Introduction:

"Devops is a way of thinking and a way of working. It is a framework for sharing stories and enabling people and teams to practise their crafts in effective and lasting ways. It is part of the cultural weave that shapes how we work and why. Many people think about devops as specific tools like Chef or Docker, but tools alone are not devops. What makes tools "devops" is the manner of their use, not fundamental characteristics of the tools themselves."(Davis & Daniels, 2016)

What Is DevOps:

The devops process is best explained by creating a user story:

Pat is a new developer in a company. They arrive on their first day and are provided with a company machine that has all the appropriate accounts and permissions for their position pre-installed. This can include

- Security settings
- Github repositories
- Specific tooling (Git, AWS, VS Code etc)
- Onboarding documentation
- etc.

The machine also has a VM (virtual machine) installed which replicates the production environment. Standardisation of tools is an important part of devops as it ensures consistency across the team and allows the new team member to get up to speed quicker.

Once they have been walked through best practices and have paired with a more experienced employee they will be able to start their work. Since they are working on a localised version of the production they can write and test their code locally, confident that any changes or experiments they carry out will not affect any other members of the team. They can be confident that at this early stage they have written good code by using a suite of local and functional tests.

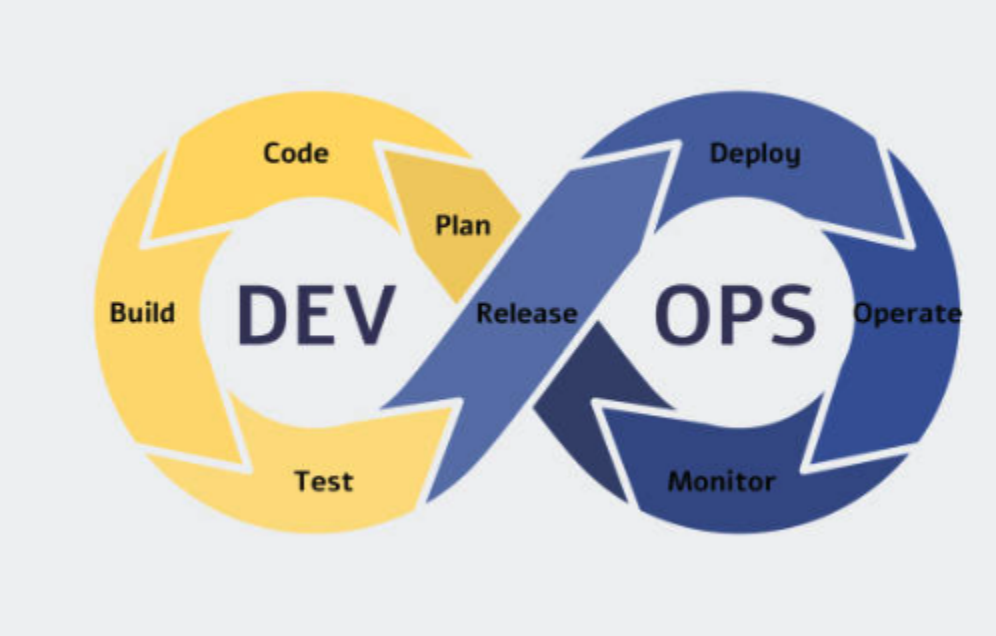
When the time comes to ship the code they can test the code on the development server which should be as close to the production server as possible. At this stage they can (and should) use a pull request to ask a mentor or a more experienced member of the team to review and examine their code so it can be improved upon and they can learn best practices and possibly any particular linting practises the company employs. If the code needs to be changed these changes are integrated into the existing PR. Again these changes do not affect any other part of the platform.

Once the code has been optimised the code is then approved and merged into the codebase. Depending on the complexity of the organisation the code is either shipped immediately or is entered into a ship queue. In both cases the code is again submitted to a battery of tests which ensures it is safe to ship to the production server.

The final steps in this devops process involves observation of the production environment. Again depending on the complexity of the organisation this can just be a visual inspection or it can involve tooling to ensure there has not been any unintended consequences (even though the development is as close as possible to the production environment it is never identical)

While initially this might seem like a large number of steps in what could be a small PR, these steps, when combined and when adopted as a process, allow for a far more streamlined and efficient development process.

Devops has been adopted by many of the largest and most agile technology companies from Netflix to Google to Shopify. It allows companies to push changes to their platform several times a day confidently and allows them to pivot at any time.



(www.istockphoto.com, n.d.)

Tooling:

In each step there are a number of different tooling to allow for automation of processes. Below are a number of different tools we can consider, each has its benefits and drawbacks.

- **Planning/ Project management:** Jira, Asana, Github Projects
- **Version control:** GitHub, GitLab, Bitbucket
- **Pre-launch testing:** Postman, Microsoft unit testing framework
- **CI/CD tooling**: Jenkins, Github Actions, Circle CI
- **Observability**: Splunk, Datadog, Bugsnag.

Our Current Situation

From speaking to Ren it is clear that at this time we have no such processes in place and as a result our codebase has become unwieldy and needs some work to allow us to streamline our processes. This will allow us to

- improve our deployment strategy
- ensure greater platform stability
- allow us to be more agile and react quicker to customer demands

There is also no code ownership which is something we need to develop as with code ownership comes greater engineer investment in the codebase and pride in the platform.

Ren has substantial talent but will be retiring soon taking this knowledge with him. He is onboard with the changes we are proposing here and is also willing to mentor Jalen in the current codebase. Jalen, as a recent graduate, has experience in the more modern practices and is willing to mentor Ren in the adoption of them. This is a great starting point for us as once we have buy-in from the entire team this will allow us to streamline. Both have agreed to act as ambassadors for the project

Modernising Existing Code:

Reasons:

From reviewing the code base I can see we are looking at a monolithic code base containing several business critical capabilities. Because of this it has not been regularly updated and improved, because of the fact that any changes made will affect the entire code base so it is a large undertaking to even update dependencies. This has led to a dangerous situation when an issue with one dependency can take down the entire platform.

An ideal software situation involves code that is tightly bound (all are a part of the company's goals) but loosely coupled (each component can be altered without affecting another component).

Documentation and code ownership are also lacking and we are in danger of losing several domain experts in the next couple of years leaving us exposed if anything happens to the software and we do not have the engineer who understands the issue at hand.

This is not a simple and inexpensive process and is something that will take some time to do and is something that should never be rushed for obvious reasons however it is something that is necessary for the survival of the company.

Challenges:

This will be a complicated project and will present a number of challenges:

- Introducing new procedures to an already volatile workplace environment.
- Maintaining the current code base while creating a new code structure.
- Asking engineers to retire and refactor code they might be invested in.
- Time

Benefits:

The benefits far outweigh the challenges:

- Quicker onboarding processes - an engineer only needs to build an understanding of their area rather than the entire platform so can become productive quicker.
- A modern code base that is more resilient - if an issue occurs with a component it might degrade the platform but will not take it down.
- Better documentation so we are less dependent on tribal knowledge.
- A more agile platform- we can react to customers demands and issues in days instead of weeks.
- The introduction of automation for several processes will allow us to do more with less, this means we will be able to cut our staff costs longterm

Financial reward:

According to research there are huge benefits to be had from modernization. On average, senior IT decision makers believe it has the potential to boost annual revenue by over 14%. Modernisation also has the potential to reduce business operating costs by over 13% . Most (80%) senior IT decision makers believe that **not** modernising IT systems will negatively impact the long-term growth of their organisation. *(www.avanade.com, n.d.)*

Code Modernisation Process

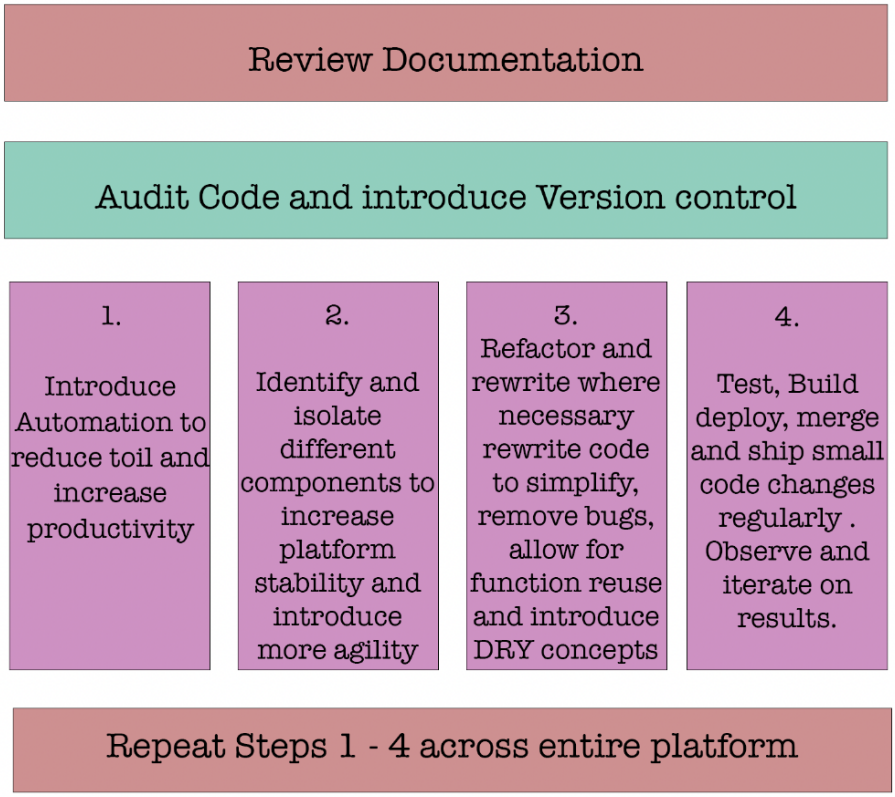
As with all long term projects this is a complicated and in depth procedure but also like all complicated projects it can be broken down into smaller steps that will all stand on their own and can be implemented into the companies workflow

The first steps are to review the existing documentation to understand the code so we can be aware of how altering code in one area will affect other areas and identify any knowledge gaps.

Following this we need to perform an audit of the codebase to identify the different elements that make up the codebase and how they all add to the company's goals. Once these components have been identified we can start to work on them in isolation in a development environment.

This is the beginning of modernising the code and will involve several steps:

- Identifying and isolating common functions and methods to prevent replication
- Identify bugs that exist
- Working on the new code in isolation
    - Refactor where possible
    - Rewrite where needed
- Test in the dev environment
- Introducing a new CI/CD pipeline to test build and deploy the new code
- Merge and ship the changes
- Observe and ensure they are doing what we want to do and we are not seeing any issues
- Repeat



It is important to consider that this code modernisation and refactoring process is not a one and done process. Once we have refactored and restructured it will become an ongoing process where we constantly aim to improve the platform. What is also important to consider is the fact that once we have reduced toil (by introducing automation) and refining scope (by isolating components) this becomes a quicker (and more enjoyable) process for engineers, thus ensuring their buy-in.

Using the real world example:

Using the particular code sample provided we can see how this process can be applied.

Review Documentation:

From reviewing the documentation we can see exactly what this script is designed to do:

> *"This script will run several network tests commands and display an exception if the server is not configured to receive Inbound calls or added as a TrustedHost."*

The review also identifies a number of areas where changes need to be made and possible refactoring that can take place.

Audit Code:

Auditing the code allows us to build up an understanding on where we can explore the mission critical process, where there is repetition and inefficiencies (re entering the same values, repeating tests etc). This script should be edited with a version control too using PRs to work on branches and only once the changes are considered successful merged with main

Automation

When reading the code we can see that there are several steps that need to take place before running these tests which need to be neutered manually, these can be considered toil and can be abstracted out and automated so they run whenever the user logs in.

Refactoring

There are a number of parameters that are reused by the script, these can be declared at the top of the script and then called as needed. There are a number of tests that are running internally, these can be moved to a test script which will allow them to be reused by other scripts as needed and will allow the engineer to focus on .

Test Build etc:

What is important here is that all changes are made incrementally and tested as each changes is shipped. Once the changes are made they should be reviewed, tested, shipped and observed before making any other changes. Any PR that introduces issues should be rolled back and reviewed.

Conclusion:

Shinty software at its core has a strong business model however it has some major challenges ahead of it . It needs to streamline, modernise and automate its processes so it can move quicker and become a more agile company. There are a number of steps that we need to take to allow us to achieve this goal

Modernise the entire software delivery system:

By taking advantage of a software delivery and version control system like GitHub we can allow our engineers to collaborate better, ship code quicker and become more confident and efficient in their work . Github also has a number of extra features including workflow automation, project management and CI/CD automation and testing out of the box so this is a simple step to allow us to work better and faster.

We are using several different languages across the platform. While this might be necessary in some cases, it is a lot more efficient if we define and work in one core language. This will also allow us to target the hiring market for developers who have experience in this language and will allow for more collaboration. We need to speak to our more experienced developers to ask then for their opinion on this as well as carry out research on our competitors to see how they work

Modernising the code base:

We are currently using a monolithic code base so a change to one part of the system can in theory affect another unrelated part of the system and this has hampered modernisation. We need to audit the entire code base and isolate the different components that make up the system and refactor them one at a time in small increments. These changes need to be tested, and observed and once they are proven to be successful we can move to the next step. This might seem like a laborious process but as I have said once toil is removed from the platform (using automation tooling) it will be a lot quicker than expected.

TimeFrame:

As you can imagine this is a laborious process and will take some time to complete. One advantage in working on different components is the fact that change is visible from the outset and each change brings tangible improvements. This is an important part of getting buy in which is why I have asked Jaden and Ren to begin the process on the script provided. This will then be presented to the rest of the team to ensure buy in. Until the code audit is completed there is no way to have an accurate estimate of timeframe but I would consider 3 months a good estimate to have the core areas of the platform updated. As I have mentioned this will then be an ongoing process but will become simpler over time and will allow us to integrate changes quicker.

Assumptions:

We are going to assume several things before starting this report

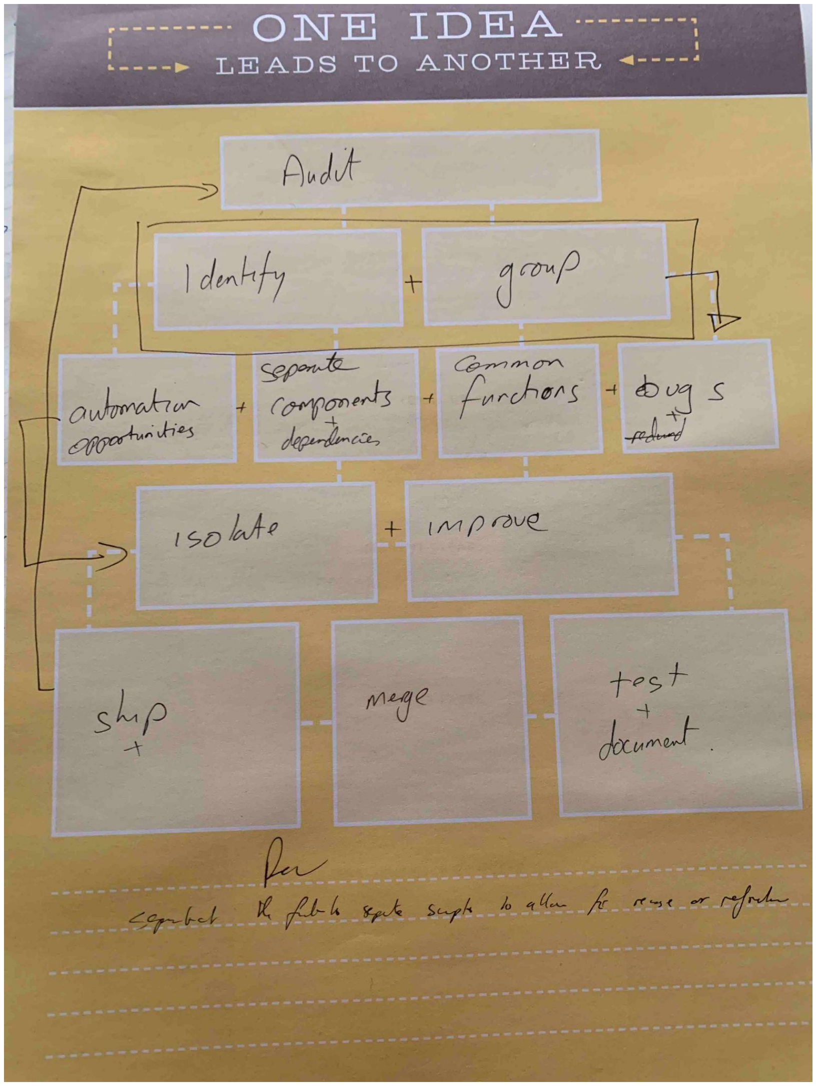
- As the company is an older company we are going to assume they have not employed any devops principles
- We are going to assume the new manager is onboard with change
- We will assume they are using an out of date monolithic code base
- We will assume that the intern has knowledge of current and best practices
- We will assume that the older dev is knowledgeable about the company, the code and its previous procedures and is willing to share this knowledge before he retires
- While the company is in trouble it is not in imminent danger of going under
-

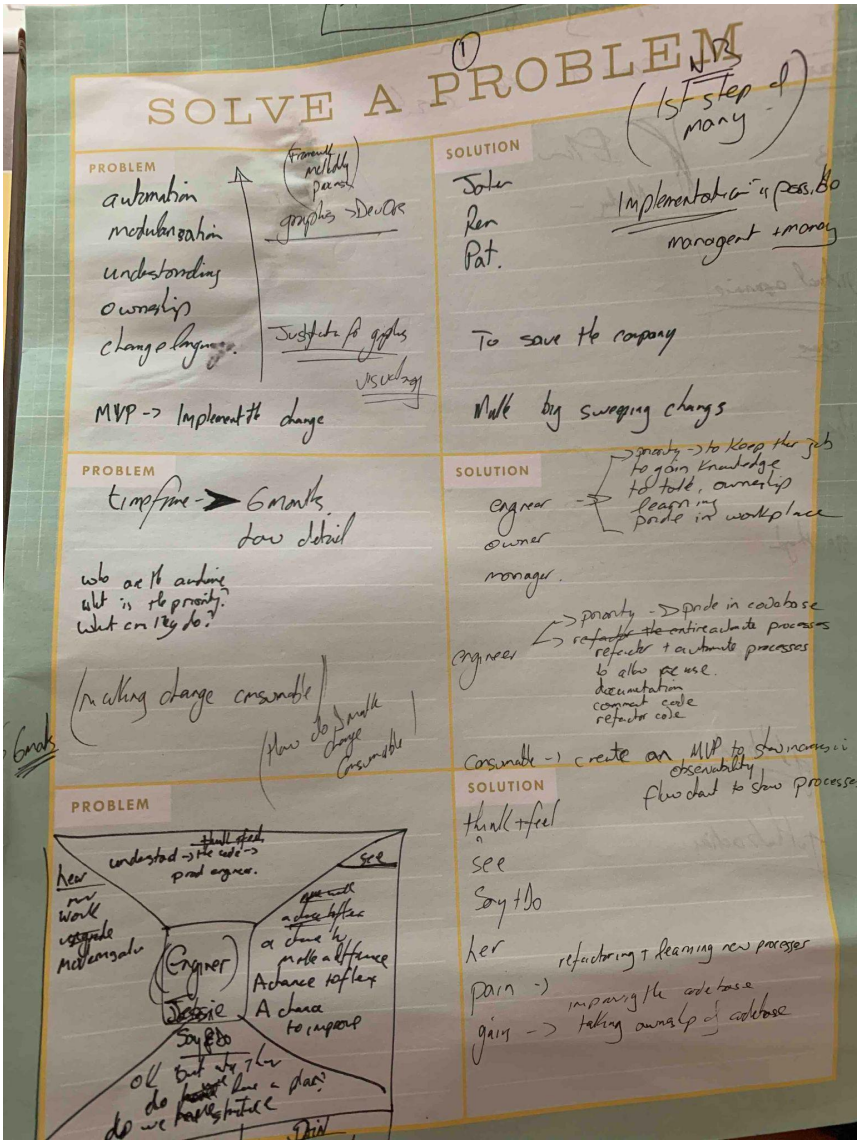
References

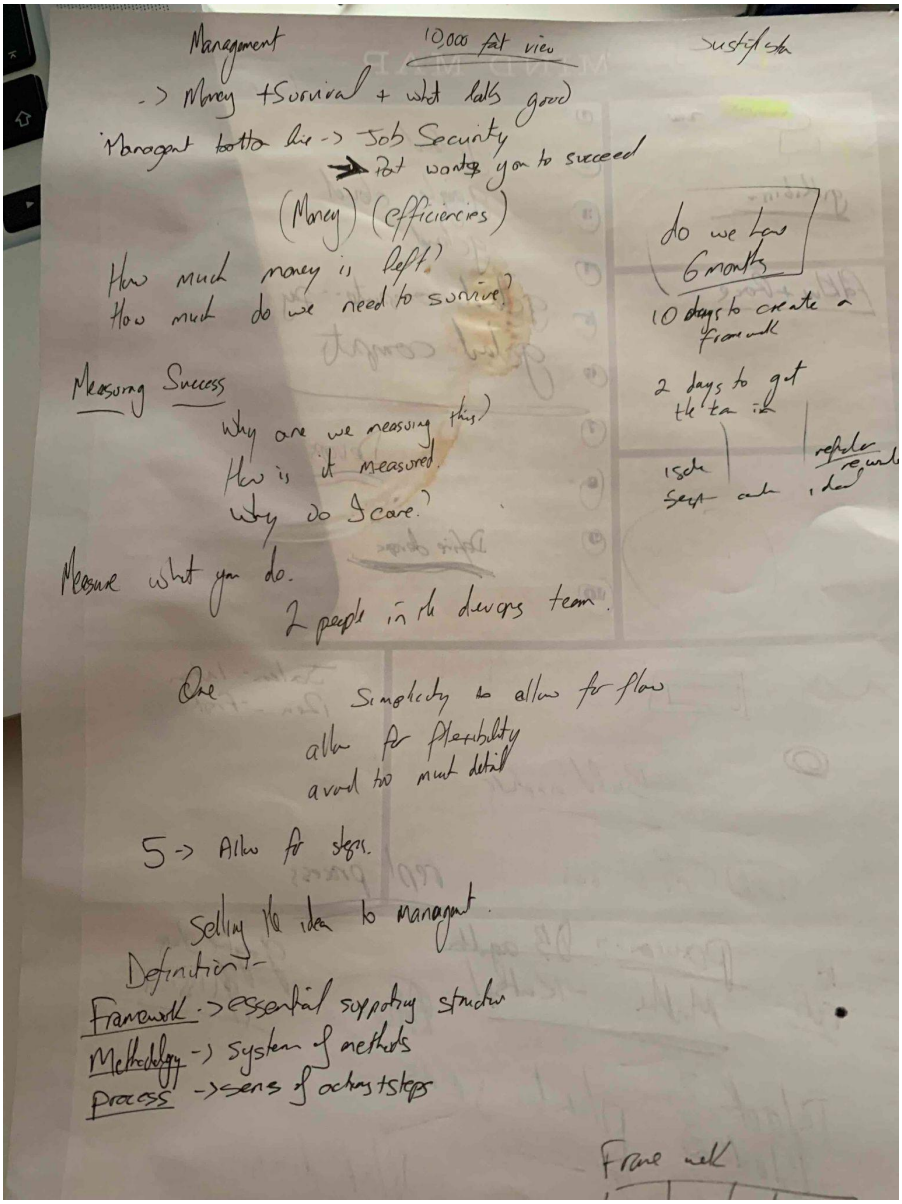
- *Davis, J. and Daniels, R., 2016. Effective DevOps. 1st ed. USA: O'Reilly Media Inc.*
- *www.istockphoto.com. (n.d.). Devops Stock Photos, Pictures & Royalty-Free Images - iStock. [online] Available at: https://www.istockphoto.com/photos/devops [Accessed 24 Nov. 2022].*
- *www.avanade.com. (n.d.). IT Modernisation Press Release | Avanade UK. [online] Available at: https://www.avanade.com/en-gb/media-center/press-releases/it-modernization-research [Accessed 26 Nov. 2022].*
- *www.codurance.com. (n.d.). Breaking Down the Monolith | Codurance. [online] Available at: https://www.codurance.com/publications/2020/09/08/breaking-down-the-monolith#:~:text=Instead%2C%20work%20towards%20having%20a [Accessed 26 Nov. 2022].*

Appendix:

- Link to Github Repo


ONE IDEA
LEADS TO ANOTHER
Audit
Identify
+
group
automation opportunities
+
separate components + dependencies
+
common functions
+
bugs
isolate
+
improve
ship
+
merge
test
+
document


SOLVE A PROBLEM
(1st step of many)
PROBLEM
automation
modularization
understanding
ownership
change of engineers
MVP -> Implement the change
SOLUTION
Jake
Ren
Pat.
To save the company
Make big sweeping changes
PROBLEM
timeframe -> 6 months
SOLUTION
engineer -> priority -> to keep the job, to gain knowledge, to build ownership, learning, pride in workplace
owner
manager.
engineer -> priority -> pride in codebase, refactor + automate processes, documentation, connect code, refactor code
Consumable -> create an MVP to show managers
observability
flowchart to show processes
PROBLEM
SOLUTION
think + feel
see
Say + do
Pain -> refactoring + learning new processes
gain -> improving the codebase, taking ownership of codebase
Engineer


Management
10,000 feet view
-> Money + Survival + what looks good
Managment looks like -> Job Security
-> Not wants you to succeed
(Money) (efficiencies)
How much money is left?
How much do we need to survive?
do we have 6 months
10 days to create a framework
2 days to get the data in
Measuring Success
Why are we measuring this?
How is it measured
Why do I care?
Measure what you do.
2 people in the devops team.
One
Simplicity to allow for flow
allow for flexibility
avoid too much detail
5 -> Allow for steps.
Selling the idea to managment.
Definitions-
Framework -> essential supporting structure
Methodology -> system of methods
Process -> series of actions/steps
Frame work