DYNANET CORPORATION

# Strategies for Reducing Technical Debt

**By Mike McEwen, PMP, Dynanet Director of Software Solutions**

Speed of development is a benefit of Agile software projects but can have an undesirable side effect. Software that developers deliver in an iteration (e.g., Scrum sprint) might work but not have the same level of quality, reliability, or consistency as more "polished" code. The team might also take credit for finishing a user story that is later found to be incomplete. As time passes, bugs may surface and the code base may become misaligned with upgraded libraries, patched operating systems, and other newer system components. The need to refactor, fix, or upgrade delivered code is called **technical debt**, a phrase that originally described the cumulative consequences of taking development shortcuts. It tends to drag a team down because it makes the code harder to maintain, stealing development cycles from new features. Technical debt is common in growing systems because it's hard to create an initial design that accommodates new features added years later. Over time, the team may also discover new ways to implement features that it wants to apply to code delivered in the past.

However, not all technical debt needs to be paid. For example, a Product Owner may not want to pay off long-term technical debt if the product is being replaced or retired. Servicing technical debt is also often not important to end users because they prefer new features over bug fixes and some fixes might not yield a noticeable difference in the user interface. You should always evaluate technical debt to see if it's worth fixing.

Here are nine ways to avoid or reduce technical debt:

1. **Invest in a Good Design**. Invest in quality upfront with a good design, architecture, platform, etc. that reflects the long-term product roadmap. Continuously invest in quality during each sprint. Apply design patterns and follow coding standards. Embrace the Keep It Simple, Stupid (KISS) principle as much as possible.
2. **Invest in Your Team.** Hire carefully and train extensively to ensure a competent development team.
3. **Follow Best Practices.** Avoid shortcuts, perform code reviews on everything, automate testing, adopt test-driven development, and implement continuous integration.
4. **Improve Your Definition of Done** to avoid delivering code that includes technical debt. For example, you might require code to successfully pass independent (system) testing before being considered complete. You can also run a complexity analysis and make sure the McCabe index is less than 10. Resist the temptation to call something done you know to be incomplete. Don't deploy interim solutions to make the delivery date unless everyone agrees to replace it in the next iteration.
5. **Educate Stakeholders** (preferably before agreeing to rush code to production) by expressing the negative impact of technical debt in business terms (e.g., the product might be less secure, require more maintenance or workarounds, or be slow). Try to quantify the detriment as much as possible and explain the risk of adding or living with technical debt.
6. **Be Vigilant about Tracking and Prioritization.** Track and prioritize the fixes for technical debt in the product backlog or separate list. If using the backlog, make sure to distinguish between debt and new features. Monitor the team's velocity and try to detect any fixes required as soon as possible. Prioritize remediation by helping the Product Owner understand it or agree on a timetable for resolving it. Your goal is a balance of new features and debt remediation within a release. Pay off the highest debts (i.e., those causing the most cost or pain) first.

7. **Establish a Remediation Budget and Monitor.** Calculate the quantity of debt in story points or hours. This is your debt payoff balance. Also calculate the rate debt is changing to see the trend.
8. **Include Remediation in Iterations.** You can include some fixes in sprints with new features or have dedicated remediation sprints.
9. **Refactor relentlessly.** Try to include refactoring in story point estimates for new features or assign related tasks to iterations. If that's not possible, assign refactoring to developers who finish their other assignments ahead of schedule. Explain the importance of refactoring to Product Owners to get refactoring prioritized higher.

Following these strategies will limit the technical debt you accumulate and "pay down" any existing debt it makes sense to remove.

**Dynanet Corporation**, a minority-owned small business located in Elkridge, Maryland, has provided a range of information technology (IT) services to multiple Government agencies. Our focus on customer satisfaction and performance has established long-term relationships with the United States Air Force (USAF), Food & Drug Administration (FDA), the General Services Administration (GSA), the Office of Personnel Management (OPM), and the States of Maryland and Pennsylvania. During the last few decades, our executive leadership team has built Dynanet's dedicated workforce from 20 employees to over 125 key personnel. Dynanet is committed to employee engagement and maintains a retention rate of over 90%. Our managers are PMP certified and our employees maintain many sought after professional certifications, such as Certified Scrum Masters and ITIL certified. Additionally, Dynanet has been appraised at CMMI Maturity Level 3 (ML3) and our quality management processes have been audited and found to be in compliance with the ISO 9001:2008 quality standards. Further, our company has a Top-Secret facility clearance. For more information, please visit Dynanet's website at www.dynanetcorp.com.

*About the Author: Mike McEwen (PMP, CSM) is a servant-leader who forms amazing teams and helps them deliver award-winning technology solutions to commercial, non-profit, and government customers.*