disabled", before the 9:30 AM market open[[3]](#footnote-3). However, there were no procedures in place for checking these emails. These messages were not system alerts that might halt the software or inform everyone monitoring the system that something had gone wrong.

Despite the system warnings, nobody bothered to check because there was no procedure in place, nor were they aware of the urgency of the situation. This could be attributed to the lack of supervisory procedures when it came to issues in their platform, or not having a system in place once an unexpected outcome occurred.

The greatest underlying cause of the failure was a lack of documentation or official procedures. This is shown through Knights having no documentation or procedure for deploying new code. This is evident by it being deployed by one technician without any testing or peer review of the operation.

There was no process for testing newly implemented code to ensure that the deployment was performed correctly. Furthermore, Knights were unaware of the maintenance required for their servers. This can be demonstrated by having code that was inactive for over five years on their active servers.

A lack of error monitoring, as well as procedures on what to do in the case of a problem; whether large or small, impacted the time it took for the technicians to be aware and fix the problem. If there were procedures in place, technicians might have reduced the damage of the problem far earlier than the forty-five minutes that it took.

## How the failure could have been avoided?

Numerous software engineering practices were violated by Knight Capital Group. The subsequent paragraphs will discuss how following good software engineering practices, could have helped Knights avoid the fault. These include: testing, reviewing deployments, automated deployment, documentation, extensive version control, maintainability and error prevention/traceability/alerts.

Review and testing are an essential practice when writing code; especially high-risk code. It allows bugs and errors to be discovered before the code is used for its intended purpose. If Knights had tested their new RLP code across all eight servers, they would have found that one server did not have the new code at all. Reviewing code is often far less reliable than testing it, but in this case, it still could have prevented the error before its use on the 1st of August.

Automatic deployment is an effective method when updating elaborate systems with new functions. A well tested and documented process of automatically pushing new code to a system is far superior to manual deployment. Human error can lead to inconsistent deployment over multiple systems. If Knights had a well established automatic deployment method, then all eight servers would have had the code pushed to them correctly, meaning that the fault may have been avoided entirely.

Documentation is a standard procedure for any code base and system. It ensures proper procedures are followed and code can be understood clearly. If Knights had proper documentation, the fault could have been avoided or at least minimised the damage.

Proper documentation could have avoided the failure by ensuring that deployments had to be reviewed by more than one technician, and tested after code completion. Documentation on the topics of troubleshooting and what to do in the case of salient errors could have reduced the amount of time it took to stop the servers from executing the orders to the market, thus mitigating damage.

Version control is a software practice used by teams of software developers to ensure working pieces of code aren't lost or overwritten while someone else is editing the code. Version control is useful for rolling back to previously working code before bugs or errors were introduced. If Knights had proper version control the damage produced by the fault may have been diminished by allowing them to revert to back to previously functioning code quickly and reduce the time of 45 minutes that it took to stop the system.

Maintainability and low coupling are important to allow old code to be easily modified or removed, as well as work with newer code. In large code bases, such as that of Knight's, the need for highly maintainable, low coupled code is even more important. Knights would have benefitted from a flag system that accurately indicates what function they are pointing to, rather than repurposing the Power Peg flag. Consequently, writing the RLP code over the Power Peg code would not have introduced points of failure. Maintaining code like this could have avoided the fault by ensuring that the old Power Peg code wasn't in the system at all, thereby never accidentally being executed by the new code.

Error prevention, traceability and alerts are useful tools to mitigate the damage of a fault as quickly as possible. Prevention means to have functions monitoring the system for abnormal behaviour and stopping the system from worsening. Traceability refers to the ability to track down the central cause of a problem accurately. Alerts are used by the system to ensure that technicians are immediately informed of a problem so they can begin fixing it as soon as possible.

If Knights had a robust error prevention method that analysed their losses over time, it could have shut down the systems before major damage was done. Knights could have benefited from error traceability as well because it would have allowed them to identify the problem and fix it sooner. Better alerts could have set the alarm off earlier that something was wrong.

## Conclusion

Overall, the Knights Trading loss was an easily avoidable software fault with the main problem being that Knights had little documentation or procedures in regards to the creation, maintenance and deployment of their code. There were no procedures that directly addressed updating systems with new code, and there were no procedures or documentation on what to do when there was a full-scale meltdown of this magnitude. If Knights had simply put in place a series of logically sound procedures that addressed how to implement new code at any stage of the process, then this massive error could have easily been avoided. The lesson learned from this is that software of this importance should not be developed "casually", there should be strict procedures and testing requirements for every new piece of code no matter how small a modification it is.

The Knight's Trading disaster was a highly devastating event, but could have been easily prevented with sound engineering practices. This lack of preparation both reveals the importance of strong engineering practices, but yields an important example for engineers to learn from. As detailed in the report, $440 million dollars of losses was incurred over a mere 45 minutes. We can see how the failure was easily instantiated and how the failure could have been avoided fairly easily by following good software engineering practices.

## References

"Knight Capital Group". *En.wikipedia.org*. N.p., 2017. Web. 19 Apr. 2017.

Securities and Exchange Commission. *ORDER INSTITUTING ADMINISTRATIVE AND CEASE-AND-DESIST PROCEEDINGS, PURSUANT TO SECTIONS 15(B) AND 21C OF THE SECURITIES EXCHANGE ACT OF 1934, MAKING FINDINGS, AND IMPOSING REMEDIAL SANCTIONS AND A CEASE-AND-DESIST ORDER*. Securities and Exchange Commission, 2013. Print. Can be found https://www.sec.gov/litigation/admin/2013/34-70694.pdf

Popper, N. (2013, August 2). *Knight Capital Says Trading Glitch Cost It $440 Million*. Retrieved from <https://dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/?_r=1>

Seven, D. (2014, April 17). *Knightmare: A DevOps Cautionary Tale*. Retrieved from [https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale](https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/)/

Heusser, M. (2012, August 14). *Software Testing Lessons Learned From Knight Capital Fiasco*. Retrieved from <http://www.cio.com/article/2393212/agile-development/software-testing-lessons-learned-from-knight-capital-fiasco.html>

Python Sweetness, (2013, October 22). [*How to lose $172,222 a second for 45 minutes.* Retrieved from http://pythonsweetness.tumblr.com/post/64740079543/how-to-lose-172222-a-second-for-45-minutes](http://pythonsweetness.tumblr.com/post/64740079543/how-to-lose-172222-a-second-for-45-minutes)

NYSE. Retail Liquidity Program. Retrieved from https://www.nyse.com/publicdocs/nyse/markets/liquidity-programs/RLP\_Fact\_Sheet.pdf

1. (2012, August 14). Software Testing Lessons Learned From Knight Capital Fiasco | CIO. Retrieved May 17, 2017, from <http://www.cio.com/article/2393212/agile-development/software-testing-lessons-learned-from-knight-capital-fiasco.html> [↑](#footnote-ref-1)
2. (2014, April 17). Knightmare: A DevOps Cautionary Tale – Doug Seven. Retrieved May 17, 2017, from <https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/> [↑](#footnote-ref-2)
3. (2013, October 22). python sweetness — How to lose $172,222 a second for 45 minutes. Retrieved May 17, 2017, from <http://pythonsweetness.tumblr.com/post/64740079543/how-to-lose-172222-a-second-for-45-minutes> [↑](#footnote-ref-3)