OverOps

Whitepaper

5 Steps That Helped Me Turn a $15M Loss Into Successful Software Delivery

Pierre Bouchard, Senior IT Executive in the financial sector shares his unique perspective on how to drive innovation, motivate engineers and promote accountability throughout the organization.

Pierre Bouchard has more than 20 years of experience applying effective engineering solutions for the most complex problems at companies including JP Morgan Chase, BNY Mellon and Wells Fargo.

As part of our innovator spotlight series, we asked him to share his unique perspective on how to drive innovation, motivate engineers and promote accountability throughout the organization.

By OverOps, Inc. 2020

Moving Quickly vs. Moving Too Quickly

It's no secret that **if your competitor gets their product out before you, you lose market share**. This creates immense pressure to accelerate development and delivery of new, exciting features.

Agile methodologies were created to enable fast-moving engineering groups to meet demanding deadlines. However, there is a fine line between moving very quickly and moving too quickly, which makes us vulnerable.

In this case, Agile Methodology changes to what I like to call Fragile Methodology. You may manage to get your product out before the competition, but if (e.g. when) it fails and harms customer experience, **you will still lose market share**.

We have to find a middle ground where we release innovative features that work well. **Speed is important, but quality is fundamental**. This is especially true for large financial services and banks (the industry I work in), as each mistake can end up costing hundreds of thousands, if not millions, of dollars – but it really is true for anyone working on software.

In this article, I'll share the 5 steps that helped me turn a $15M "lost cause" into a successful project – by maintaining the balance between shipping code quickly and ensuring application reliability and observability.

5 Essential Steps to Avoid the Fragile Methodology

1. Create a Mentality of Code Hygiene

In order to avoid falling into the trap of fragile methodologies, *we need to create a mentality of code hygiene*.

Code quality must start at the very beginning of product development, as early as the prototype stage. According to a report from the Consortium for IT Software Quality, developers introduce an average of 100 to 150 errors for every thousand lines of code they deploy.

It's easy to form bad habits when trying to keep up with high frequency releases. From beginning to end, all members should be trained to build and deploy code with quality in mind or there is a high risk that every change will create heavy technical debt.

2. Introduce Radical Transparency & Open Communication

The bottom line is that if you can't run something clean from the get-go and you don't deal with it immediately, you are creating technical debt that will spread like cancer.

The first part of maintaining code hygiene has to be creating awareness about the effects of code changes. Because any change, no matter how small, opens us up to risk of errors, it's important to promote radical transparency within the organization.

Next, it's important to have those brutally honest conversations and to deal with issues early, before they develop into something that is too large to manage.

Each day, before work begins on new tasks, the results of the previous night's build should be reviewed. Metrics to consider include newly introduced errors,

previously resolved errors that have resurfaced and and performance issues. These should be clearly communicated across the entire team.

3. Gamify Issue Identification and Resolution

When code changes are frequently reviewed and the right information is made available, identifying and resolving issues becomes more manageable.

In some cases, gamifying the process can be successful in encouraging developers to more quickly burn down errors and/or reduce technical debt. The saying that you can take a horse to water, but you can't make them drink is also true for developers. Gamification of this process is beneficial for the overall code quality of the application(s), and also challenges developers to continually improve their personal abilities.

4. Verify Build Reliability Between Each Development Phase

Agile methodologies, as previously mentioned, enable us to move faster and CI/CD frameworks work to support that from a practical standpoint. To avoid customer-impacting issues, quality gates are defined and enabled across the SDLC to prevent unverified code from passing to the next stage of development or to production.

Quality gates can be configured in CI/CD tools such as Jenkins, TeamCity, etc. and automatically block code with issues that developers or QA teams miss.

5. Share, Train and Expose Everyone to Best Practices

Without consistency and shared learning, these practices are not sustainable long-term. In order to see continual improvement, ensure that everyone on the team is aware of best practices and knows how to use the available tool stack to its fullest potential.

It's helpful to *create centralized documentation and/or videos for training purposes* to facilitate on-boarding for new team members and to keep everyone up to date with current processes.

How I Used These 5 Steps to Turn a $15M Loss Into a Successful Project

In one of my previous roles, I inherited a project that had already cost the bank I was working for $15M. The first thing I did was to introduce TDD.

Test-driven development (TDD) refers to a style of programming in which three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring). It essentially enables you to write a "single" unit test describing an entire aspect of the program.

Next, I initiated a culture of radical transparency by showing results every day of the previous night's build. To do so, it's crucial to have a granular view of the application's behavior. How many new errors did the build introduce? Did previously resolved issues resurface with the newly introduced code? Are there any critical errors that put the system at high risk for failure?

Once the data is available, the hyper transparency model can be gamified to encourage developers to burn down errors and technical debt. Human egos can be challenging, of course, and the hyper transparency model makes it painfully obvious when one developer's code quality is better or worse than another's. By gamifying the process, developers are challenged to continuously improve their skills.

With the manual parts of the process optimized, we had to consider the places where automation was needed. It was important for us to shift left, that's one of the main things that led us to develop the hyper transparency model for development. So, we also added automated quality gates to create an additional

layer of verification before code changes passed through each stage of development.

By focusing on increasing transparency across the organization, the team was able to progress by leaps and bounds. Within approximately 6 months, 80% of technical debt was taken care of and the project was back on track.

Conclusion

Producing quality software at the pace expected of us isn't easy, but with the right focus it is doable. First, create a mentality of code hygiene from the get-go and promote radical transparency across all levels of the company from junior staffers to executives. Together, these two aspects form the foundation for high-quality software development.

Beyond that, focus on simplifying identification and resolution of errors and on automating as many processes as possible throughout the SDLC.

We know that if our competitor gets their product out before us, we lose market share. And we know that if we get our product out before the competition and it craps out, we still lose market share. Now, the focus is finding a middle ground where we release innovative features that work well.

Remember: Speed is important, but quality is fundamental.