# Improving Developer Velocity - a Case Study

## Introduction

In a 2020 paper published by McKinsey titled *Developer Velocity: How software excellence fuels business performance*, it stated that companies with top "Developer Velocity Index (DVI)" outperform the market with 4-5 times revenue grow and 55% higher innovation ("Measured by level of adoption of new technologies and ability to innovate faster and beat competition through innovation led growth"). To learn more about Developer Velocity, you can download a copy of the white paper at this Azure page.

From the McKinsey paper, Developer Velocity is defined as the ability to drive transformative business performance through software development. Empowering developers, creating the right environment to innovate, and removing points of friction can all contribute to a high Developer Velocity.

The goal of this white paper is highlight some best practices that could help an organization to achieve these high-level ideal organizational states. The best practices in this paper is more focused on communication and collaboration best practices in an Software Engineering Team. We will use the engagement between Microsoft and XP Investments as a case study and demonstrate on how we used these best practices to improve developer velocity.

## Engagement Background and Challenges

XP is a financial services company located in Brazil. According to their "about us" page, their purpose is to "transform the financial market to improve people's lives."

Microsoft is a global software company whose mission is to "empower every person and organization on the planet to achieve more."

In Spring 2021, XP and Microsoft started an engagement with engineers from both companies to co-develop the foundation of the next generation of XP's data platform.

The goal of this data platform is to provide a centralized platform that has:

1. Data Catalog
2. Monitoring / Observability
3. Alerts
4. Logging
5. Discoverability
6. Workflow Management
7. Self-Service

Not only are the engineers from the two companies never worked with one another before this engagement, the teams within the respective companies are also relatively new. This presented challenges are how to foster a culture of collaboration and healthy communication. The team was also geo-distributed. Engineers from Microsoft were primarily located various parts in the US and XP engineers were located in Brazil. The team needed to adapt and effectively collaborate across different timezones.

This project was limited to 8 weeks. This meant the team needed to design, scope and execute within that timeframe. Given this timeframe, the scope needed to be reasonable but still needed to implement a demonstrable product that has a high ROI in terms of impact to future development.

## Core Principles

Here are the core principles that are at the root of the best practices utilized in this engagement:

- **Build Trust**: Trust is at the center of how we worked with one another. There are assumptions about trust that is central in our day-to-day work: Trust that developers will raise any blockers or issues that arise. Trust that leads will be transparent about risks and manage external distractions. Trust that colleagues have the project's and other colleagues' best interest at heart. Some of the best practices outlined fosters trust in one another.
- **Provide Context over Control**: One of the core tenets on Netflix's Culture Page, Context over Control is about providing the team with as much context as possible so that the team can achieve autonomy and self-direction.
- **Aim for Clarity over Consensus**: It is better to have clarity on the situation or challenge, and understand the tradeoffs between options than to strive for agreements amongst the team without having clarity on the whole picture.
- **Validate Early and Fail Fast**: An invalidated hypotheses is as important as a validated one. It's advantageous to prioritize the risks in the early part of the project; this allows more time to pivot.
- **Strive for Depth**: Understanding the foundations is essential for a team to stay flexible and agile. It allows us to make better decisions. And understanding the motivations behind the tools, languages, and frameworks we use allows us to choose the right ones for the job.

## Best Practices

We used the following best practices during the 8 week engagement. In combination, these best practices created a highly-functional project team. Despite time zone differences and language barriers, the team accomplished what was scoped and maintained high morale and strong sense of team.

### Envision Together

Forming a vision together allows the everyone on the team to have a shared ownership of the end-product. However collectively shaping a vision is often a messy and long process. XP used a methodology called Lean Inception to provide a structure for this process. Lean Inception facilitates the process by providing discussion goals and milestones. The result of this process is a shared vision of a solution that everyone is excited about.

### Architecture Design Sessions (ADS) and Records (ADR)

Architecture Design Sessions (ADS) are designed to guide the participants through a series of design discussions. During the design sessions, participants can break into smaller groups to answer certain architecture questions and conduct investigations on the viability of certain ideas. These groups can then make recommendations on certain approaches. At the end of the ADS, the result is a detailed architecture diagram that specifies the initial design of the architecture. Accompanying the architecture diagram should be a collection of Architecture Decision Records(ADRs).

Architecture Design Records are to capture the particular challenge, and the reasons why a particular decision was reached. It should also include other viable options considered as well as the impact and tradeoff of the decision. ADRs are designed to be iterable, meaning that designs and decisions can be changed if situations or new findings appear. This is done by adding to the collection of ADRs and reflect the change. You can find an example of an ADR at the end of this paper.

Because Architecture Design Records is a community-backed initiative, there is good community support on its tooling. Markdown ADR (madr) provides a definitive template for the document. There are even tools such as Log4brains that allows for statically generated websites built on madr.

## Informed Captains and Decision Making

Informed Captains is another core idea from Netflix's Culture Page. An informed captain's job is to make a particular decision. This eliminates "design by committee". This person is empowered to gather all facts and opinions about a subject that requires a decision. The important element of gathering opinions is to emphasize on gathering as much differing opinions as possible; this encourages people who might initially be hesitant to voice an opinion that is different than the majority opinion. We assigned different informed captains for different decisions and rotated the feeling of responsibility.

For most decisions that requires a designated Informed Captain, the decision making process is as follows:

1. Problem Statement - During a standup, the Informed Captain presents the decision point that needs to be made. (I.e. Given our architecture, what message broker should we use?). The problem statement is then posted on the General Teams channel for record keeping and to maximize visibility.
2. Gathering Phase - This can be a meeting and/or a Teams thread. The Informed Captain gathers as much opinion and facts as possible.
3. Proposal Phase - Informed Captain makes a proposal to the team on Teams, outlining the decision as well as the reason why the direction was chosen. The team is able to respond to the proposal by having dissenting opinions by highlighting the disadvantage to the approach.
4. Finalize Phase - Informed Captain makes the final decision by taking into account the dissenting opinions, then communicate the decision on Teams.
5. Documentation Phase - Informed Captain documents the decision in the form of an Architecture Decision Record.

## Working Agreements

The goal of an working agreement is to provide absolute clarity on how the team will work together. For this engagement, the working agreement comprised of 5 main sections:

- Collaboration
- Communication Norms
- Code management
- Definition of Done
- Norms for reviewing and submitting pull requests

Often these norms or practices are not written down, by being explicit, the agreement eliminates assumptions from engineers and stakeholders and forms healthy expectations from the rest of the team.

As an example, communication norms contained items like:

- WHEN core working hours for the project - our team spread across 4 time zones
- WHERE our communications happen (ie. Teams - not email, channels - not chat)
- WHAT our communication etiquette were (ie. prefer async, meetings should begin/end on time, should contain an agenda)
- HOW User stories should be composed (ie. feature-centric, can be implemented with code or demonstrable)

You can find an example of a working agreement at the end of this paper.

## Code-With

This engagement was a collaboration between Microsoft and XP. The notion of "Code-With" was at the center of this engagement, and empowered the engineers from both companies to engage fully. Code-With motion emphasizes that development expertise flow in both direction, back and forth between the two companies. This notion asks engineers to pair program, and be comfortable to present the incremental work they done on a daily or weekly basis. It is counter-intuitive to think that taking time to teach or having two engineers work on one task would increase Developer Velocity, but the positive effect is immediate.

## Workstreams

Creating XP's next-generation data platform is no small feat. Given the large of engineers we had on the project, we decided to have multiple workstreams on this engagement. The goal of the workstreams is to provide smaller scopes for engineers to have a focus within a sprint.

To serve as an example, we identified the following workstreams for this engagement:

- DevOps
- API
- Transformation
- Streaming

With the workstreams identified, we ask the engineers from both companies to self-organize with the following guideline:

- Each workstream needs a Workstream Lead. Ideally this is someone who has enough knowledge or context.
- Each workstream needs to have at least one engineer from both companies.
- All workstreams should be roughly the same size (number of engineers).

Self organization gives the engineers the autonomy to work with the technology they want.Engineers were not restricted to a single workstream and had the freedom to move between workstreams between sprints. Some people preferred to stay in a workstream for the duration and that was also acceptable.

## MonoRepo

There is fierce debate within the development community around MonoRepos. We found MonoRepos helpful for projects as it helps with collaboration, and enables team members to jump between different workstreams. The design around MonoRepos requires more effort as the project can grow quickly and become unwieldy. We were intentional on the file structure to ensure that there is minimal effort for a team to break off of the

monorepo and create a separate repo. We achieved this by encapsulate all aspects of a particular application into its own directory, shown below:

```
/(project)
|  /docs
|  /infrastructure
|  |  /pipelines
|  |  /iac
|  /apps
|  |  /app-a
|  |  |  /cicd
|  |  |  /docs
|  |  |  /tests
|  |  |  /src
|  |  /app-b
|  |  |  /cicd
|  |  |  /docs
|  |  |  /tests
|  |  |  /src
```

## PRolice

PR reviews is usually an ambiguous process. Situations like these often come up:

1. Pile ups - sudden large backlog of PRs waiting to be reviewed
2. Stuck PRs - engineers disagreeing and gridlocking PRs
3. Large PRs - PRs that try to implement multiple tasks in one PR and touch a large number of files

We solved first two situations with a role call the "PR Police" or "PRolice", and the third is mitigated with the "Code management" section of the working agreement.

The PRolice is a person (rotated by Sprint) that monitors the PRs and individually notifies people to review them. If a PR isn't approved in a day or more, the PRolice will investigate the reason. If a PR is stuck, the PRolice facilitates the conversation and strives for a resolution. If there is a pile up of PRs, PRolice actively calls out reviewers to review these PRs. After implementing the PRolice (and small tasks), we didn't have any PRs that spanned across the day boundary.

> Note: The PRolice is a person. We find automated notifications are impersonal and often get ignored by engineers. Also important to note that this role should be rotated regularly.

Benefits of having a PRolice are:

1. Higher morale - as every one is actively reviewing other's PRs instead of just one or two people who are used to reviewing PRs
2. Higher velocity - as PRs are merged more efficiently
3. No one gets stuck in the "no one cares about my PRs" mentality

## End of Sprint Demos

Because this engagement had a relatively short execution window, our sprints were one-week sprints. End-of-sprint demos are to showcase our work to one another and to the main stakeholders. The demo sessions also are all recorded to maximize the audience. The goal of the end of sprint demos is ultimately to celebrate the work that the team accomplished that sprint, and also to gather feedback and share ownership with the stackholders.

## Conclusion

The goal of this white paper is to outline and highlight some best practices that both fosters a culture of collaboration and improves an organization's Developer Velocity. Of course, every team is different and every company is facing different challenges. We hope this paper provides some useful practice that you find helpful for your organization.

## References


- "Developer Velocity" by McKinsey
- Work Agreement Template
- ADR Example
- CONTRIBUTING guidelines
- Architecture Design Sessions
- Lightweight Architecture Decision Records