UK Edition
Chapter Extract

Tech Trends 2014

Technical debt reversal

Deloitte University Press

Download the full report at **www.deloitte.co.uk/techtrends**

Technical debt reversal

Lowering the IT debt ceiling

Technical debt is a way to understand the cost of code quality and the impacts of architectural issues. For IT to help drive business innovation, managing technical debt is a necessity. Legacy systems can constrain growth because they may not scale; because they may not be extensible into new scenarios like mobile or analytics; or because underlying performance and reliability issues may put the business at risk. But it's not just legacy systems: New systems can incur technical debt even before they launch. Organisations should purposely reverse their debt to better support innovation and growth – and revamp their IT delivery models to minimise new debt creation.

TECHNICAL debt is not a new term, but it's gaining renewed interest. Originally coined by Ward Cunningham in 1992, the phrase describes the "not quite right" code typically introduced with initial software releases because of an incomplete understanding of how the system should work.[1] Organisations that regularly repay technical debt by consolidating and revising software as their understanding grows will likely be better positioned to support investments in innovation. And like financial debt, organisations that don't "pay it back" can be left allocating the bulk of their budgets to interest (i.e., system maintenance), with little remaining to develop software that can support new opportunities.

Technical debt is often the result of programmers taking shortcuts or using unsophisticated techniques. It's typically misfeasance, not malfeasance. For example, a developer may copy and paste code blocks without thinking through the longer-term consequences. If the code ever needs to be updated, someone will have to remember to fix it in each instance.

But sometimes, technical debt is simply the result of dealing with complex requirements. To meet a project deadline, complicated proprietary code may be developed, even though simpler alternatives may have been available. With each such action, technical debt proliferates. This is like high-interest, short-term borrowing. If you don't pay off the debt promptly, compounding kicks in.

The impact of accumulated technical debt can be decreased efficiency, increased cost, and extended delays in the maintenance of existing systems. This can directly jeopardise operations, undermining the stability and reliability of the business over time. It also can hinder the ability to innovate and grow.

Articulating technical debt is the first step in paying off its balance. With new tools for scanning and assessing software assets, CIOs can now gauge the quality of their legacy footprint – and determine what it would cost to eliminate the inevitable debt. A recent study suggests that an average of $3.61 of technical debt exists per line of code, or an average of more than $1 million per system.[2] Gartner says that "current global IT debt is estimated to stand at $500 billion, with the potential to rise to $1 trillion by 2015."[3] While the idea of debt doubling in a year's time may seem astonishing, we're in the midst of unprecedented investments in disruptive technologies – often with deep hooks into core systems. The push for rapid innovation in unproven domains is also leading to compounding debt.

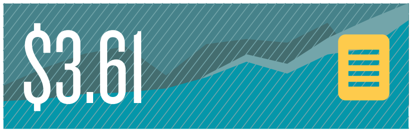
These estimates address only the literal definition of technical debt – how much it would cost to fix the exposed code quality issues. But there's also another dimension, which we call "architectural debt." Architectural debt refers to the opportunity costs associated with system outages or the inability to deliver new capabilities. In some cases, architecturally complex defects can absorb as much as 52 percent of the total effort spent repairing defects.[4] They can also derail new initiatives.

Technical debt is not limited to legacy systems; every new project has the potential to add to the backlog. With that in mind, you should incorporate the cost of technical debt into project management processes and portfolio reporting. This kind of transparency can not only raise awareness of quality among development teams, but can also provide a foundation for talking to the business about the hidden cost of IT delivery. By documenting your debt-decreasing efforts, you can account for those efforts – important progress that would likely not otherwise be visible (or appreciated).[5]

The ability to quantify technical debt can provide a common point of reference for the C-suite when you are deciding how to prioritise IT projects for an organisation. Typically, technical debt should be paid down within the context of delivering against business priorities by incrementally refactoring existing solutions and using improved development processes to minimise new debt accumulation. Incorporating techniques described in our *Real-time DevOps* chapter[6] can help reduce waste generated when software is developed.

Some organisations may also need to spur projects that address especially messy issues such as bolstering performance, preventing production issues, or preparing for future strategic investments. The goal is a sustained, prioritised reduction of the balance sheet, where each project systematically improves on the baseline.

For most organisations, technical debt comes with the territory, an unavoidable outcome of decades of technology spend. The big question is: How will you manage the liability? Understanding, containing, and mitigating technical debt can be a platform, not only for a stronger IT foundation, but for a renewed level of trust and transparency with the business.



Technical debt per line of code within a typical application.[1]

The defect removal efficiency of most forms of testing.[2]

Estimated global annual expenditure on software debugging in 2012.[3]

Portion of total effort spent repairing architecturally complex defects, though they account for only 8% of all defects.[4]

Sources: [1] Alexandra Szynkarski, "Time to start estimating technical debt," October 29, 2012, http://www.ontechnicaldebt.com/blog/time-to-start-estimating-technical-debt, accessed December 27, 2013. [2] Namcook Analytics LLC, "Software defect origins and removal methods," July 21, 2013, http://namcookanalytics.com/software-defect-origins-and-removal-methods, accessed January 6, 2014. [3] University of Cambridge, "Financial content: Cambridge University study states software bugs cost economy $312 billion per year," https://www.jbs.cam.ac.uk/media/2013/financial-content-cambridge-university-study-states-software-bugs-cost-economy-312-billion-per-year/#.UryqUGRDsS4%20, accessed December 27, 2013. [4] B. Curtis, "Architecturally complex defects," December 19, 2012, http://it-cisq.org/architecturally-complex-defects, accessed December 27, 2013.

Countdown to zero technical debt

NASA's Mars Science Laboratory project was classified as a "flagship mission" – the agency's first in almost a decade. It was a $2.5 billion project to land a car-sized, roving science laboratory, Curiosity, on Mars. The rover launched in 2011 and landed on Mars on August 5, 2012, with the continuing objective of determining whether Mars ever contained the building blocks for life.

Building a roving science lab is an immense challenge. Curiosity is an order of magnitude larger than any rover that had previously landed on Mars: It weighs almost a ton, stands seven feet tall, contains a robotic arm that could easily pick up a person, and includes a laser that vaporises rocks. Curiosity's software is essentially the brain of the rover – integrating its many hardware functions to provide mission-critical functionality such as the descent and landing sequence, autonomous driving, avionics, telecommunications, and surface sample handling.

The software initially developed for Curiosity was inherited from previous rover missions. The core architecture was developed in the 1990s on a shoestring budget. The Curiosity project put approximately four years of work into building on top of that architecture for NASA's most complex mission to date. As the launch date approached, NASA started to see that the project wasn't coming together: The software had bugs and inexplicably failed tests; there were issues with the hardware and the fabrication of key components.

The project faced a difficult question: Do we push on towards a 2009 launch or delay the mission? The unique aspect of launching a mission to Mars is that the opportunity only exists once every 26 months, when Earth and Mars align. If they delayed the launch two years, there was a risk that the project might be cancelled altogether.

The project team decided to postpone the mission and began an incredible regrouping effort. The software team had to quickly decide whether to fix the current software or to start over completely from scratch. Given the existing software's technical debt, it was unlikely they could determine the magnitude of the lurking issues, or confidently plan for new project milestones. The decision was made to tear down the foundation and rebuild using the old code as a reference.

The team started from the beginning: revisiting the requirements, software design, coding, and reviews, and testing and implementing standard processes. The team instituted what they called the "Power Ten," a set of 10 basic rules each developer followed. The team developed coding standards, implemented multiple automated code analysers and testing tools, and established a cadence of releases – one every four months. They unit tested every line of code and instituted code reviews early in the development lifecycle. Two hundred code reviews produced 10,000 peer comments and 25,000 tool comments – each one reviewed and resolved.

The results were staggering: 3.5 million lines of code, over 1 million hand-written, across 150 different modules. But this time, the numerous bugs and unexplained failures were gone. The standards, though they required additional work, added stability and quality. And with the fresh start, the team were adamant that technical debt be minimised – building a new foundation for future missions.

Though NASA's approach required a remarkably difficult decision, the results were worth the effort. The world can now watch as Curiosity tells us more than we ever dreamed we might know about Mars. And the achievements of the mission led to the announcement of a new $1.5 billion mission to Mars in 2020.

Where do you start?

Technical debt calculation can begin when you have clear visibility to the quality of code for legacy systems as well as projects on the horizon. Only with both sets of information can you make the trade-offs necessary to manage technical debt effectively. For companies eager to get ahead of the technical debt curve, here are some important steps:

- **Assess the status of code for all significant investments.** Calculate your technical debt. Know the size of the hole you're in – and whether or not it's getting deeper. Evaluate the importance of each system to understand whether the technical debt has to be addressed – and in what timeframe. Aim for surgical repairs when possible, but recognise that some ageing systems may be beyond incremental fixes. Prevention is preferred, but early detection at least allows for a thoughtful response.

- **Find out how future investments are dependent on your legacy systems.** Is your architecture ready for new initiatives? Can it scale appropriately? How well are back-end complications understood and fed into planning efforts? Should you launch legacy modernisation efforts now to get ahead of impending business demands?

- **Think through the availability of talent to support debt remediation.** For some aging systems, your resources may not be sufficient for cost-effective updating. Talent should be factored into your analysis. Think of it as a multiplier on top of the raw technical debt calculation – and use it to define priorities and timelines.

- **Hold developers accountable.** Consider rating and rewarding developers on the quality of their code. In some cases, fewer skilled developers may be better than volumes of mediocre resources whose work may require downstream reversal of debt. Regularly run code complexity reviews and technical debt assessments, sharing the results across the team. Not only can specific examples help the team improve, but trends can signal that a project is headed in the wrong direction or encountering unexpected complexity.

- **Spread the wealth (and the burden).** Communities are great ways to identify and address technical debt. Peer code reviews are leading practices for informal spot checks. Formal quality assessments by seasoned architects can find issues that would be undetectable with standard QA processes. Learn from open source communities, where quality is continuously refined by the extended pool of developers poring over each other's code.[7]

- **Determine your debt repayment philosophy.** Companies have different profiles when it comes to debt for the various parts of their asset pools. Debt is not inherently bad; it can fuel new investments and accelerate product launches. But left unchecked, it can be crippling. There's no single right answer for the appropriate amount of technical debt, but its accumulation should be a conscious, transparent decision.

Bottom line

When CIOs operate like venture capitalists,[8] technical debt is a big part of the financial picture. Without a clear view of the real cost of legacy systems, CIOs lack the information required to make effective decisions about new initiatives and investments. While it's important not to get obsessed with technical debt, it's also critical to understand and plan for it. Every new project automatically comes with technical debt as a cost of doing business. Reversing technical debt is a long-term investment, but if left unaddressed, it can bankrupt your ability to build for the future. Capers Jones, a long-term technical debt specialist, once said: "If you skimp on quality before you deliver software, you end up paying heavy interest downstream after the software is released for things you could have gotten rid of earlier, had you been more careful."[9] He was right.

Authors
Scott Buchholz, director, Deloitte Consulting LLP
David Sisk, director, Deloitte Consulting LLP

1. Ward Cunningham, "The WyCash Portfolio Management System," http://c2.com/doc/oopsla92.html, accessed December 31, 2013.
2. CAST, "Technical debt estimation," http://www.castsoftware.com/research-labs/technical-debt-estimation, accessed January 9, 2014.
3. Andy Kyte, "Measure and manage your IT debt," Gartner, Inc., August 9, 2010 (last reviewed June 19, 2013).
4. B. Curtis, "Architecturally complex defects," December 19, 2012, http://it-cisq.org/architecturally-complex-defects/, accessed January 9, 2014
5. David K. Williams, "The hidden debt that could be draining your company," Forbes, January 25, 2013, http://www.forbes.com/sites/davidkwilliams/2013/01/25/the-hidden-debt-that-could-be-draining-your-company/, accessed December 21, 2013.
6. Deloitte Consulting LLP, Tech Trends 2014: Inspiring disruption, 2014, chapter 10.
7. Simon Phipps, "Oracle's closed approach keeps Java at risk," Infoworld, April 26, 2013, http://www.infoworld.com/d/open-source-software/oracles-closed-approach-keeps-java-risk-217297, accessed December 21, 2013.
8. Deloitte Consulting LLP, Tech Trends 2014: Inspiring disruption, 2014, chapter 1.
9. Joe McKendrick, "Will software publishers ever shake off their 'technical debt'?," ZDNet, January 26, 2013, http://www.zdnet.com/will-software-publishers-ever-shake-off-their-technical-debt-7000010366/, accessed December 21, 2013.

Contacts

John Starling
Partner, Technology Consulting
Deloitte MCS Limited
020 7007 2389
jstarling@deloitte.co.uk

Kevin Walsh
Head of Technology Consulting
Deloitte MCS Limited
020 7303 7059
kwalsh@deloitte.co.uk

About Deloitte University Press
Deloitte University Press publishes original articles, reports and periodicals that provide insights for businesses, the public sector and NGOs. Our goal is to draw upon research and experience from throughout our professional services organization, and that of coauthors inacademia and business, to advance the conversation on a broad spectrum of topics of interest to executives and government leaders.

Deloitte University Press is an imprint of Deloitte Development LLC.

Designed and produced by The Creative Studio at Deloitte, London. 32430A