The Failures of Knight Capital Trading.

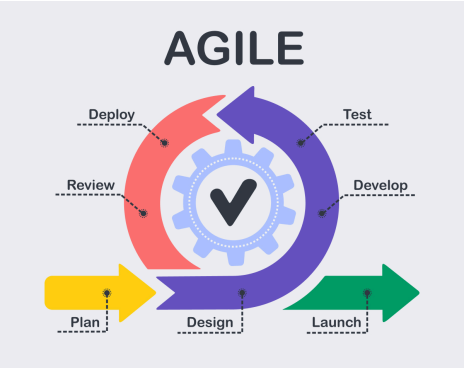
On August 1st, 2012 Knight Capital Trading had a new software running with a bug that they were not aware of. This bug unfortunately cost the company upwards of $440 million dollars and could have possibly been prevented if proper software design principles were in place. In this analysis of this software development failure we will look into the software itself, what went wrong, and how it could've been prevented.

Knight Capital was an investment firm that would trade on the New York Stoke Exchange (NYSE) and had many customers that this failure no doubt affected such as TD Ameritrade, E* Trader, Vanguard and many others. This failure started when the NYSE announced that they would be creating a dark pool of its own called the Retail Liquidity Program (RLP) in October of 2011. This dark pool allows large institutional investors and traders to "now buy or sell large blocks of shares with relative anonymity and without moving the public markets;" (Article). The SEC gave approval for the RPL in early June of 2012 and it was set to go live by August 1st. This meant that Knight's software team had one month to update their software in order to trade on this new dark pool efficiently. This is where the root of the problem stemmed from, because the time frame was so short the developers were in a rush from the beginning. They began undating the software from the algorithms for efficient orders, to the SMARS or the Smart Market Access Routing System. This SMARS software would "receive orders from other upstream components in Knight's trading platform ("Parent" orders) and then, as needed based on the available liquidity and price, sends on more representative ("child") orders" (Article).

So where did this all go wrong? One key failure that occurred in this situation was that the software engineers working on the updates had code in the project that was not being used. This code is eventually what will be the massive problem in this update. The problem with not getting rid of this unused code is that they decided in their rush to update the software that they were going to reuse a flag that was at one point connected to this code. This was the first major mistake, you should always get rid of old unused code such as this as well as not reusing flags or other pieces of code that might still affect other parts of your code. It is important to practice low coupling in code to make sure that when you think you're doing one thing you aren't accidentally affecting some other part of your code. This leads us to our next big mistake that Knight Capital engineers made. When deploying the code to the companies servers the one engineer that manually deployed all the code to the 8 servers accidentally only deployed the code to 7 out of the 8 servers without noticing. Now typically this would be easily prevented by a couple of common practices. One being that more than one engineer works on deploying the code to these servers giving it at least one more set of eyes to see if there is a mistake. Another practice that we see in all areas of business, not just software engineering is having a boss or superior check your work when you are done especially when you are doing a project that is critical in making sure the code works as intended. This mistake along with the reuse of code is what led to the critical failure of this software. To give some context specifically for the reused

code, the flag that they reused in this new update was a reused flag connected to an algorithm they called Power Peg. Power Peg was a script that was set up to buy at high prices and sell when they were low, meant to help the company make sure that the market was reacting properly in test scenarios. If you know anything about stoke trading it's easy to understand how this is the opposite of what you want to do in the real market. So what happened on August 2nd that led to the catastrophic failure of Knight Capital and its software was that when the market opened at 9:00am the 7 servers that were running the new code were working correctly however when the code switch the reused flag to one it inadvertently started the power peg code on the 8th server without the new code on it. This led the server to begin to buy stocks at high prices and sell them low at a very large scale on the NYSE. This is where we run into the next large mistake that I think could have helped mitigate the damage if proper software principles and habits had been implemented. This large mistake is stated in the given source as "The NYSE then alerted Knight's chief information officer, who gathered the firm's top IT people; most trading shops would have flipped a kill switch in their algorithms or would have simply shut down systems. However, Knight had no documented procedures for incident response"(Article). This is a huge problem when building software. We as software engineers are not capable of accounting for all possible variables and bugs that could affect our code, and when dealing with money on this scale we need to account for this by having failed safe procedures for when things inevitably do not go to plan. If Knight had a plan or a kill switch option they might not have decided to revert all the other servers back to the old code making their problem 7x worse because they assumed that the new code was the problem.

I have given some suggestions throughout this analysis while explaining all the faults in the design and deployment however now I will try to give formal suggestions and deliver ideas that could help in the process to avoid these problems in all future software design projects. Firstly I want to suggest that when designing a massive project it is extremely important to use version control. This can allow you and your partners not only to seamlessly work on a project all at the same time, but it also allows you to delete code you are not currently using without the fear that you will be losing it forever. This can be a huge help in development because even if you have a working project and then you add a bunch of new code that causes a huge bug it is easy to get back to the working code you had before. It is a super helpful tool that everybody should use. My next suggestion when creating a large software project is to give yourself enough time to follow a design process like the figure below:



Following a design cycle that works in a circle like this can be a little more time consuming however it ensures that you will get the best chance of finding bugs and problems in your code before you fully launch it. Something like this might not be possible to do well in a month like the team at Knight Capital, however if the engineers were allowed more time on the project they most likely would have had time to test and rework code which would have saved Knight Capital the hundreds of millions they lost. Using a design path like this also ensures that your code that you are putting out is well polished because you have multiple opportunities to check for and fix mistakes that you may have not seen when first writing the code. Using these suggestions may not have fully prevented this horrible mistake however it would have given the engineers a much higher chance of not making these mistakes. In conclusion it is extremely important to follow proper software engineering practices because they are there for a reason.

Resources:

Dolfing, H. (2019, June 5). *Case study 4: The $440 million software error at Knight Capital*. Henrico Dolfing - Independent Board Advisor. https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html