HOW TO RESOLVE TECHNICAL DEBT: THE AGILE ROOT OF YOUR PROBLEM

We're seeing technical debt occur in mainframe development as more teams move away from Waterfall to Agile. However, regardless of what development philosophy your mainframe team follows, there are a few common flaws that cause technical debt to accrue.

When people talk about technical debt, they're really referring to the long-term consequences of "quick and dirty" program changes made to meet deadlines instead of coding "the better design," as software development author and speaker Martin Fowler puts it.

How IT organizations accrue technical debt is like how people accrue financial debt, Fowler says. It "incurs interest payments, which come in the form of the extra effort that we have to do in future development because of the quick and dirty design choice."

This is especially true of teams doing Agile Development. When you're agile, the pace is faster and you're delivering functionality customers need as soon as possible. Things just need to work so you can keep going, and although they need to work well, "quick and dirty" work still happens and technical debt accumulates.

We're seeing this occur in mainframe development, as more teams move away from Waterfall to Agile. However, regardless of what development philosophy your mainframe team follows, I think there are a few common flaws that cause technical debt to accrue:

- **Poorly defined Dev and Ops practices**, including technical shortcuts made to meet project

deadlines
- **Lack of documentation** for code logic, impairing a developer's understanding of a program to find code to change
- **Inadequate development tools** for source code management, debugging, coding standards and enforcement, code coverage and testing
- **Zero or ineffective automated testing**, leaving room for mistakes to be made or testing to be neglected due to manual processes

It's good to know what activities, or lack thereof, are causing technical debt. But when your goal is to pay it off, you also have to consider where your technical debt exists.

Where Technical Debt Exists

Depending on the culture, processes and tools surrounding your mainframe shop, where you're seeing technical debt accrue most will vary, especially for those moving from Waterfall to Agile. There are at least five common areas where technical debt arises for mainframe teams on that journey.

1. Inheritance from Waterfall

Waterfall development gives you time to thoroughly plan, research, design, code and test and fix bugs before deploying your new functionality. The downside is you aren't delivering fast enough to satisfy customer, business and marketing demands.

Unlike Waterfall, Agile Development can be used to create smaller pieces of targeted functionality faster to meet business pressures sooner with new features. This is good for companies and their customers in a digital age. However, if you're moving mature applications from Waterfall to Agile, technical debt is often inherited from old systems and old code.

From Old Systems

Any system that has been around a while most likely has its share of technical debt. The older the system, the more developers—some better coders than others—likely worked on it. Code is continuously updated and changed, and older code bases grow over time, making it all harder to maintain.

From Old Code

The more code added to existing applications, the greater chance of introducing new complexities or breaking existing logic, resulting in more defects and increased development time. Older bugs are usually more difficult to find and fix.

And even if products are vetted for high performance before release, technical debt is inevitable when you're moving fast. It builds up quickly due to lack of time available to refactor code, build a test automation environment or even write good documentation.

2. Insufficient Documentation

Documentation-based technical debt is really its own beast. It's easy to accrue and even easier to do

a lousy job with, but it's often overlooked.

With Agile, systems change much more frequently than Waterfall. This means documentation needs to evolve quickly, too.

There is a tendency for developers to document changes without regard to the overall flow of customer-facing product documentation. If proper care isn't given to how documentation is added to a guide, you will quickly end up with a bloated, unusable set of instructions.

When you start transitioning mature Waterfall applications into Agile processes, technical debt in existing documentation is usually evident—instructions are left out or in, steps are missed, things aren't conveyed clearly and new updates exacerbate the problem.

However, for new applications developed within Agile processes, it's easy to build great documentation because you're starting from scratch—that is, if you strictly adhere to the definition of done.

3. The Definition of Done

It's difficult for technical debt to accrue from new development if the definition of done is strictly followed. What that means is:

- Development work is completed as defined in a story and satisfies acceptance criteria.
- Manual and/or automated test cases are added and executed as required.
- Any applicable documentation is completed.
- There are no defects in the new code associated with the story.
- The product owner has accepted the changes.

In Agile, the product owner is responsible for declaring a story done, but the whole team is responsible for enforcing the definition of done. When the definition of done is not adhered to, technical debt can result as:

- Inefficient code or code that doesn't satisfy business objectives.
- Insufficient testing, which manifests in future development.
- Unsatisfactory technical or customer documentation.
- Buggy code that will require future maintenance work.

4. Taking on Too Much Work

Technical debt can also result if your mainframe team takes on too much work. This is especially true for Agile teams working in sprints. There are several areas where this can happen.

Sprint Planning

In sprint planning, the team should be confident they can complete all stories planned for that sprint, otherwise, they may rush and take shortcuts to complete the work, which will result in inefficient code, defects and poor quality. Stories should be written with specific acceptance criteria, refined and sized before a team commits to including them in a planned sprint.

Timebox Agile

Many teams have a hard time saying, "No," and take on more during a sprint than can possibly be completed. This is especially true within timebox Agile, where teams "sprint" toward delivering a minimum viable product (MVP) by the project's targeted completion date. With timebox Agile, an MVP should always be defined, sized and agreed upon upfront so the team doesn't get caught continuously chasing an ever-expanding project scope.

Work-in-progress

Each team member should work on one story at a time to minimize work-in-progress (WIP). Too much WIP may result in technical debt because people jump from one task to another instead of focusing on completing the highest-priority work before taking on more.

Scope Creep

Teams should not feel pressured to take on more work than they believe they can complete during a sprint. Scope creep, adding work to a fixed-length sprint or time-boxed project without adding resources, often results in technical debt. The team needs to speak up and push back and know their capacity.

5. Being Pulled Away from Sprint Work

In Agile, scrum teams should be dedicated to sprint work. Interruptions such as customer issues and application downtime negatively impact and often result in incomplete sprint work or developers working overtime.

Developers should be focused on their committed sprint work and not worried about being pulled off to handle customer and production issues—that's what customer-solutions teams are for: ensuring customers receive the attention they deserve while developers produce the functionality customers require.

Of course, always focusing on new development can cause technical debt. Time and resources should be allocated to properly refactor and maintain code, improve automated testing and update existing documentation. This doesn't add new functionality but will keep the team from slowing down and make them more efficient in the long run.

Technology has advanced and provided more efficient and effective hardware, software and network functionality that may require coding changes to take advantage of. The easier it is to read and understand the code, the easier it is to change and modify.

How to Pay It Off

Technical debt obviously isn't limited to teams doing Agile, and chances are these points apply to plenty of teams doing Waterfall, albeit to a different extent. Regardless, all mainframe teams are being required to move faster today, whether they're striving to become Agile or fighting to keep their Waterfall status—and when you move faster, technical debt accrues.

Now that you have an idea of why technical debt is growing and where, let's talk about how you can pay it off. In my next post, I'll discuss this as well as some of the consequences of technical debt and

how to prevent and manage it moving forward.