Print
Close

Seven Strategies for Technical Debt

Ryan Shriver

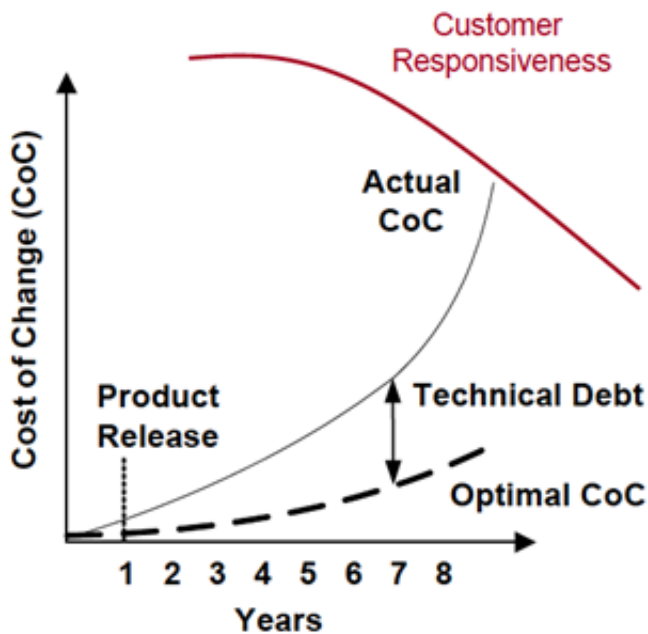
September 13, 2010

Have years of haphazardly designed and piecemealed systems paralyzed your organization? Are costs rising and customer responsiveness falling? Do your teams complain about how hard it is to make simple changes? These are all telltale signs of an all-too common problem: technical debt. As its name suggests,technical debt refers to the tradeoffs teams make with respect to maintainability and adaptability in order to meet release dates. After a series of seemingly small tradeoffs for the sake of speed (often at the request of management), teams incur technical debt that makes future changes costlier and riskier. This hidden cost--when left uncontrolled--can force organizations to face costly decisions of when to re-write or replace systems because their maintenance costs are too high.

Fortunately, this fate is avoidable--but only for organizations that make a commitment to investing in technical debt repayment. This article introduces you to technical debt, including common symptoms of organizations suffering under technical debt. You'll learn the basic steps to set up a repayment plan, the common causes of technical debt and effective strategies for paying it down.

What is Technical Debt?
Ward Cunningham first coined the term in the early 1990s (as described in this video). Although there are many definitions of technical debt, the three that I like most include:

- *"... doing things the quick and dirty way sets us up with a technical debt, which is similar to a financial debt. Like a financial debt, the technical debt 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. We can choose to continue paying the interest, or we can pay down the principal by refactoring the quick and dirty design into the better design. Although it costs to pay down the principal, we gain by reduced interest payments in the future."*- Martin Fowler
- *"'Technical debt' refers to delayed technical work that is incurred when technical short cuts are taken, usually in pursuit of calendar-driven software schedules. Just like financial debt, some technical debts can serve valuable business purposes. Other technical debts are simply counterproductive. The ability to take on debt safely, track their debt, manage their debt and pay down their debt varies among different organizations. Explicit decision making before taking on debt and more explicit tracking of debt are advised."*- Steve McConnell
- Jim Highsmith offers a graphical way of showing technical debt as the difference between the actual cost-of-change and the optimal cost-of-change, as seen below:



Technical Debt Chart from Jim Highsmith

I like Jim's definition because it's a bit more holistic and includes the processes in addition to the code contributing to technical debt.

Common Symptoms of Technical Debt

Now that you know what it is, how do you know if technical debt is impacting your organization? The most common symptoms I see indicating technical debt include:

- **Poor Customer Responsiveness**: Everyone's generally frustrated about the elapsed time to deliver updates to customers. Because management is customer facing and making personal commitments, they in particular don't understand why they can't be more responsive to seemingly simple customer requests. Technical debt can often be the culprit.
- **Long Delivery Times**: Related to responsiveness, it seemingly takes forever to get new releases out the door. Six months to a year is not uncommon, and management in particular is perplexed as to how to shrink their time to market in order to be more competitive.
- **Late Deliveries**: Managers often complain how they just "*relied on the estimates developers gave them*." What they don't know is that it's extremely hard for even the best developers to estimate when there's a high level of technical debt. This is due to the risk and uncertainty with making any change, even minor ones (i.e. fixing one bug creates two more). Technical debt related to poor design and development practices is a tax on development, increasing the likelihood of late deliveries and frustrated teams.
- **Lots of Defects**: Managers and teams observe long periods of time spent in testing with many defects found very late in the process. This can cause late deliveries or require the creation of expensive hot fixes after release. In one recent company I visited, a variety of stakeholders from managers to team members estimated that 50 percent of their development capacity was spent fixing defects. In other organizations I've worked with, 30 percent is not uncommon. Not surprisingly, this leads to our next symptom…

- **Rising Development Costs**: This tends to get management's attention, especially when it considers how rising costs negatively impact plans for business growth. Systems with high levels of technical debt don't scale well to meet the needs of business, which inhibits revenue growth and results in shrinking profits due to ever-rising development costs. As Olivier Gaudin says, "Bankruptcy is the logical extension of technical debt uncontrolled...we call it a system rewrite."
- **Frustrated and Poor Performing Teams**: When I ask the managers and teams about their morale, they often talk of being dedicated employees who are tired of working countless nights and weekends working overtime. Some are burned out and may quit (or already have), but most want to deliver real value to the customers. Teams are sandwiched between managers demanding dates and their own desire to "do the right thing" and pay down technical debt throughout the project. Morale deteriorates as technical debt continues to mount with each short cut taken without a view toward ending the vicious cycle.

Of all the symptoms, I personally connect with the last one the most. I work with a variety of organizations where I see the impact technical debt makes directly on the joy and livelihood of the people I meet. Development is supposed to be challenging and fun, enjoying the interactions of colleagues and solving tough problems with creative solutions. But often times it's not, and managers are just as frustrated as their team.

Tackling technical debt head on requires engagement, motivation and a plan. By delivering focusing improvements iteratively, paying down your technical debt can deliver value to executives, managers and the team.

Creating a Repayment Plan
Tackling technical debt head on requires engagement, motivation and a plan. By delivering focused improvements iteratively, paying down your technical debt can deliver exponential value to customers, executives, managers and the team as you improve the future value of your team's efforts. A repayment plan begins with:

- Engaging executive management in explaining what technical debt is
- Explaining how it's impacting responsiveness, costs, delivery and morale
- Getting buy-in to make the necessary investments to pay down technical debt and proactively manage it going forward
- Creating a realistic plan and appropriate governance model

When engaging executives, it often helps to explain technical debt as analogous to financial debt. As Tom Brazier explains, "*any time a software team follows bad engineering practices they incur two kinds of cost. First, there is the cost of repaying the 'capital', i.e. undoing bad code and replacing it with well-engineered code. Second, there is the 'interest', the ongoing increased cost of supporting, maintaining and enhancing the software*."

These analogies can help teams explain their improvements as repaying capital through refactoring code and adopting agile engineering practices. Interest payments act as impediments to higher velocity, holding teams back from hyper productivity. If management wants the responsiveness, adaptability and savings associated with hyper productivity, tightly managing your technical debt is a must.

As Israel Gat notes, monetizing technical debt can have two large implications:

- A credit limit on technical debt can be established. For example, when the technical debt reaches a certain level (say 25 cents per line of code), new functionality is put on hold. The team applies itself to aggressive refactoring to reduce the debt to an acceptable level.
- For companies who capitalize software, technical debt could become a line item on the balance sheet. It will simply be listed as a liability.

Whether you value your technical debt based on your team's estimates to address them or through the use of tools like the Technical Debt Plugin available for Sonar, monetizing and relating in financial terms can help make your case for investment from management.

Planning to get out of technical debt means establishing some objectives and prioritizing for focus. Objectives such as "improve time to market", "reduce defects" and "improve team morale" are often the inverse of the symptoms. For clarity, I prefer to quantify these objectives by setting scales of measure along with desired target and constraint levels of performance. When used with impact estimation, organizations can make value-based decisions based on the teams estimated improvements to prioritized objectives. The ideas with the best "bang for the buck" at reducing technical debt are prioritized for development in upcoming releases.

If your teams are using agile, then once the tasks for reducing technical debt are prioritized they can be integrated with the product backlog. During release planning, the technical debt tasks can be scheduled alongside stories for delivery.

Strategies for Paying Down Technical Debt

By now hopefully you've convinced your organization to make investments in paying down principle and reducing interest payments. But where should you start? Your technical team will certainly have ideas, so engage them. Have them setup tools such as Sonar to baseline the current system with respect to important code quality metrics. This can be used to measure progress and show stakeholders that technical debt exists and is measurable. In my travels to different software organizations, I see many of the same issues again and again related to the causes of technical debt. The most common causes of technical debt and my recommended repayment strategies include:

Common Causes of Technical Debt	Strategies for Paying Down Technical Debt
Poor coding and testing practices	Most poor coding and testing practices I see are done by good people who tended to 1) lack the time and/or 2) the knowledge to "do it right". Management has to make the time through proactive investment, but so does the team. Each team member needs to invest in their own knowledge and education on how to write clean code, their business domain and how to do their jobs optimally. While teams learn during the project through retrospectives, design reviews and pair programming, teams should learn agile engineering practices for design, development and testing. Whether through courses, conferences, user groups, podcasts, websites or books, there are many options for learning better coding practices to reduce technical debt.
	Most architects over-focus on user features and under-focus on the important qualities such as maintainability and adaptability that

Poor system design	directly relate to technical debt. I recommend identifying and quantifying target and constraint levels for each critical quality, then using your current baselines as the starting point for improvements. Additionally, architects need to learn about evolutionary design principles and refactoring techniques for fixing poor designs today and building better designs tomorrow. Lastly, a governance group should meet periodically to review performance and plan future system changes to further reduce technical debt.
Poor communication and collaboration	If nothing else, start doing a stand-up meeting once a day with the entire team to synchronize your schedules. Keep it to 15 minutes but use it as a way ensure things are getting done and people are collaborating and focused on the most important tasks. Ideally setup common open space for the cross-functional teams with large visible information radiators to show progress. Let the teams self organize. Setup Task Boards, Burndown Charts or Kanban Boards for very simple, low-cost ways to reduce technical debt.
Lack of standard development process	I'm continually amazed at how many organizations don't have standard, repeatable processes for delivering solutions to customers. They have "their blended way" and can often point to a few slides diagramming their process, but interviews quickly review the documentation doesn't reflect how it's done in practice. My recommendations are to establish a process for iteratively delivering value to stakeholders. I think Scrum is a great framework for this, especially when paired with XP practices. Lean and Kanban are other popular approaches as well. Most teams new to agile typically receive guidance from agile coaches like myself who provide consulting and training on transitioning to new ways of delivering value.
Poor requirements with no common definition of "done"	Establish a common "definition of done" for each requirement, user story or use case and ensure its validated with the business before development begins. A simple format such as *"this story is done when: <list of criteria>"* works well.Theproduct owner presents "done" to the developers, user interface designers, testers and analysts and together they collaboratively work out the finer implementation details. Set expectations with developers that only stories meeting "done" (as validated by the testers) will be accepted and contribute toward velocity. Similarly, set expectations with management and analysts that only stories that are "ready" are scheduled for development to ensure poor requirements don't cause further technical debt.
	The availability of tools to automate most all aspects of development is amazing. In all popular languages and platforms today, open source and commercial tools are available to automate builds and perform the continuous integration of code changes, unit testing, acceptance testing, deployments, database setup, performance testing and many other common manual activities. In addition to reducing manual effort, automation reduces the risk of mistakes and over-

Lots of manual effort for a release	reliance on one individual for performing critical activities. I recommend to first setup automated builds (I prefer Ant, nAnt or rake), followed by continuous integration (I prefer Hudson). Next, set up automated unit testing (I prefer JUnit, NUnit or RSpec) and acceptance testing (I prefer FitNesse and Selenium). Finally, set up automated deployments (I prefer Capistrano or custom shells scripts, whatever works). It's amazing what a few focused team members can accomplish in a relatively short period of time if given time to focus on automating common activities to reduce technical debt.
Lack of proactive investment in paying down debt	As discussed above, I recommend engaging management and explaining what technical debt is and how it impacts them using analogies they understand. Monetizing the debt will likely make it easier to justify investments, especially if listed as a liability on the balance sheet. Sometimes you must take on technical debt, but this should be a conscious decision, not a reckless one. As Martin Fowler says, "*The useful distinction isn't between debt or non-debt, but between prudent and reckless debt*." Once you get a commitment, work with product planning to reserve budget in the upcoming release for improvements that pay down technical debt and keep track of progress in your results backlog. To ensure repayment of technical debt remains on track, create a governance model that includes the right mix of stakeholders who meet periodically to review progress to date and plan future investments to reduce technical debt.

Summary

Like any personal debt repayment plan, organizations that aggressively pay down technical debt only to relapse and have it build up again will inevitably need to invest in big initiatives to pay it down again. This can be very de-motivating for your team. Organizations that take a longer-term view and continually manage technical debt as they would manage any other liability can make informed decisions based on when it's advantageous to take on technical debt and when it's advantageous to pay it down. These organizations also establish governance structures to ensure improvements continue into the future.

In this article, you've learned what technical debt is and the common symptoms of organizations suffering under it. You've learned the basic steps to set up a repayment plan, the common causes of technical debt and effective strategies for paying it down. Hopefully you've learned some ideas you can try in your organization to help reduce your technical debt and not only make management happier, but also the team members who want to do "the right thing"!

Ryan Shriver is a Managing Consultant with Dominion Digital, a Virginia-based process and technology-consulting firm. Based in Richmond, he leads the IT Performance Improvement Solution which includes Agile Adoption, Agile Engineering, IT Process Improvement and IT Services Management. With a background in systems architecture and large-scale agile development, Ryan currently focuses on measurable business value and systems engineering. He writes and speaks on these topics in the United States and Europe, posting his current thoughts at theagileengineer.com. Ryan can be reached at rshriver@dominiondigital.com.

Copyright © 2010 gantthead.com All rights reserved.

The URL for this article is:
http://www.gantthead.com/article.cfm?ID=258854