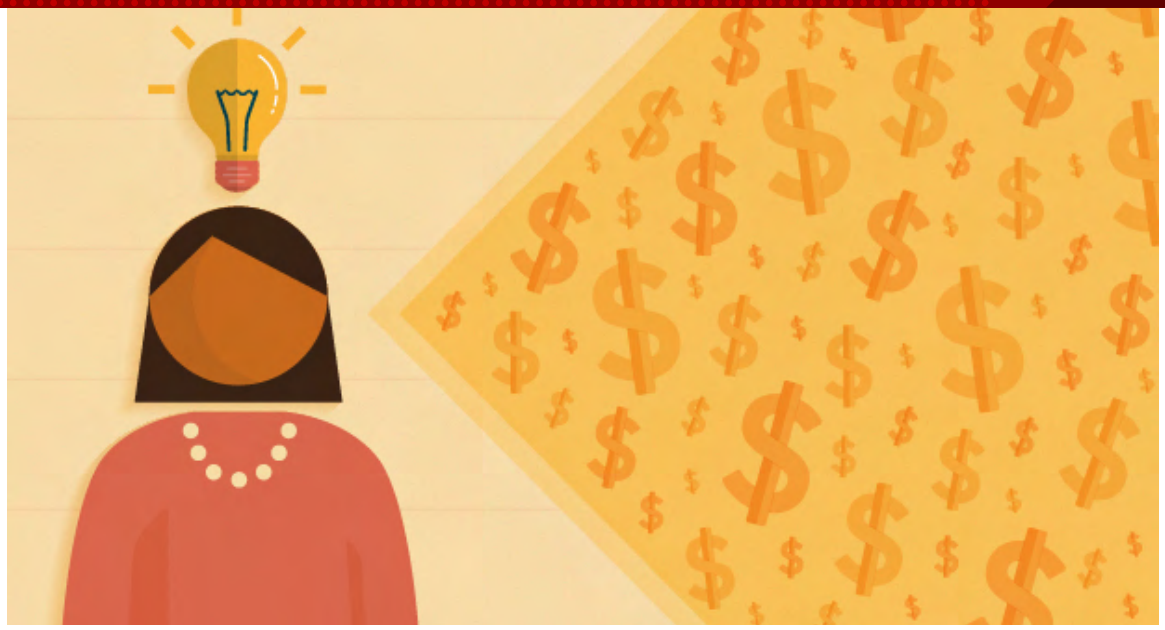
THE
ENTERPRISERS
PROJECT

Technical debt: The IT leader's essential guide

Technical debt:

The IT leader's essential guide



Is technical debt hampering your enterprise's ability to speed up software development and time to market? Is it causing financial or talent headaches? Are you struggling to discuss technical debt with your colleagues? Here's our essential guide on technical debt and how to manage and reduce it.

If you think student loans and mortgages are scary, start talking to IT leaders about technical debt. In the age of digital transformation, technical debt can hold you back in terms of software delivery speed – and burden you with ongoing financial obligations and talent worries.

What is technical debt? Just like a car loan debt, a technical debt is a tradeoff you make now to achieve a specific goal, knowing that a cost will come due later. You may not be able to put cash down to buy a car outright when you need it, but you can finance the car purchase. Similarly, when a software development team faces a tight deadline, you may not have time to completely refactor an application – but you know you will need to at some point. Some decisions to take on technical debt make business sense: You can't wait to respond to a rival's move, for example.

Still, the more of these "make do" decisions an organization chooses – making do with aging legacy software and systems – the more technical debt accumulates. And that debt affects many parts of enterprise IT plans, since today's systems often form complex webs of interdependence.

It's important to realize, too, that technical debt also applies to outdated processes. Leaders need to acknowledge this reality when deciding which debt to eliminate and which to live with for now.

"When thinking about modernization, don't focus solely on modernizing the technology," says David Egts, chief technologist, North America public sector for Red Hat. "Think about modernizing the people and processes too. If you only modernize the technology with a faster computer, you may not be addressing underlying reliability and scalability issues forcing the modernization effort in the first place. These issues can be addressed by reskilling the workforce to write cloud-native applications, as well as using agile and DevSecOps practices."

Working with aging legacy systems also creates talent issues for organizations. For example, if you're too dependent on a small group of people (or person) with institutional knowledge about particular systems, you're at risk if those people leave. And recruiting new talent is harder if your IT shop is seen as a mishmash of aging technologies rather than a place where people can become experts in modern development tools.

This ebook will help you explain technical debt and weigh strategies for getting a handle on it. Let's explore:

Table of contents

How to explain technical debt in plain English

By Kevin Casey

What exactly is technical debt? When discussing your organization's technical debt – and possible changes to it – with various audiences, you need to articulate the key issues in plain terms. Here's expert advice on how to do that

As far as IT topics go, technical debt offers a built-in advantage when it comes to defining and understanding it: The name is more self-explanatory than most tech terms.

"Conceptually, technical debt is similar to financial debt," says Mike Duensing, CTO and EVP Engineering at Skuid. "For the borrower, it is more important to purchase an item and have it in hand now than it is to save up the funds to purchase in all-cash."

We take on technical debt for reasons similar to why we take on financial debt: We need something now (or very soon) that we don't have the "cash" to pay for in full. So we borrow to get what we need. With software, this generally means making coding or design decisions that are sub-optimal – or that we know will need to be addressed and updated in the future – in order to get what we want or need into production sooner.

"We take on technical debt for reasons similar to why we take on financial debt: We need something now (or soon) that we don't have "cash" to pay for in full.

"Technical debt – or code debt – is the consequence of software development decisions that result in prioritizing speed or release over the [most] well-designed code," Duensing says. "It is often the result of using quick fixes and patches rather than full-scale solutions."

Some amount of technical debt is virtually inevitable, Duensing and other software pros point out. Leaders and teams must strike a balance between business goals and a litany of design and implementation decisions; a developer or team that never makes anything other than the most optimal choices in their code is going to be hard-pressed to ship with any kind of regularity or speed.

So the intrinsic financial analogy comes into play again: We must use debt responsibly. We take on debt understanding that it will need to be repaid, for starters. And continuously piling on new debt without ever reducing your existing balances is going to cause all manner of problems – if not actual ruin – down the road.

For many IT leaders, the big picture goal is to move money away from legacy tools and processes over to more innovative work. Other related goals may include a better user experience, say with an updated SaaS application versus a legacy one; a more agile IT infrastructure; and cost savings. One pain point is that legacy systems and processes tied up with technical debt often involve a complicated web of interdependencies, of both tools and policies.

As you discuss your organization's technical debt situation - and possible changes to it - with a variety of audiences, it's important to be able to articulate these issues in plain terms. So let's delve into some expert advice on doing just that.

Technical debt: 3 definitions

Part of responsible technical debt use lies in a straightforward understanding of the concept and how it manifests. Let's arm you with some other definitions of technical debt, in addition to Duensing's above.

"Technical debt is the result of the design or implementation decisions you make and how those decisions age over time if they aren't incrementally adjusted or improved. The longer you hold fast to those designs and implementations without incremental adjustments or improvements, the larger the debt, or effort, becomes to make those needed changes." –Justin Stone, senior director of secure DevOps platforms at Liberty Mutual Insurance

"Technical debt is when the implementation – the code – for a product becomes unnecessarily complex, inconsistent, or otherwise difficult to understand. While there is no perfect code, code that contains technical debt [moves] farther [away] from a good solution for the problem it solves. The more debt, the farther the code misses the target. Technical debt makes it harder to understand what the code does, which makes it harder to build upon, and ultimately results in poor productivity and defects in the product." –Christian Nelson, VP of engineering at Carbon Five

"Technical debt is the cost of technical decisions that are made for the immediacy, simplicity, or [budget] that, while easy today, will slow you down or increase your operational costs/risks [over time]. Most often it's related to technical products, but can be found in most business processes and use cases. Many times this technical debt can turn into 'human spackle,' where knowledge workers do repetitive tasks that could be automated." –Justin Brodley, VP cloud operations & engineering at Ellie Mae and co-host of The Cloud Pod

How to explain technical debt to non-technical people: The car loan analogy

Some technology topics prove more difficult to explain to folks outside of IT. (Think service meshes, or serverless, for example.)

Again, this is a "beauty" of technical debt. The term itself is already doing some of the work for you. Which is good, because technical debt can have a broad impact throughout your organization

– think of the "human spackle" Brodley describes above – and its ability to build new things, effectively serve customers, and more.

"Technical debt is no different than debt in your day-to-day life," Stone says. "You can take on debt to have something of value you need now and worry about that debt later, or you can delay what you need now to save and take on less debt for that same thing of value."

Some things are more of a necessity than others: A car to get to and from work, for example. Many people can't just decide to postpone that purchase for months or years to save up the necessary cash: So they take out a car loan. That means they're paying extra for the car over time, but they get what they need now.

> "There are always tradeoffs for when you want it versus when you need it"

The same principle applies when making development decisions that will impact a codebase over time. (Brodley notes that even if your vehicle is paid in full, you've still got maintenance to consider: Ignoring it will likely cause more expensive problems later, including downtime for major repairs.)

"There are always tradeoffs for when you want it versus when you need it," Stone says. "And just like with personal debt, the longer it takes to pay off technical debt, the more interest and cost accumulate. Time may show that the initial decision wasn't as good as you once thought because of the mounting costs and interest payments."

Another analogy: Meet the Franken-car

Some technology topics prove more difficult to explain to folks outside of IT. (Think service meshes, or serverless, for example.)

Technical debt isn't just a matter of short-term and long-term costs, either. It can also affect system performance itself. Let's continue to use your daily vehicle as a metaphor, except the car itself is a proxy for one of your applications. Actually, in this case, imagine two cars.

"The first [is a] well-designed, high-end model from whatever brand you like the best," Nelson says. It's well-designed on the outside and runs beautifully and reliably on the inside. Everything works as expected, has a purpose, and doesn't surprise you."

That "car" is a software team's goal: Your application works as expected. It fulfills its purpose well without surprise breakdowns or costs.

The second car, however, is not quite the same – even if it functionally resembles a car.

"It gets you from A to B, most of the time, and it has most of the same features as the first car. This car is different though," Nelson says. "First, it's obviously not a car, but really a retrofitted tractor, and before that, it looks like it has some parts that you would see on a boat. It's both clear

this vehicle has morphed over time from solving one problem to another related problem, and the people who made it didn't have the time or skills to remove signs of its previous incarnations. When switching on the turn signal, the horn sometimes honks, too. Rolling down a window pops open the trunk when you're unlucky. And you still haven't found the emergency brake."

This Franken-car is what you drive when you have no viable alternative. Yes, it should get you from A to B, but don't expect a pleasant ride. This, Nelson says, is what working with a codebase saddled with excessive technical debt is like.

Remember, technical debt doesn't apply just to technology: It applies to processes as well. That's important to remind people of, as you evaluate which tech debt has to go, and which you can live with for now.

"When thinking about modernization, don't focus solely on modernizing the technology," says David Egts, chief technologist, North America public sector for Red Hat. "Think about modernizing the people and processes too. If you only modernize the technology with a faster computer, you may not be addressing underlying reliability and scalability issues forcing the modernization effort in the first place. These issues can be addressed by reskilling the workforce to write cloud-native applications, as well as using agile and DevSecOps practices."

Now let's delve into explaining why people should care about technical debt: What problems accumulate over time?

Why people should care about technical debt

Explaining technical debt to non-technical stakeholders is relatively simple. Explaining why they should care about technical debt may be a trickier challenge, especially in perpetual "do more with less" environments.

Those stakeholders want new applications and features, updates to existing systems, and so forth – and they want them yesterday. This can put good development decisions at odds with delivery speed and other goals, so you make trade-offs or take on debt.

The built-in financial metaphor also conceptually extends to the costs of that technical debt over time, especially if you never repay it. If you're not careful, you're suddenly scrambling just to cover your minimum payments.

"Tech debt builds up as a backlog of 'bills' you need to address as more debt occurs in day-to-day business," Duensing says. "Like monthly minimum payments, it is important to clear out as much as you can on a routine basis before it becomes unsustainable to address with the resources you have on hand."

This is something other leaders in your organization should understand not as a matter of code or design decisions, but as a bottom-line issue.

"When you have too much tech debt, your team ends up spending all of its time repairing shortcuts and can't spend time on new, revenue-building projects," Duensing says.

That's a trade-off even the least technical executive can grasp – and care about.

Are words like "transformation" and "innovation" thrown around regularly in your meetings? Make clear that unchecked technical debt will eventually hamstring both and put you at a competitive disadvantage.

"Technical debt makes building new features that should be easy, hard, and features that are sophisticated, impossible," says Nelson. "This crushes an organization's ability to innovate and leaves them vulnerable to competition. It also leads to poor quality products that are buggy, slow, and often suffer from regressions."

Are customer needs suffering?

Another reason to care: Rampant technical debt will almost inevitably impact your company's ability to be responsive and adaptable to evolving customer needs. Stone from Liberty Mutual offers a useful target to aim for when it comes to technical debt: "We want solutions that can last for the useful life cycle of their capability with the least amount of overhead possible."

"Technical debt makes building new features that should be easy, hard, and features that are sophisticated, impossible"

"People should care about this balance so that they can make the right decisions for their customers when they need it," Stone continues. "Without being aware of the balance, you may find yourself in a situation where you can't respond to your customer needs because you have to pay down too much technical debt to be able to accommodate their otherwise easy iterative enhancement. If you have too many of these scenarios, you will quickly find that your customers will find other solutions that can meet their needs in a timely manner."

The human toll: Talent troubles

Finally, unsustainable technical debt can be both an underlying cause and a symptom of greater problems in your organization – a crummy culture, broken processes, and so forth. This means a technical problem is also very much a people problem.

"It has a significant human toll," says Nelson from Carbon Five. "Organizations that don't take technical debt seriously – and dedicate time to keep it under control – will have a harder time retaining talent, and in the worst cases, a harder time hiring as well."

What causes technical debt – and how to minimize it

By Kevin Casey

How does technical debt accrue in your organization – and how can you keep it to a manageable level? Consider these four strategies.

Technical debt is all but inevitable. In fact, even the headline for this article presumes some level of tech debt in your organization – because unless you work in a magical fantasy land where the code is always pristine and developers never have to make trade-offs, it's true. Most IT leaders and software engineers will agree with this premise to varying degrees.

"Wherever software is written,technical debt is a reality," says Mike Duensing, CTO and EVP of engineering at Skuid.

We recently shared tips on explaining technical debt in plain terms that everyone can understand – an important topic because of that reality – and because IT leaders need to be able to teach others in the organization about why properly managing tech debt matters.

> "Technical debt fundamentally accrues when software design and implementation decisions bump up against – or straight-up collide with – business goals and deadlines.

Today, we're looking at ways to manage your organization's technical debt so that it stays at manageable levels. First, let's be clear about how it accrues.

What causes technical debt

Technical debt fundamentally accrues when software design and implementation decisions bump up against – or straight-up collide with – business goals and deadlines. This is why it exists in virtually every organization: If you always waited until every last line of code was perfect before you deployed, well, your organization probably wouldn't be around for long.

"Technical debt occurs when IT teams need to forgo certain development work – [such as] writing clean code, writing concise documentation, or building clean data sources – to hit a particular business deadline," says Scott Ambler, VP and chief scientist of disciplined agile at Project Management Institute. "It is inevitable and can sometimes make sound business sense, such as when speed-to-market is critical, when resources are limited, or information is incomplete."

At some point, though, you need to pay that debt down. You'll also need to keep things in balance: Too much technical debt, just as with financial debt, can become a crushing problem.

"When the business puts too great an emphasis on deadlines, the team often cuts corners to try to meet them," says Christian Nelson, VP of engineering at Carbon Five. "This might work in the short term, but the growing technical debt will crush productivity before long and require a significant effort to pay down."

A common technical debt example

Here's a common scenario, courtesy of Justin Brodley, VP cloud operations & engineering at Ellie Mae and co-host of The Cloud Pod. Let's say there's a multi-million-dollar deal on the table, or a new multi-million-dollar revenue stream, but it will require a net-new application development.

"It is a normal by-product of software development: the UI/UX team designs the best interface they can. [Then] the development team looks at it and says it will take six months to build, [but] the product management person needs to ship in three months," Brodley says. "What can we take out of the product to get the best UI but without the complexity?"

So the dev, product, and UI/UX folks start taking things out to reduce complexity, but they're still short of that three-month timeline. The next question, naturally: What else can we take out?

"Well, you can remove things like developing automated tests and error handling, [or] make it vertically scale instead of horizontally scale," Brodley says. Then you reach some version of this decision: "We'll fix that in the next sprint."

That's technical debt, and it's not unlike how a homeowner who suddenly needs a new air-conditioning unit in the middle of a sweltering summer might say: "We'll pay for this over the next 12 months instead of all right now."

Is technical debt bad?

This isn't a bad thing, Brodley and others note. Technical debt becomes a bad thing when you don't actually follow through on the "we'll fix this later" promise.

These kinds of tradeoffs aren't the only cause of technical debt. Most technologies are eternally becoming outdated; deferring maintenance or upgrades, for example, is also a source of debt.

Not matching the right people to the project or job also drives technical debt. "Sometimes teams of mostly junior developers are expected to build complex software without the necessary support – [such as] mentoring and guidance," Nelson says. "It's unfair to expect someone with less than a few years of experience to exercise the judgment of a senior developer. It's no surprise that projects like this can accumulate tech debt really quickly."

Nelson adds that technical debt shouldn't be thought of as some heinous crime of software development; it most often accrues for good (or at least understandable) reasons.

"It's important to point out that technical debt isn't really the result of deliberate malfeasance, and rarely incompetence," Nelson says. "Lack of shared values, aggressive business deadlines, and unrealistic expectations are the primary culprits."

Now let's examine four pieces of advice to minimize your ongoing technical debt:

How to minimize technical debt: 4 strategies

"Technical debt is often unavoidable and can even sometimes make good business sense. It's not about avoiding technical debt completely, it's about keeping it within reasonable bounds," says Ambler from PMI. "Much like a financial balance sheet, you need to think of keeping a healthy technical balance sheet – one with a strong debt-to-equity ratio."

Doing so depends on a mix of technical and design practices, culture, prioritization, and education. Here are some things the experts advise in terms of keeping that technical balance sheet in proper alignment.

1. Treat technical debt as a tool, not a scourge

Just as with financial debt, technical debt is more likely to become a problem if you ignore it or otherwise pretend it doesn't exist. Instead, embrace it and treat it like a tool – one that needs ongoing attention and maintenance.

"The first step is to acknowledge that there will always be some technical debt and that can be a healthy thing," says Justin Stone, senior director of secure DevOps platforms at Liberty Mutual Insurance. "Spending time trying to have zero technical debt can also mean you aren't delivering value to your customers quickly enough when they're trying to solve new and innovative problems."

> "The key is to actively manage the key trade-offs that have the most customer value with the least debt."

Stone offers a good rule of thumb for beginning to sort "good debt" – the technical equivalent of a manageable mortgage, for example – and "bad debt" (the technical equivalent of rampant high-interest credit card debt): "The key is to actively manage the key trade-offs that have the most customer value with the least debt." He adds that this kind of assessment will vary even within the same organization.

"How much technical debt is good for a given area or application may differ from another," Stone says. "Most importantly, you have to consider basic lifecycle management practices as a must-do against technical debt. If you make these nonnegotiable, you can then talk about the other tradeoffs and debt paydowns that you have in relation to customer outcomes and not the systems running behind the scenes."

2. Don't let business decisions become a cover story for lazy development

There will always be some tradeoffs between design/development choices and business goals; just don't let them become an excuse for poor or lazy development practices. Well-thought-out technical design should be a priority, says Duensing from Skuid.

"Align your engineering culture around a 'do it right the first time' mentality in order to develop systems with solid architectural foundations," Duensing says."Well-designed, hardened solutions will have less of a downstream impact on customers and operational teams that provide post-deployment support. Time to delivery pressures will always be there, but don't take design shortcuts as an acceleration method."

3. Don't improvise your management plan for tech debt

Count Brodley among those who view technical debt as inevitable. The trick, Brodley says, is to approach it with care and understanding. That means you need to treat it like any other important program or tool in your organization: You need a strategy and a plan for how to execute it.

"Product backlog features to fix tech debt need to be documented, and your product management, engineering, QA, and operational teams should commit to a percentage of tech debt work," Brodley says. "This could be done through targeting refactor sprints if you're an agile shop, or designating a release as [a] maintenance release and focusing on tech debts."

For IT shops that follow site reliability engineering (SRE) practices, Brodley notes that you can use error budgets as a tool for keeping technical debt from spiraling out of control.

"An error budget is a measure of your service when it is not meeting Service Level Indicators," Brodley explains. "When the budget reaches zero, you aren't allowed to deliver [new] features until you stabilize the product and reduce your error budget balance."

4. Give people the time and resources to tackle debt

In addition to prioritizing good documentation and technical practices, you need to ensure you're empowering people to use technical debt in a healthy manner – and pay it down over time. If your developers are under nonstop pressure to deliver new features or products, how will they ever address previous tradeoffs?

"Developers should be expected to – and be given the time and support to – leave the codebase in a better place than they found it," says Nelson from Carbon Five. "Every commit is an opportunity to move the needle in the right direction. And production code should include automated tests, which are run automatically on every change."

"Technical debt is somewhat subjective, so teams should dedicate some time to reading about the qualities of good code, and most importantly, talk to each other so that they form a set of shared consistent ideas about what it means for their project," Nelson says.

Moreover, keep in mind that while there are some common definitions of technical debt, what it actually looks like will vary from organization to organization. There's not some objective truth about what, precisely, constitutes tech debt. This means that you need to give people time to learn the topic and build their own definitions and frameworks.

How to repay technical debt: 4 strategies

By Kevin Casey

Technical debt isn't inherently bad – used properly, it's an effective tool for shipping code faster and more frequently.

However, as with financial debt, you'll eventually run into problems – potentially serious ones – if you ignore your existing balances while continuously borrowing more and more.

"Like financial debt, tech debt is not always bad, but we must be smart about it," says Travis James Fell, product manager at Hypori. "Incur tech debt in light of the business benefit, understand other debt already incurred, and have a plan to pay it off."

That's part of a solid formula for using technical debt wisely: Understand what it is (and how to explain it to others), what causes it, and what benefits you gain from taking it on – the "equity" in the financial-technical metaphor. Then, as Fell says, track your balances and have a plan to pay them off over time.

In a sense, repaying technical debt is simple: It is just a matter of going back and addressing the tradeoffs or compromises you made in a system to achieve a particular goal or benefit, such as meeting a critical deadline. It's when you never actually do this that trouble abounds.

4 ways to repay your technical debt

We've previously addressed what technical debt is, and why it exists in the first place. As described above, the how of repaying technical debt may be fundamentally simple, but it becomes far more complicated when you can't solve the where (it exists) or when (to repay it). Let's look at four strategies for solving this problem.

1. Set "repayment terms" for your technical debt

Financial debt typically comes with a specific payment due date – say, the first of every month for your mortgage payment. Unchecked technical debt, however, will happily saddle a codebase forever if you don't set those repayment terms yourself.

That starts with knowing where debt exists (see also #2 below) – you won't necessarily receive a statement or bill every month. Then set a plan for paying down your balances so they remain manageable.

"Do a full tech debt audit and create a timeline to pay off all that debt with a follow-up process to make sure your company doesn't move forward carrying a high debt ratio," suggests Mike Duensing, CTO and EVP of engineering at Skuid.

2. Listen to your developers – closely

From an IT leadership perspective, you'll need to keep in mind that you might be a degree or three removed from the day-to-day maintenance work (or "debt service") that typically comes with technical debt in a codebase.

This is one way that technical debt can spiral out of control: You simply don't know where it exists, or to what extent. That makes it kind of impossible to address (or "repay") later – so make sure you're asking questions and understand where the debt exists in your organization and its systems.

> Unchecked technical debt will happily saddle a codebase forever.

Moreover, make sure you're listening – not just to the answers when you proactively ask questions, but on a regular basis. In particular, don't be too quick to chalk up the complaints of your development teams as disgruntled grumbling. There might be a pattern in the feedback signaling that it's time to address a particular balance owed.

"If every time they go into an area of code, developers are voicing how they are working around a poor design, this is a warning flag that requires attention," Duensing says.

3. Treat technical debt as part of your overall strategy

The audit that Duensing recommends may be easier to achieve if you plan for it from the start, meaning: Bake technical debt (and its repayment) into your overall software development pipeline and strategy.

Fell notes again that tech debt isn't inherently bad. In fact, it can make great sense in early-stage products or with the minimum viable product (MVP) approach to development. In general, it's a welcome tool in environments under pressure to ship code faster and more frequently (which is to say, almost all of them.)

"Repayment requires an understanding of what tech debt is and where it exists in the product, an estimate of the ongoing maintenance costs, engaging the business on the timing of other investment decisions, and including tech debt payoff on a software product roadmap," Fell says.

Indeed, it should be treated not as a dirty word but as an explicit part of that roadmap.

"I think that tech debt should be a part of well-maintained technical strategy," says Yoseph Radding, a software engineer and co-founder of Shuttl.io. "Instead of planning big features, engineers should work to plan smaller milestones that get the product to market faster, but also plan on how to pay down the technical debt that they create."

4. Find the repayment practices that work best for you

Some teams may find that easier said than done, so let's look at some practical tips for actually addressing technical debt over time. A general theme here is that you'll need to set priorities

based on the criteria that matter most to you and your organization. Here are some options:

· Track the maintenance costs (or "interest") and prioritize accordingly

Conventional financial wisdom says to pay down higher-interest debts first; for example, like a credit card with a high interest rate. You can apply a similar principle to tech debt by tracking and analyzing what it costs you over time.

One strategy that has worked for me is to have a separate task board that tracks how much time, in human hours, [is] spent [maintaining] a particular piece of tech debt," Radding says. "If the [human]-hours result is too high, then that is a piece that we have to tackle immediately. Having the time in hours shows the business how much money is sunk into the problem and [enables you] to prioritize the fix appropriately."

· Consider other prioritization approaches

Scott Ambler, VP and chief scientist of disciplined agile at Project Management Institute, notes there are two common tactics for managing technical debt.

One is to tackle the biggest problems first. "Biggest" could be measured in something like human hours, as in Radding's example above. Ambler also points to error rates as another metric. Regardless, the idea is to address the largest or most urgent debts first, though Ambler notes there's a cost-benefit consideration here.

"You can look at the most pressing debt – e.g., identify where there are the highest error rates or what will have the biggest impact on the business – and address it first," Ambler says. "This can sometimes take longer and is often a more involved process."

There's a flip side to that strategy: Knock out the smaller issues first as a means of achieving faster results.

"This approach offers more immediate results and can quickly show business leaders the value of paying off technical debt – especially if you need their buy-in to support the larger repayment efforts," Ambler says.

· Allocate specific amounts of development time for repayment

"We'll deal with that later" is not a plan. Bank scheduled time and resources to manage technical debt. Agile teams (or any team that applies agile or DevOps principles to their work, really) can create user stories for tech debt management, for example. Regardless, the general idea is to take whatever methodologies or practices your team works with and apply them to technical debt appropriately.

"A best practice is to allocate stories for debt remediation in each sprint," Duensing notes. "This ensures you will continue to make incremental progress."

Incremental progress – rather than zero debt – is usually the right goal. "Zero progress," on the other hand, is not advisable...

"With mismanagement, tech debt can cause your development team to fall apart," Radding says. "Tech debt management is an essential part of engineering."

How to reduce technical debt: 5 tips

By Stephanie Overby

What is technical debt costing your organization? Let's examine strategies for measuring – and reducing - your reliance on legacy technology and processes

When will the day of reckoning arrive for your organization's technical debt?

> Technical debt is now such a sizeable off-balance sheet liability that it must be accounted for to the business

While the CEO may be focused on shiny new technologies, most IT organizations still live with many legacy systems and processes that are draining, if not dangerous – not only in terms of mounting costs but also increased risk exposure. What's more, many of these software and infrastructure situations can stand in the way of opportunities for digital transformation.

In lean years with tight funding, IT may have ignored issues or applied clunky fixes in infrastructure and application maintenance – and the impact of those fixes can remain hidden for years. However, as these systems underperform, stop working altogether, or otherwise stand in the way of company goals, it's becoming clear to IT leaders that this so-called technical debt needs to be addressed sooner rather than later.

Indeed, in many organizations, technical debt is now such a sizeable off-balance sheet liability that it must be accounted for to the business, says Dion Hinchcliffe, vice president and principal analyst at Constellation Research. Veteran CIO and current chief digital officer and CTO for senior living provider Affinitas Life Wayne Sadin has gone so far as to describe the scale of technical debt in most companies as a larger liability than that which lurked in the financial statements of Enron or Worldcom two decades ago.

What's holding you back?

It's tricky to convince business leaders or board members to pay down technical debt when they're more focused on what's next than what's holding the organization back.

However, CIOs can harness that desire for transformation to illuminate the technical debt issue. A legacy of short-sighted choices is standing in the way of organizations moving toward

the future. Some of this aging hardware and software proves unfit as a foundation for modern business systems and processes. Building real-time mobile apps on top of decades-old architecture requires, at minimum, some significant remediation. In addition, some companies are spending the bulk of their budgets keeping these neglected IT systems up and running, leaving little left over for innovation.

The solution is to account for technical debt – locating it, categorizing it, and determining the cost of eliminating or reducing it. It's not an easy – or even perfect – process, but it can be done. There are ways to calculate not only the expense but also the potential benefits of addressing technical debt to align investments with digital business goals.

5 steps to reduce and manage technical debt

Sadin shared five steps CIOs can take to begin measuring and managing their technical debt load:

1. Conduct an inventory of the IT stack

This is the first thing Sadin does when taking on a new IT leadership position: Account for every piece of infrastructure and application in the environment in order to get the lay of the land.

2. Calculate the costs of updating systems

Some aspects of technical debt are relatively easy to determine, such as the cost of bringing a general ledger suite up to date. Custom-built software can be more difficult to quantify, but IT staff or auditors can help you understand the issues. In some cases, IT leaders may determine that it makes more sense to keep the debt on the books if a certain system is associated with a business unit that is going to be divested or the cost of remediation outweighs the benefits.

3. Play the risk card

Technical debt significantly increases the risk of a cyber breach or attack. The good news is that most executive teams and board members are now acutely aware of the importance of mitigating cyber risk, and CIOs can tie the issue of technical debt to this critical business priority.

4. Dig into opportunity costs

"The more insidious problem, however, is not that risk which knocks you in the head," says Sadin, "it's the things that technical debt prevents." Quantifying the progress that technical debt could prevent is harder to do.

However, IT leaders who understand future business direction can conduct gap analyses to help put some numbers around this issue. If a company wants new IoT functionality, for example, the IT leader can work with those in architecture to quantify the cost of upgrading foundational systems to support that strategy. If there is no clarity around the business's future technology demands, IT leaders can conduct research into trends in adjacent industries, to project what some of those might be and how well suited the IT environment is to support those emerging technology options.

5. Have the difficult conversations

This is perhaps the hardest part, because few executive teams or boards will want to hear that their plans for a new virtual reality function will require an additional multi-million-dollar investment, or that they need to make a significant investment to bring the ERP suite to the current release.

However, having the hard numbers makes it at least easier to understand. "It's an uncomfortable discussion and the board is going to want to know how we got here," Sadin says. "But it needs to be discussed at the C-suite and board level, and the discussion should be focused on risk and opportunity."