White Paper: **August 2022**

Developer Toil: The Hidden Tech Debt

How to use the Developer Toil Audit to speed up software release cycles, make developers more productive, and increase the business value of your apps.

By Susie Forbath, Tyson McNulty, and Michael Coté

Table of contents

Introduction

Anything you can do to speed up your software release cycle will improve the quality, resilience, and business value of your software. In this paper, we describe one approach organizations use to speed up this release cycle, the Developer Toil Audit. This is a systematic process for finding, valuing, and prioritizing fixing waste in your software process.

First, the process finds wasted time and process debt in how you build and release software. Second, the process then helps you justify delaying working on features in favor of eliminating and automating your software creation process. The result is speeding up your software release cycle. The more frequently you can change and release software—even with just small changes—the more opportunities you have to learn what works and put those features in front of customers, employees, and other users.

Developer toil is a type of tech debt. Tech debt is a broader concept that encompases all of the short-term shortcuts and maintenance compromises made in favor of shipping features now instead of spending time fixing, or "paying down," the debt. For example, instead of updating an API framework you depend on to pull in new data used for insurance adjusting, you spend that time to add multi-file upload to the insurance claims process. Most people focus on this part of tech debt: neglecting your app's code. But we find that focusing on another type of tech debt—developer toil—has equal benefit, sometimes more in the early stages of a company's efforts to improve its software process[1].

This paper describes the overall Developer Toil Audit with examples and how to get started. First, let's spend a brief time looking at why faster release cycles are one of the best ways to improve how you create your software and, thus, how your organization functions.

We've developed this method and process at VMware Tanzu Labs after helping hundreds of organizations improve their software development and delivery process. We still use it today as part of a broader technical debt assessment methodology. If you're interested in finding out more after reading this overview, don't hesitate to contact us.

1. In the Kubernetes world, as of the Spring of 2022 at least, the phrase "developer experience" or "DevX" is being used more frequently to describe, well, the day-to-day experience developers have creating their software. Of course, you want that experience to be good, which increases developer productivity and happiness. Addressing developer toil is one the best and easiest ways to improve DevX.

Faster release cycles create better software

You're probably tired of hearing that "software is eating the world." We know we are! In the more than 10 years since this aphoristic piece of strategy was published, it's come to mean that successful organizations are relying on software as their primary "storefront" with customers and how they optimize internal operations. That is, software helps them run their business better. Thus, for these organizations, improving how they do software directly improves how their business runs.

The benefits of more frequent releases are both improved quality and business outcomes. "Quality" here means fewer problems in the software through the ability to quickly resolve errors, security problems, and performance issues. "Business outcomes" is just fancy business-think language. People also use the phrase "business value." Both phrases mean that your software helps run your organization and achieve whatever goals or mission you have, be that profit, services to citizens, or execution of a mission. These capabilities and outcomes make your software, and therefore your business, more resilient and successful[2].

Despite the benefits, many organizations are held back from more frequent releases. One survey found that, for example, 48% of executives say they haven't changed their applications in a year or more[3]. This matches our anecdotal evidence over the years. What we've found is that these organizations are dealing with a high amount of developer toil and technical debt. People working on software spend too much time on processes that could be automated or even eliminated. Ironically, this toil builds up over time as people prioritize shipping features rather than addressing the problems in their overall software process. Although they're able to change their software quickly at first, just like debt in real life, if this toil and tech debt are neglected, they take over and consume the organization's ability to grow. This results in long, infrequent release cycles. A feature that seemed simple and once just took "15 minutes" now takes weeks, or even months to get in front of users.

Well, enough with telling you that there's a problem. You'll know if you're suffering from the legacy trap and have a sense of being overburdened with tech debt. Let's get on to discussing the methods we've seen organizations use to address developer toil and speed up release cycles.

2. For a deeper study on this, see the book Accelerate: The Science of Lean Software and DevOps.

3. "Improving Customer Experience And Revenue Starts With The App Portfolio," Forrester Consulting, March, 2020.

The Developer Toil Audit method

After working with hundreds of organizations who've modernized their application portfolio, we've created the following technique to find and reduce developer toil:

1. **Audit:** We use a questionnaire about common developer activities and practices. Over the many audits we've done, we've come up with a starter set of questions, but you should also include questions specific to your organization's maturity and systems. This questionnaire of course must be easy to fill out and only done when needed—otherwise it becomes a type of toil itself!
2. **Act:** Based on the findings from the audit, developers and product managers work together to value and then prioritize the work needed to address developer toil. Deciding to delay shipping app features is a difficult decision, so we use a quantitative model that links reducing development toil to increasing business value.
3. **Repeat:** Once the work to address the most painful areas of developer toil is done, repeat the process again. After each cycle of improvement, the questionnaire is completed again to check for improvements and find new toil.

Like all self-help advice, these steps seem simple at first. But, their simplicity is more about usability: complex, powerful processes have been made simple so that they work.

The other thing to keep in mind is the first principle of all improvement schemes: you actually need to do it, and keep doing it. Developer toil is like weeds: you have to garden consistently, not just once.

Case study

What does being buried by tech debt look like? Here's an example from a VMware Tanzu Labs client engagement:

The product team we worked with was struggling to deliver features quickly, at one point taking an average of over 400 hours to complete a feature!

In late September of 2021, we introduced the team to Developer Toil Audits. Using this process, we refocused the team to prioritize fixing the develope toil that mattered the most for their problems. In this case, our goal was tohelp them reduce their story cycle time, the time it takes from idea to working software.

After the developer toil audit, they moved from delivering a story approximately every two weeks to delivering a story daily!

Understanding tech debt and developer toil

Before you can start fixing your developer toil problem, you have to find it. And to find it, you have to know what you're looking for. So, let's first look at a more detailed definition of developer toil and then go over how to find it.

Identifying tech debt

When it comes to software, the concept of "debt" helps us explain deficiencies in internal quality and the associated impact on the software features, the business outcomes, and the ability to change in the future. For example, using a credit card allows you to buy something now, but includes the possibility of paying more for it later if you don't pay your credit card bill on time. This same pattern applies to thinking about tech debt.

"Debt" is a great metaphor for software because it describes the impact of short-term gains on long-term quality and optionality. With software, a team can take on tech debt to deliver features with less time and effort now, deferring costs to the future. These "costs" can come in several types. Here are a few examples:

1. **Compromises made in the code to ship faster:** Instead of doing the work to make the code easier to configure and add to in the future, developers can "hard code" parts of configuration making it hard to change the code in the future. Historically, the time it takes many apps to create a mobile version of their web interface, even by simply "reskinning" the UI, exposed compromises in many code bases. The deadliest form of this type of tech debt are compromises made in security.
2. **Using older, even unsupported versions of third-party and open source software, as well as platforms and systems:** Instead of spending time to update the database, teams may decide to ship features. This type of debt also includes decisions to keep using older systems like mainframes instead of doing the work (if appropriate!) to move to new systems.
3. **Developer toil:** Teams can decide to put up with manual processes and other types of "toil," as we define below, in favor of shipping software. For example, complying with architectural review boards may require a developer day's worth of time to fill out a spreadsheet. Instead of spending the four days to automate this process, teams may decide to ship features instead.

This paper focuses on the third type of tech debt described above, developer toil. For discussion and techniques to address other types of tech debt, check systems like the Swift Method and our upcoming book, *Escaping the Legacy Trap*.

Although fixing other types of tech debt is critical, we believe that addressing developer toil is one of the most effective ways to increase developer productivity. But, at the same time, it's been one of the most neglected types of tech debt in recent years. So, let's take a look at what exactly developer toil is and then, finally, go over how you can pay down that type of tech debt.

What is developer toil?

With the phrase "developer toil," we're borrowing the idea of "toil" from a new operations school of thought called Site Reliability Engineering (SRE). Google's original SRE definition is focused on operations staff that build and run the platforms used to run software in production[4]. We adapt this definition to developers as:

Developer toil is the repetitive, predictable, constant stream of tasks that support adding features to software, but don't actually directly decide what those features are or write the code to create them.

You could fancy this up by saying "any activities that don't directly create business value," but who has the tolerance for that kind of biz jargon anymore?

Let's look at a story of toil to give you an idea of what toil smells like.

Story time: "It'll just take 15 minutes."

You walk into an app planning meeting, and developers are having a conversation with their product manager on estimates for a new feature. They work at an insurance company, The Mid-Eastern Warm Smiles Insurance Company, "Smiles'' for short. The company is growing by introducing 24-hour insurance products: so if you want to learn how to skateboard today, but you work in the gig economy, you'll want to buy some insurance immediately in case you break an ankle.

Things are going well, Wall Street likes Smiles' promise to grow revenues by adding new insurance lines to its 150-year-old business of traditional insurance. Right now, the actuaries need to incorporate a new stream of data into their pricing engine: a good dataset going over skateboard injury rates per demographic and geography. Thankfully, the International Skateboarding and Paddleboarding Society has been cataloging this information since 1956, so there's an API to get this information. Integrating that information into the actuary's app requires some slight data reforming.

"Adding a field to list that?" a developer says. "No problem, that's, like, 15 minutes of work. An hour at most if I have to refill my coffee while doing it or attend an HR meeting."

And, sure, it may actually be the case that once the developer puts their fingers on the keyboard and starts typing, it only takes an hour. But, someone else chimes in…

"Oh, except we'll need to handle state changes," they say. "Remember when we pulled in data about the sunlight fading effects on paper for the Pokémon card game insurance product? That was supposed to just take 15 minutes, too…but it ended up taking a week."

4. Their original definition of toil is focused on operations staff that support running applications in production, so it's focused on service support: "the repetitive, predictable, constant stream of tasks related to maintaining a service."

“And we’ll need to migrate the database,” someone says from the back of the room. The developers nod in agreement.

“You’re right…two days, then?” says the original developer, “assuming there’s no mandatory ergonomics assessment and training I need to take in those two days.”

Everyone is about to stand up to go get some coffee, but another person adds, “Oh, and we’ll need to get security and compliance to review the changes. I mean, we’re integrating in third-party data and attaching to a third-party system.” They pause and add, “and we’ll probably need to talk with the networking team as well.”

“Time to break out the spreadsheets!” they add, getting a laugh from the room.

“Also, we haven’t updated our database yet. I don’t think version 7.05G has the ability to combine this skateboard dataset with our existing dataset,” adds someone else.

“But…it should have just been 15 minutes,” the developer says, realizing they’ll have to skip a lunch break and eat a frozen burrito at their desk again. (Not to mention complete that ergonomics training they’re three weeks overdue on.)

Developer Toil Audit

The first step to uncovering developer toil is to ask about people’s struggles, frustrations, and even boredom. The Developer Toil Audit uses a survey to do this. As you’ll see, the survey questions span the end-to-end process of building and running software. That focus lets us get beyond the “it’ll just take 15 minutes” myopia in the above story. Except for a few free-form questions, respondents answer each question on the Likert scale of 1 to 5, with 1 being “disagree strongly” and 5 “agree strongly.” Without further adieu, let’s look at the survey prompts:

1. What are we making?

2. We have a strong vision for our product, and we’re doing important work together every day to fulfill that vision.

3. I have the context I need to confidently make changes while I’m working.

4. I’m proud of the work I have delivered so far for our product.

5. I’m learning things that I look forward to applying to future products.

6. My workstation seems to disappear from under me while I’m working.

7. It’s easy to get my workstation into the state I need to develop our product.

8. What aspect of our workstation setup is painful?

9. What could we do to make it less painful?

10. It’s easy to run our software on my workstation while I’m developing it.

11. I can boot our software up into the state I need with minimal effort.

12. What aspect of running our software locally is painful? What could we do to make it less painful?

13. It's easy to run our test suites and to author new ones.

14. Tests are a stable, reliable, seamless part of my workflow.

15. Test failures give me the feedback I need on the code I'm writing.

16. What aspect of writing, reading, and navigating tests is painful?

17. What could we do to make it less painful?

18. Our code does pretty much what I expect it to do upon reading it. I can track down code I want to modify with minimal guesswork.

19. Our code has good modularity; I can make targeted, contained changes.

20. What aspect of writing, reading, and navigating code is painful?

21. What could we do to make it less painful?

22. We collaborate well with the teams whose software we integrate with.

23. When necessary, it's within my power to request timely changes from other teams.

24. I have the resources I need to test and code confidently against other teams' integration points.

25. What aspect of integrating with other teams is painful? What could we do to make integrating with other teams less painful?

26. I'm rarely impacted by breaking changes from other tracks of work.

27. We almost always catch broken tests and code before they're merged.

28. What aspect of committing changes is painful? What could we do to make it less painful?

29. Our release process (CI/CD) from source control to our story acceptance environment is fully automated.

30. If the release process (CI/CD) fails, I'm confident something is truly wrong, and I know I'll be able to track down the problem.

31. What aspect of our release process (CI/CD) is painful?

32. What could we do to make it less painful?

33. Our team releases new versions of our software as often as the business needs us to.

34. We're meeting our service-level agreements with minimal unplanned work.

35. When something is wrong in production, we reproduce and solve the problem in a lower environment.

36. What aspect of production support is painful? What could we do to make it less painful?

These are general questions that are a good start and will likely fit for most organizations. We advise spending time to come up with some questions that are specific to your organization, and even remove questions that don't apply. For example, highly regulated organizations should ask about tasks involving compliance.

Quantifying the results

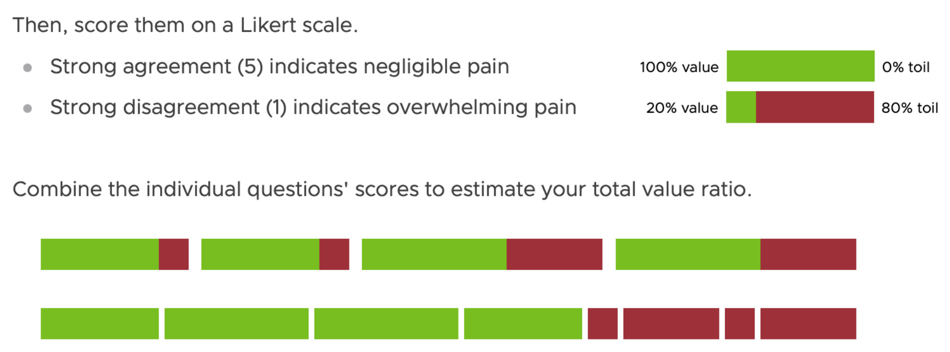
Once the survey is completed, you can do some quick analysis to locate and prioritize developer toil. Because we used a Likert scale, relative agreement or disagreement with the question will represent the degree of toil present. In broad strokes, full agreement (5) means we're close to the ideal: you're toil free. A zero would mean complete blockage: 100% toil, nothing ever gets done. Everything else is somewhere in between.

For example, let's look at the question, "I have the resources I need to test and code confidently against other teams' integration points." Integrating with other teams is critical because it means, for example, that multiple teams can use APIs like the skateboard injury information in the Warm Smiles story. Developer toil is just as often about infrastructure integration, often integrating with security systems. Indeed, in one of our surveys on Kubernetes usage, 42% of respondents said that integrating new technologies with existing systems is an impediment to developer productivity.

To find, quantify, and, thus, rank developer toil, add up the Likert answers for each question. In our example below, we have answers of 4, 4, 3, and 3. You can sum these or average them to find out the degree of toil per question. With that, you have an estimate of total value to total effort expended. This all can be, of course, automated in the form tool you use: it's really just simple spreadsheet work.

How to quantiy the results

Sample scoring


Then, score them on a Likert scale.

- Strong agreement (5) indicates negligible pain
- Strong disagreement (1) indicates overwhelming pain

Combine the individual questions' scores to estimate your total value ratio.

Creating, running, and analyzing this survey should not become a new type of toil. You can create and run this survey easily with Google Forms, Microsoft Forms in Office 365, whatever internal survey tool you use, or even paper if you like the smell of crisp copy paper. Choose a survey tool that's quick and easy to use. We have links to a starter survey and spreadsheet form tool that you can clone in the Appendix.

Once the survey and quick analysis is done, you now have a single metric for each type of developer toil. And by combining all the questions, you now have metrics to track overall developer toil. If you're the type of person who likes dashboards and gauges, you can go wild with some new visualizations.

By quantifying each type of your developer toil, you're now closer to making decisions about which toil to address.

Mapping to business value

A simple ranking of developer toil is better than nothing. However, before deciding which developer toil to address, we recommend linking each type of developer toil to the business value created by fixing the toil. Put briefly, the value you get will be related to the time saved, and, thus, the ability to ship more features in the future. By addressing developer toil, you're fainting business agility.

For example, it may only take developers a day to fill out the spreadsheet for compliance, two days waiting for a response, and then maybe another day to make changes for failed audits, then half a day to complete the spreadsheet again, and…well, let's just say you should budget two developer days for compliance audits. It may take five developer days to meet with auditors and automate this process, reducing the amount of time it takes to pass audits to just half a day. In any given release, two days instead of five days is an obvious choice: you'll choose to keep the manual audit process to gain three extra days each release to ship more features.

Once you think about fixing toil as increasing business agility and potential, you can see that fixing developer toil creates business value! Another outcome of addressing developer toil is plain old productivity increases, which can be tracked as time saved, and thus, time-as-money saved. Fixing developer toil will help you grow your business and also reduce costs.

Strategically fixing developer toil

To start deciding which types of developer toil will have similar payoffs, find the top ten types of developer toil from your surveys, and spend some time to estimate the business agility gains you'll get if each is fixed. You'll also need to get a rough estimate of the time it takes to fix each one. Don't worry about this being perfect. Between developers and product managers, you'll get a perfect enough sense to start making decisions.

1. Although it's a "soft benefit," another important outcome is increased staff morale. Stronger employee hiring ability and retention are, or should be, a strategic imperative for any organization that depends on software. Morale also increases software quality: happier people make better software.

At this point, you should have a pretty good idea of which developer toil to fix. The last step is to do the usual product management prioritizing to weigh fixing these items with other tasks you could do. These are strategic decisions that product managers need to make. To fix toil, you need to stop shipping features. You need to slow down the business. At the start, before product managers have been empowered to make these kinds of decisions, higher level management will also need to be involved.

This a high-level overview of how to value and then select developer toil to work on. This mindset is important for addressing the short-term thinking that created developer toil in the first place. Next, let's look at one way we've worked with organizations to implement and run this process: chore stories.

Chore stories

We have a special name for stories that do not deliver business value: chores. Chores don't have observable consequences for the user, and therefore cannot be accepted by a product manager that does not value fixing developer toil as increasing business agility. Some common examples of chores you might find on a product team include fixing flaky tests, setting up workstations, and planned refactoring. These are exactly the kind of things that might come out of a developer toil audit. Because we've assigned value to many types of developer toil based on increased capacity and improved productivity, we can start to weigh fixing the toil against shipping a feature. The product manager can now make strategic decisions about fixing developer toil.

To create stories for developer toil, we recommend the following format:

> **To improve** <tech debt categories>
>
> **We should** <do activity>
>
> **So that** <impact>

Here are two examples of this format:

> **To improve** our release process and committing changes
>
> **We should** add linting to our CI pipeline
>
> **So that** we catch code style errors before they're integrated

> **To improve** production support
>
> **We should** run multiple instances of our app in acceptance
>
> **So that** we can discover session persistence problems earlier

Product managers may also find it helpful to identify who the direct "customer" is for each story. If you think of the customer as the beneficiary of the story, in many cases, the customers are developers. The customers for chore stories may also be security and compliance staff, or even operations staff. Or, you could just leave it in the passive voice and assume the customer is something akin to "the greater good."

This story format helps product managers understand:

- Which specific type of developer toil we think will be improved by completing this chore (whether it's a cluster of toils, or just one)
- The specific activity that needs to be achieved to have the expected improvement
- The impact we think this activity will have on our product

As a product, this provides the necessary information to prioritize chore stories against feature stories and bugs. This format and the analysis from the developer toil audit will also justify the importance of this work to business stakeholders.

To get you thinking about more chore stories, here are some common examples we've encountered:

Automating code hygiene: If your analysis determines that the most painful part of your development practice is your release process, you might hypothesize that adding linting to your CI pipeline will help you catch code style errors before they're integrated.

Implement a secure software supply chain: A more involved, but higher value hypothesis might be that you could automate many of the manual and time-consuming security and compliance checks by creating a secure software supply chain with a tool like Cartographer and the Accelerators in VMware Tanzu Application Platform.

Matching developer labs to production environments: If you have low scores in production support, you might expect that running multiple instances of your app in acceptance will allow you to discover session persistence problems earlier.

Strategic considerations for chore stories

As with most strategic thinking, your organization and context will drive how you strategically create and consider each chore story. The insufferable old maxim, "it depends…," applies here. Nonetheless, here are some approaches product managers can use when thinking through which chore stories to work on.

Consider the product lifecycle: You should balance addressing tech debt and delivering features in each iteration. But the balance depends on where you are in your product lifecycle and the outcomes you're trying to achieve. If you want to get fast feedback and learn if there's a market for your product idea, you might decide to tackle tech debt later on, once you establish product/market fit. For later stage products, production support becomes all the more critical, and so you might want to prioritize robustness in this category.

Empathize with your developers: Ask to sit side by side with your developers for part of the day to better understand their workflow. Think of it like customer research: what are their pain points? How can you help solve them while still addressing the business need?

Emphasize outcomes over implementation: One of the strongest tools in a product manager's toolkit is their ability to focus a team around an outcome rather than a solution. Product managers already do this when nudging the team away from discussing implementation details while pointing feature stories. Similarly, we can leverage this skill when writing chores to help the team focus on fixing the problem at hand rather than implementing "cool technology X."

Keep chores small in scope: Another product manager superpower is keeping stories small in scope: writing the smallest feature that delivers a complete piece of value. This helps you prioritize more nimbly and validate assumptions sooner. The same line of thinking should be applied to chores. What's the smallest activity your team can do to impact your development practice? You'll learn if you're going in the right direction before dedicating more time to addressing this tech debt category. If priorities change, you'll be able to shift direction more easily.

Removing developer toil increases developer productivity

Through our experience as consultants working with product teams in different industries, we've used Developer Toil Audits to focus and motivate product managers and developers to fix toil and speed up their release cycles, and thus, improve how their organizations build software. This directly improves the business by adding more capabilities and capacity to deliver more features, even in the short term.

Hopefully we've helped you understand the value of finding and fixing developer toil. Each time you fix tech debt, you gain more capacity and capabilities to create new features and improve your software. This is how you use a modern software culture to increase business agility. Or, put more simply: less developer toil leads to better software, and better software leads to better business.

To get started, take a look at the sample survey and customize it to fit your organization. If you'd like help, you can request a free consultation with a product management leaders from Tanzu Labs.

Appendix

How to use the Value Ratio Calculator to quantify developer toil

This section provides instructions on how to use our sample Value Ratio Calculator tool to quantify the impact of paying down tech debt and reducing developer toil.

If you've skipped ahead to this point, we recommend reading about the overall approach and the starter set of survey questions provided in this white paper.

After conducting a survey, product managers and developers can work together to quantify and prioritize work based on what will have the greatest impact on developer toil. For example, if your team is deciding whether to delay shipping app features to address tech debt, you can use this quantitative model to link the benefits between eliminating tech debt and toil with increasing business value.

Orientation to the Value Ratio Calculator

To get started, first orient yourself to the tool. The Value Ratio Calculator tab of the spreadsheet contains formulas that calculate the point value of different activities related to developer toil and tech debt.

The "Current Scores" section at the top of the sheet will auto-populate with values from the survey form responses.

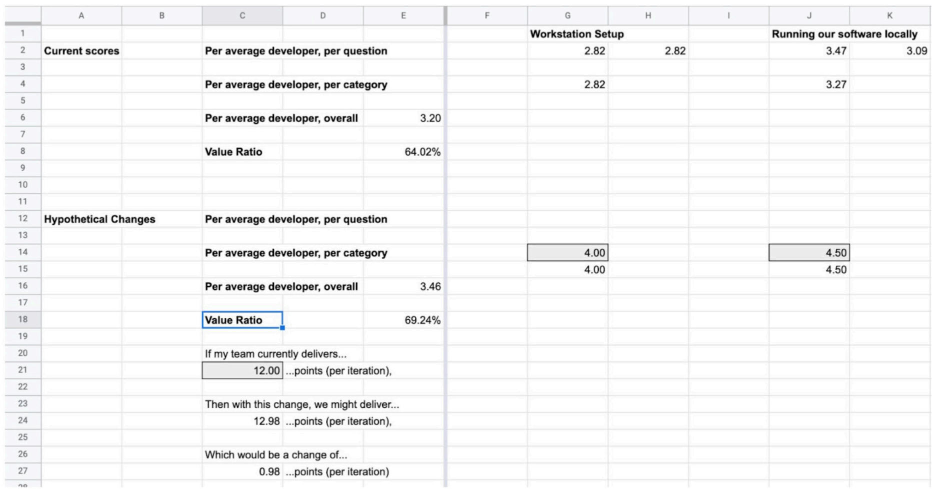
Next is the "Hypothetical Changes" section. Row 14 of the spreadsheet contains gray cells where a user may enter a hypothetical score for each topic under consideration. Changing the values in these boxes adjusts the outputs on the far left of the spreadsheet. This displays the number of points a team could hypothetically deliver per iteration and corresponding value ratio, based on the delta of the current scores and hypothetical changes.

For example:

The first two topics listed in our example survey are "Workstation Setup" and "Running Software Locally."

We can calculate the value of improving one or both of these areas by entering hypothetical scores for each in the corresponding gray boxes in row 14. Then, review the corresponding changes in the value ratio and number of points delivered per iteration.

The image below shows the Value Ratio Calculator tool. In this example, the current scores for “Workstation Setup” per average developer per category is 2.82. The current score for “Running our software locally” is 3.27. If the hypothetical “Workstation setup” score is changed to 4 and the hypothetical “Running our software locally” score is changed to 4.5, the value ratio is 69.24% with a 0.98 increase in points delivered per iteration (from 12 to 12.98).



	A	B	C	D	E	F	G	H	I	J	K
1							**Workstation Setup**			**Running our software locally**	
2	**Current scores**		**Per average developer, per question**				2.82	2.82		3.47	3.09
3											
4			**Per average developer, per category**				2.82			3.27	
5											
6			**Per average developer, overall**		3.20						
7											
8			**Value Ratio**		64.02%						
9											
10											
11											
12	**Hypothetical Changes**		**Per average developer, per question**								
13											
14			**Per average developer, per category**				4.00			4.50	
15							4.00			4.50	
16			**Per average developer, overall**		3.46						
17											
18			**Value Ratio**		69.24%						
19											
20			If my team currently delivers...								
21			12.00	...points (per iteration),							
22											
23			Then with this change, we might deliver...								
24			12.98	...points (per iteration),							
25											
26			Which would be a change of...								
27			0.98	...points (per iteration)							

Even without conducting a survey, product managers can also use this tool to estimate the impact of paying down different types of tech debt—or work backwards from a desired outcome to help determine how much investment is needed in different areas and in various combinations.

We suggest experimenting with the tool and the survey, customizing and refining your questions and topics to best suit the specific toil and tech debt tradeoffs facing your team.