October 15, 2018 | By Karen D. Schwartz

Managing technical debt, once and for all

Some degree of technical debt may be inevitable in any of your software development projects. But you should work at identifying it, paying it down, and making sure it doesn't keep popping up as fast as you can knock it down.

"There's never time to do it right, but there's always time to do it over" is an adage that applies entirely too often in programming and other IT projects. In the guise of "let's just get it done," technologists often choose the fastest, easiest method of writing code or developing new

(/us/en/insights.html)

product features rather than taking a more thorough, thoughtful approach (https://www.hpe.com/us/en/insights/articles/becoming-a-senior-developer-9-experiences-youll-encounter-1807.html) that would make it easier for the next developer on the project.

The collateral damage to the quick-and-dirty approach is commonly called technical debt (https://www.agilealliance.org/introduction-to-the-technical-debt-concept/). From a user perspective, technical debt is anything that impacts the user experience, such as the user interface, system performance (https://www.hpe.com/us/en/insights/articles/checklist-optimizing-application-performance-at-deployment-1805.html), or security. From a developer standpoint, it's anything that prevents the team from adding new features or supporting the application effectively. By failing to think through the project plan and build it right the first time, the organization later wastes time and money (https://xkcd.com/1421/) in bug fixing, maintenance, and re-engineering.

While all applications and projects generate some amount of technical debt, there is a point where the weaknesses become too much. Those issues can cause applications to experience more bugs, crash, go offline unpredictably (https://www.hpe.com/us/en/insights/articles/how-to-design-software-that-doesnt-crash-when-the-internet-connection-fails-1705.html), or simply break. One survey found that technical debt severely limits an IT department's ability (https://www.accenture.com/t20180223T103955Z__w__/us-en/_acnmedia/PDF-72/Accenture-Exponential-IT-Anthem.pdf) to innovative, migrate to new technologies, or respond to market changes.

"It gets pretty frustrating when technical debt starts impacting my sprint testing, and the list of issues that are deferred until the debt is resolved keeps growing," says Vikram Sharma, technical project manager at Impetus Technologies. "It leads to back-and-forth triaging and frequent reprioritization."

Sometimes, technical debt is OK

The detritus left by sloppy programming and lax coding practices accounts for only a portion of technical debt. Other technical debt is the result of older systems that have become increasingly complex (https://www.hpe.com/us/en/insights/articles/what-to-do-when-equipment-is-old-in-the-way-and-yet-still-functional-1807.html) over time, new security vulnerabilities that weren't around when the original code was written, or changing business or technology requirements.

Sometimes developers create technical debt on purpose. For example, a software developer working with a startup might create a minimum-viable product to validate an idea or garner feedback. In that case, the developer doesn't want to over-engineer or spend too much of the customer's time or money. The developer is willing to take on some technical debt in the process, knowing that if the product takes off or the customer raises additional funding, those issues can be addressed.

The bottom line is this: Some degree of technical debt is inevitable, and it really doesn't matter how it occurred. The key is identifying it, paying it down, and making sure it doesn't keep popping up as fast as you can knock it down.

Culture matters

Technologies and techniques are important to paying down and minimizing the regrowth of technical debt, but attitude, culture, and habits are probably more so.

Sally Ransom, a software architect at Schlumberger, stresses the importance of scoping how products are to be implemented—and recognizing that sometimes it causes you to butt heads with other people. When you know that something ought to be done a different way, giving in introduces technical debt, as she explained in a presentation about the role of software architects at the recent Grace Hopper conference.

You won't win every battle, Ransom said, even when you know a project may cost more in maintenance and fixing bugs. Your only option is to argue with the product manager or stakeholder by pointing out that cutting corners is a risk and that it's better to put attention on mitigating the problem now: "If the solution is not strong and flexible, it will cause problems in the future."

Sometimes it's a more ingrained, systemic problem. Development processes may be weak, or the company may prioritize budget over all else.

"Often, it's because developers simply don't want to spend time on tasks that are boring or routine, like maintaining the application stack, instead of building new features," says Alan Zucker, founding principal of Arlington, Virginia-based Project Management Essentials. "Had Equifax patched those servers (https://www.ftc.gov/equifax-data-breach), for example, we wouldn't have had the greatest breach to date."

It's true that managing technical debt isn't sexy—it's natural for developers to prefer working on new features instead of functionality to ensure better uptime—but it's got to be part of the deal. Some development teams, for example, are required to spend 10 percent of their time during every sprint dealing with technical debt.

Sometimes, you need to insert the quality-software attitude into the culture with incentives. One company, for example, gave away an iPad. "It was hard for us to get started with unit testing because it was a new process for us, so we told our developers that everybody who builds their first unit test would get a chance at winning an iPad," says Patrick Turner, CTO of Winston-Salem, North Carolina-based Small Footprint, which specializes in agile software development. "The next thing you know, many of them did it, and everybody's confidence increased."

Identifying and prioritizing technical debt

So, how can you recognize when you're incurring technical debt, and what can you do to avoid it? It's as much art as science. Best to let the experts tell you how they do it.

Start the old-fashioned way by communicating with users and software developers, Turner advises. For example, his company relies on user experience researchers, whose job is to talk to users and watch them use the application. The UX researchers do their best to understand what works and what doesn't, what's easy to use, what performs poorly, and where there are obvious security vulnerabilities (https://www.hpe.com/us/en/insights/articles/the-it-security-disconnect-stop-talking-about-it-and-just-fix-it-already-1807.html).

While these issues might not seem like technical debt, Turner says they are. "Steve Jobs is credited with the idea that if software needs a manual, there's something wrong with it," he says. "As a result, this has created a lot of technical debt that might not have otherwise been seen as such."

Turner also emphasizes talking to the development team. “Try to understand what’s impeding them from getting their job done,” he says. “Read between the lines. If you are consistently 50 to 100 percent over your estimated time to complete a project, you can be sure there is something going on under the covers that needs to be addressed.”

What you measure is what you get

Metrics can help to address technical debt, at least in the sense of finding out where time and money is being misplaced. Kane Mar, an independent agile methodology trainer and coach in Brisbane, Australia, likes to use the number of defects raised from a production environment over a period of time as a guide.

Total cost of ownership also is a valuable metric, Mar says, but it’s much more complex to measure, and different organizations have different TCO calculations. Sharma prefers architecture review metrics (https://robertoverdecchia.github.io/papers/ICSA_2018.pdf), which can be correlated to project quality metrics, covering both code and testing metrics. “This correlation not only provides a good list, but helps identify the impact,” he adds.

Some developers use more automated methods. While no automated tool can identify all types of technical debt, some are helpful. For example, a code analysis or system-level analysis tool can help evaluate code quality, analyze interaction between components, pinpoint defects, and estimate the time it would take to fix the problems. With that information, decision-makers can compare the cost of living with the defective code with fixing it and removing a portion of technical debt, based on time and personnel costs.

Weeding the garden

Paying down technical debt is challenging, but it’s certainly achievable.

Many developers prefer to refactor code instead of completely rewriting it whenever possible. Refactoring is a regular and continuous maintenance of the application, wherein the team regularly improves the application’s performance without changing its underlying functionality. In contrast, a rewrite is usually a fundamental change in the code, including a change in the functionality. “Think of refactoring as changing out your old incandescent bulbs for LED bulbs; a rewrite would be changing out the light fixture,” Zucker says.

Refactoring is an especially useful tactic for older code bases that grew unwieldy and complex over the years. Refactoring can make the code more succinct and help decouple features. This helps prevent the problem of inadvertently breaking other parts of an application during a code rewrite.

Another way to avoid breaking more than you're fixing is by separating parts of the code base into microservices (https://www.hpe.com/us/en/insights/articles/serverless-computing-explained-1807.html). It's another way to decouple applications, but it's done in a way that actually creates independent components, or microservices.

To minimize the amount of technical debt you create in the first place, pick the right technology to work with, advises Richard Simms, CEO of Tyrannosaurus Tech (https://tyrannosaurustech.com/), an Atlanta-based custom software developer.

"We use a lot of new and very modern technology stacks, but we always make a distinction between cutting and bleeding edge," Simms says. "You want to use technologies you know are well supported and well documented, so you know they will stand the test of time."

Choosing the right team members—well-organized developers who write clean code—makes a difference. It's also important to assemble a well-balanced team (https://www.hpe.com/us/en/insights/articles/a-cios-secrets-for-hiring-for-innovation-1703.html) that includes both great developers who want to build new features and those who actually like the challenge of dealing with technical debt.

While there is no one right way to manage technical debt, there is clearly a wrong way: not dealing with it at all. Keeping on top it is key to a functional, productive development environment.

Technical debt: Lessons for leaders

Don't take the easy way out. Keeping technical debt at bay requires a systematic approach.

Fostering a culture that prioritizes management of technical debt—sometimes even offering incentives—is the best way to go.

<

(/us/en/insights.html)

Eradicating technical debt is still a somewhat manual process, but plenty of processes and methodologies can help.

RELATED

15 books that influenced top UX and UI influencers

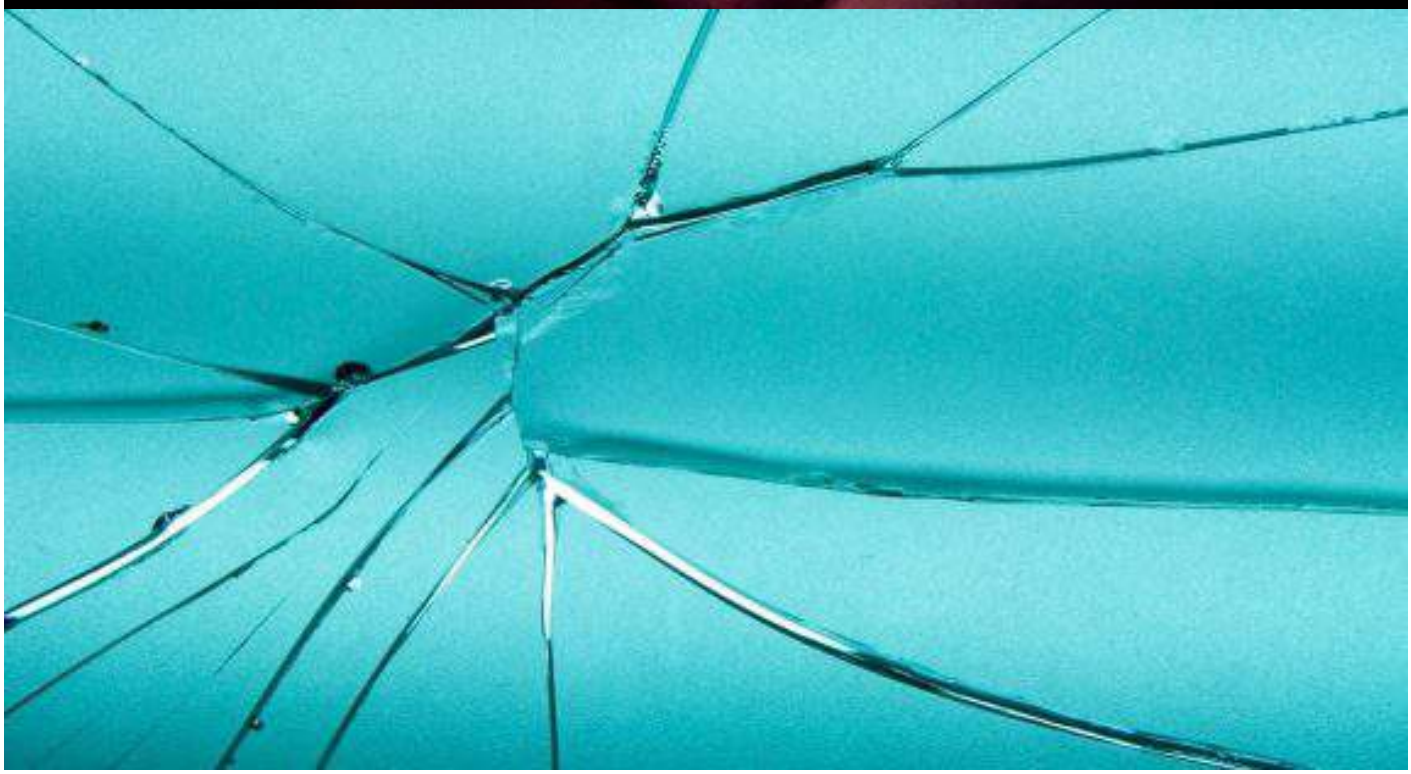
The write stuff

(/us/en/insights/articles/15-books-that-influenced-top-ux-and-ui-influencers-1903.html)

(/us/en/insights.html)

When technology is something to sing about

A bit off key



Top 7 signs a company doesn't have a glass ceiling

Yes, they want you!

(/us/en/insights.html)

(/us/en/insights/articles/top-7-signs-a-company-doesnt-have-a-glass-ceiling-1902.html)

(/us/en/insights/contributors/karen-d-

Karen D. Schwartz (/us/en/insights/contributors/karen-d-schwartz.html)

Follow @karendschwartz

Karen D. Schwartz is a technology and business writer with more than 20 years of experience. She has written about a broad range of technology topics for numerous publications. In addition, she has written many articles, white papers, case studies, and book chapters for vendors in the technology sector.

TOPICS

DevOps (/us/en/insights.html?topic=devops)

Careers & Culture (/us/en/insights.html?topic=careers-culture)

(/us/en/insights.html)

How can we help?

Help me find something

How to Buy(https://www.hpe.com/us/en/buy-parts-products.html)

Product Support(https://www.hpe.com/us/en/support.html)

Email Sales (https://www.hpe.com/us/en/forms/email.modal.html?path=/content/hpe/country/us/en/insights/articles/2018/10/managing-technical-debt-once-and-for-all)

1-888-342-2156(tel:1-888-342-2156)

(https://www.linkedin.com/company/hewlett-packard-enterprise)

(https://twitter.com/hpe) **(http://facebook.com/hewlettpackardenterprise)**

(http://www.youtube.com/hpe)

(https://www.hpe.com/us/en/newsroom/rss.xml)

COMPANY

NEWS AND EVENTS

PARTNERS

(/us/en/insights.html)

SUPPORT

COMMUNITY

CUSTOMER RESOURCES

UNITED STATES (EN) (HTTPS://WWW.HPE.COM/US/EN/COUNTRY-SELECTOR.HTML?NAVIGATEFROM=/US/EN/INSIGHTS/ARTICLES/MANAGING-TECHNICAL-DEBT-ONCE-AND-FOR-ALL-1810.HTML)

(https://www.hpe.com/us/en/home.html)