Leadership Document

This document is intended to outline my leadership style and principles, in the context of leading a team of software engineers in an Agile environment

Processes

Assessing and improving a team's technical practises

Key to making effective, long-lasting changes to any team's ways of working is full consultation from the start. Speaking individually, and as a team, to gather anecdotal evidence of common pain points and known areas of frustration/inefficiency is a good way to identify starting point(s) – and ensure buy-in/trust.

If historic data on the team's metrics (velocity, cycle time etc.) is already available, that can accelerate the improvement process, and in itself provide an indication of which areas should be addressed first.

Once the key areas for improvement have been identified, sensible measurable goals can be set and the progress can be tracked against the above mentioned metrics. It's important to emphasise that this is a team effort, and not a critical focus on any individual.

Balancing 40-50% hand-on technical work with leadership responsibilities

The practical way of ensuring this balance would be to measure the average number of story points completed by the team per sprint and make my typical commitment half of that value. Realistically, depending on the flow of work at that time, my hands-on commitment may increase or decrease, so flexibility is crucial. I love both sides of the role equally, so doing more or less of either is no inconvenience - and in cases where I've not been coding for a while, I can always shake off that rust with one of my many personal projects in my spare time. My first duty is to my teammates, so I am always happy to put a coding task on hold momentarily if I'm needed to provide assistance or advice.

Specific processes to implement to improve quality and delivery speed

Identification - and automation - of any repetitive, laborious processes is always one of my top priorities. This increases the speed of delivery, the quality of the developer experience, and massively reduces human error (which in turn leads to an increase in quality).

The quality and coverage of the testing is also vital. If we have comprehensive, stable and trustworthy automated testing, this allows us to operate a much quicker CI/CD pipeline, where releases can happen little and often - and our customers get to reap the benefits of continual incremental improvements.

The time it takes to implement even the smallest change - from saving the project, to building/running it and observing the results of any change - is a critical metric for me. If it takes a long time to observe any change (sometimes up to *45 minutes* in the worst situations I've seen) then that limits the number of iterations an engineer can make on any given day. This time must be reduced to an absolute minimum, through a combination of good hardware, reliable networks, and well architected systems.

How to foster a "build it, run it, own it" culture

Leading by example is very important in this scenario. If I am asking engineers to maintain a level of responsibility for the continual operation of their applications - including out-of-hours support - then I have to demonstrate that same willingness to do so.

But it is imperative that the engineers are equipped with the proper tools to carry out these responsibilities. The monitoring and alerting tools must be fit-for-purpose, and the procedures for any contingency must be clearly defined and logical to follow.

On a personal level, if you create the right atmosphere amongst the team – one of openness and confidence, where people can take pride in their work and pride in being part of the team – then a level of ownership of the code and infrastructure starts to grow organically.

Coaching engineers with different experience levels

Whether an engineer is a 20-year veteran or a fresh-out-of-University graduate, the most important element in any kind of coaching relationship is respect. This can manifest in different ways, eg. With an experienced engineer, I will often ask questions from the point of view of seeking to improve my own capabilities, prompting them to revisit and bolster their own existing knowledge, but this will often be a targeted question that is relevant to their current problem – an attempt to steer them towards success in a way that is much more a journey of “self discovery” than targeted instruction.

A similar approach can be adopted with less experienced colleagues, but you do have to be more conscious about recognising the right time to actually intervene with a suggested solution.

The key point in all of this is that “the best solution” is what we are after, in all situations, regardless of the seniority of the person who proposes it. Learning how to get the best out of each unique individual is the real aim.

Strategy for implementing infrastructure-as-code practices within the team

As is probably clear in my code implementation, this is an area in which I would like to increase my knowledge - so that I can define architectures in a totally self-sufficient manner, as opposed to having my hand held by a dedicated DevOps engineer.

With that said, part one of this strategy would be to pair or mob with an experience DevOps engineer, so the team and I could be coached in best practices and learn of any common pitfalls etc. ahead of time.

Once the team is up and running with these practices, we would continue to build on our skills and experience with code reviews of relevant YAML files, regular demonstrations of full teardown/redeploy procedure, and occasional knowledge sharing sessions whenever anyone in the team discovers something new and useful.

Relevance to my technical implementation

First and foremost, my approach to the technical implementation was to absolutely minimise the time between making a code change and observing its results. I implemented the ability to stop and restart the applications in a matter of seconds, allowing rapid development. This is relevant in terms of reducing cycle time – of the ticket/task in

general, as well as the individual dev cycle.

I provided comprehensive automated testing, to give the engineers confidence when making changes, and also make life easier for the testers to approve any changes for release – increasing our speed of delivery.

With regards to “build it, run it, own it”, my implementation gives the engineer full control over the entire architecture. Nothing is obscured from them, and they may change any aspect of it. Fully stopping or starting all applications is very quick and simple, reducing the complexity of any out-of-hours support.

Once this initial architecture is presented to the team, engineers will be able to work on any of the applications individually, meaning that the more experienced developers can make improvements to the quite generic placeholder code that is there, and the newer developers can look at each service and start to build an understanding of distributed systems.