

Logo

Description automatically generated

Final Project Report

THE FEDERAL AVIATION ADMINISTRATION: TECHNICAL DEBT

CASEY, JAMES Q. | DEREBEEW, DEAN B. | DETAR, DRAKE D. | DOWHOWER, ANNA E. | RIECKE, DANIEL A. | WONG, ELIEZER E.

November 18, 2021

**Table Of Contents:**

**Executive Summary** pg. 3

**Introduction:** pg. 4

* Project Statement pg. 4
* Overview pg. 4

**Section I: Current Situation Analysis** pg. 6

* Problem Statement pg. 6
* Constraints to be considered pg. 7

**Section II: Alternatives, Evaluation and Recommendations** pg. 8

* General Estimates of Cost and Merit pg. 8
* Comparison of Alternatives pg. 8
* Recommended Alternative pg. 8

**Section III: Findings** pg. 10

* Definition and Ways to Identify Technical Debt pg. 10
* How Other Organizations Address and Handle Technical Debt pg. 13
* Calculating Technical Debt pg. 15
* Preventative Measures and how to manage Technical Debt pg. 16

**Section IV: Software Solutions** pg. 21

* Gartner pg. 21
* Third Party Solutions pg. 22

**Appendix A: Signed Statement of Work** pg. 24

**Appendix B: Documentation, and Research** pg. 27

* Client Register pg..27
* Communications Plan pg. 27
* Risk Register pg. 28
* Home Depot Software Engineer Interview Notes pg. 29
* Planned Project Schedule pg. 32
* Annotated Bibliography of conducted research pg. 37

**Technical Debt**

**Final report**

**Executive Summary**

**Prepared by The University of Oklahoma Field Project Team 1**

**11/16/2021**

The FAA has discovered three main types of technical debt. Those being deliberate, accidental or outdated, and Bit Rot. Deliberate technical debt is usually incurred by business decisions to help reduce the time to market for a product. Accidental or outdated refers to the technical debt that is incurred through changing requirements and design decisions that cause the original design to be flawed. This may require significant refactoring, unless the application is updated about every year to clean up code that makes it more difficult to implement new features. The last term mentioned for technical debt, Bit Rot, represents bad code that is developed over time to become overly complex. This happens when organizations have multiple developers working on the same project who may not understand the original design of the application. Individual developers are the most responsible for this type of technical debt. The terms described above serve as the foundation for our research in finding ways to identify, calculate and manage technical debt. In terms of calculating technical debt, the team found this formula: Technical Debt Ratio = (Remediation Costs / Development Cost) \* 100. Remediation costs refer to the time it takes to refactor the necessary code, while development costs relate to the costs associated with developing the new software. This may be difficult to do manually, thus our team has identified a tool that can aid the calculation of your technical debt ratio, SonarQube. One of our researchers conducted an interview with a Senior Software Engineer for Home Depot to gain further insight into how larger companies manage their technical debt. The information gained from this interview provided the team with a set of questions to ask before refactoring code. This helps manage outdated or accidental technical debt once it has been identified. Following that, the next section highlights the best practice approaches to tackling technical debt within an organization per IS experts. Some of these includes getting all team members on board and having them understand the importance of reducing technical debt, including proper documentation, conducting code reviews, and using agile architecture. Finally, we included a table of potential software tools with their prices and description at the end which could help in creating a system to manage technical debt.

**Introduction**

The FAA is a governmental organization with numerous responsibilities concerning civil aviation and the U.S. commercial airspace. With their many applications, they would like to analyze and evaluate technical debt. So far, the FAA has discovered three main forms of technical debt, namely, Deliberate, Accidental or Outdated, and Bit Rot. Moving forward, they seek to potentially identify new forms of technical debt and find a way to measure it. On that tangent, the FAA would also like to weigh out the effects of their technical debt and find a way to reduce it if it poses an issue in operations.

**Project Statement**

The FAA is interested in figuring out how to manage their technical debt and would like the team to find ways where they can measure it and compare the debt to other similar organizations, giving the FAA a baseline to figure out if their technical debt hinders their operations. This project will have the team investigate their three main forms of technical debt by studying other organizations and their methods employed to deal with it. Furthermore, the team plans to explore existing third-party solutions as well as industry best practices. In doing so, the team plans to deliver these findings and a justified recommendation that the FAA could implement to alleviate the ambiguity of technical debt and how to manage it.

**Overview**

This report will consist of four sections, namely, the **Current Situation Analysis**, **Alternative Evaluation and Recommendation, Findings, and Software Solutions**. The Current Situation Analysis deals with the FAA’s reason for conducting this project and serves to discuss the following key points:

* The issue occurring within the organization that is prompting the completion of this project
* The core requirements that determine a successful completion of the project
* The purpose of the completed project and how it would be instrumental to resolve the issue
* The constraints to be considered during the project

The second section, Alternative Evaluation and Recommendation, serves to identify approaches that the FAA could potentially implement to address the key problem while accomplishing their defined goals. This section intends to tackle the following:

* How the completion of the proposed project will address the FAA’s goals
* What are the alternatives available to the FAA to solve their problem
* What are the benefits and drawbacks associated with each alternative

The third section, Findings, is intended to communicate all the research done by the team that explains how to:

* Define technical debt and its related terms
* Identify ways to identify the different types of technical debt
* Gain insight on how other organizations manage technical debt
* Ways to calculate technical debt and software solutions to help do so
* Systems expert’s best practices to effectively manage or prevent technical debt

The fourth section, Software Solutions, focuses on the different software solutions available within the industry. This section includes:

* A table of different software available to help manage technical debt
* A description of what type of technical debt each software identifies
* The cost associated with utilizing the different types of software

**Section I: Current Situation Analysis**

Due to the FAA’s many applications and large scope of responsibilities, operations incur technical debt. Currently, the FAA has identified three main forms by which technical debt can be categorized by: Deliberate, Accidental/Outdated, and Bit Rot. Deliberate debt occurs when the organization intentionally takes on technical debt when prioritizing speed. This usually happens in the form of imperfect code whereby a better method might exist, but issues are not optimized due to limitations in time. Accidental or Outdated debt is incurred when deliberate technical debt is accumulated overtime or the use of older systems are no longer supported by vendors. Lastly, Bit Rot is incurred unintentionally as programmers work on code in their own unique ways that lead to spaghetti code, or when systems are transitioned out. Over time, these technical debts could accumulate and decrease code agility as systems mature.

**Problem Statement**

At this juncture, the FAA believes that they can identify their technical debt and is confident that it falls in the categories identified above. Despite that, they are invested in finding out if their technical debt translates to performance inhibition. To do so, they would like the team to accomplish the following:

* Firstly, investigate ways by which they can measure the level of their technical debt to determine if it is acceptable.
* Next, research and apply the measure to other organizations — government entities and private industry leaders — of similar size and technical debt. This would give the FAA a baseline to compare themselves to and determine if their level of technical debt is considered normal.
* Furthermore, research how an organization the size of the FAA with similar technical debt compares to both other government entities and private industry leaders’ standards or methods for addressing and handling technical debt.
* Finally, the FAA would also like to know more about potential third party vendors that offer a solution for solving technical debt.

In doing so, the FAA will be able to measure and compare their technical debt to others in similar industries and have a few justified solutions at their disposal to it. This will not only help them to perform more effectively but will also let them analyze the areas where they might need to improve.

**Constraints to be considered**

This project will be based on the team’s research. Unfortunately, there may be potential constraints that could impede the successful completion of the project. The biggest identified constraint is the team’s potential difficulty in gathering information. In terms of the FAA’s operations and systems, many details remain confidential and cannot be shared with the team. This might be an impediment to the project due to knowledge gaps. It is also highly probable that information on how other governmental entities deal with technical debt is not readily available to the team. This poses a challenge as the FAA would prefer the team to base research on other governmental entities. The second potential constraint is the current global COVID-19 pandemic. In addition to the usual precautions such as interacting exclusively online with the clients and the according communication limitations, it is also possible for a team member or key stakeholder to contract the virus. This will further impact the timeliness of our communication and can momentarily handicap the team. Lastly, there have been threats of a government shutdown, which would also disrupt our ability to effectively deliver a solution to the client.

**Section II: Alternative Evaluation and Recommendation**

In researching technical debt, the team may find software that can be used to either measure or help to identify technical debt. Depending on the depth of the research found, it may be possible for the FAA to use this information as a guide in building their own measurement tool for technical debt. Another possibility in the case that there actually is existing software for this purpose, the FAA could purchase that software for their own use. If these solutions are found through research, but the FAA wishes not to pursue these options, we recommend that they conduct further research on the subject to enhance their knowledge and potentially modify their existing systems to reduce technical debt.

**General Estimates of Cost and Merit**

When comparing potential options, it is important for the FAA to consider the costs that accompany each. Likely, the most expensive option would be creating their own measurement tool. This would require them to conduct further research in addition to designing and implementing the tool. As such, it would require a significant amount of time and manpower dedicated to creating their own tool. The second most expensive option would be to purchase an existing measurement software tool. There currently is no knowledge on if this even exists, but considering the impact that technical debt has on many companies, this software would likely be very expensive to invest in. The last option, to simply continue research, is potentially the cheapest option. Although there is no way of knowing how long this research would last for until a valid solution is found, it would require the least manpower and would require less investments.

**Comparison of Alternatives**

The alternative options will be further researched throughout our investigation into technical debt. Currently, we are researching how other top companies manage their technical debt, whether it be through certain software or developing their own best practices.

**Recommended Alternative**

As we gain more insight into handling technical debt, the recommended alternatives will be discussed more thoroughly at a later point. As of now, the recommended alternative is to continue research into software solutions.

The goal of this project is to provide general insights into how other organizations measure and handle their technical debt to the FAA. with the intent of the IT department utilizing the information that will be delivered to revise and improve their current information systems. Our team plans to investigate the three main types of technical debt, focusing on industries with similar information systems. We plan to investigate the current best practices for handling technical debt, as well as industry solutions that currently exist for handling similar issues.

Our project team will deliver the following:

* Detailed documentation of the project and our findings such as:
  + Statement of Work
  + Project Definition Report
  + Final Report
* Meetings with the Client(s) to further define requirements to ensure the intended scope of our research is met.
* Project Definition Presentation:
  + Present the documentation we have created thus far, allowing us to make changes and adjust our focus in potential problem areas or direction
* Proposed solution walkthrough
* Final proposed solution Presentation that will consist of elements such as:
  + Review strategic alignment emphasizing the intended impact of new system according to the FAA’s values.
  + Summary of interest areas.
  + Visualizations to aid the delivery of key points.
  + Government entity comparisons.
  + Private industry methodology for handling technical debt.
  + Details on relevant 3rd party alternatives.
  + Suggested course of action for handling technical debt.

**Section III: Findings**

From the Project Definition Presentation and Report, the FAA has tasked the team with defining the different types of technical debt using industry consistent terminology and ways to identify them; uncovering how other organizations address and handle their technical debt; finding a method the FAA could adopt to calculate their technical debt; and discovering preventative measures and management frameworks for technical debt. The findings from further research into the aforementioned areas are detailed below.

**Definition and Ways to Identify Technical Debt**

Technical Debt, also known as design or code debt is a metaphor for the result of prioritizing expedited delivery over design perfection in code. Although this can happen in several ways; technical debt always refers to the “debt” a company will have to pay off later because of shortcuts now. Technical Debt was originally coined by software developer, Ward Cunningham, to explain to non-technical stakeholders at WyCash the necessity for better resource budgeting. Looking back, Cunningham’s logic when the metaphor was created centered around the basic business concept of debt. Similarly, Mike Duensing, CTO and EVP Engineering at Skuid agrees, stating “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 like why we take on financial: We need something now that we do not have “cash” to pay for in full. (Duensing, 2020)

Given those definitions, what qualifies technical debt? Is it a tool that can be either bad or good like financial debt too?

Cunningham reflects on how his idea initially sparked: “With borrowed money, you can do something sooner than you might otherwise, but then until you pay back that money, you’ll be paying interest. I thought borrowing money was a good idea, I thought that rushing software out the door to get some experience with it was a good idea, but that of course, you would eventually go back and as you learned things about that software you would repay that loan by refactoring the program to reflect your experience as you acquired it.” (Ernst, 2015)

Despite most IT professionals at least being familiar with the term, Technical Debt has a wide variety of accepted qualifications.

“Shaun McCormick’s definition of technical debt focuses more on the consequences in the long term, ‘I view technical debt as any code that decreases agility as the project matures. Note how I didn’t say bad code (as that is often subjective) or broken code.’ He suggests that true technical debt is always intentional and not accidental. “(Casey, 2020)

“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 (Brodley, 2020)

“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 (Nelson, 2020)

“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 (Stone, 2020)

Overall, we are providing these contrasting definitions to expose the FAA to other industry leaders’ opinions. From the context of this project, we are focusing on defining Deliberate, Accidental or Outdated, and Bit Rot specific technical debt definitions. These types vary in degree of concern as well as ideal methods for addressing a solution. Deliberate debt should be handled while keeping in mind the time and resources necessary to repay something later versus the benefit in the present. If all stakeholders are more aware of the implications of technical debt in the long run, individual members can contribute to tackling technical debt from every facet of an organization. Our research indicates that logging this type of debt when a decision to defer work is made helps assure it gets repaid and eliminated. Ideally, stakeholders of each project should be held accountable for accruing deliberate tech debt. Accidental/Outdated technical debt occurs when designing software systems and is going to happen in some form regardless of how well you plan. In general, balancing your design with both efficiency and the future in mind is critical to reducing the difficulty of adding new functionality in the long run. Operating on this type of debt calls for a refactoring of a system every year or so in “steady” state. Bit Rot technical debt always happens over time, as a system is incrementally changed by different programmers, with varying styles and degrees of understanding of the original design, a system will gradually attain more complexity than needed. Experts from our research suggest this technical debt should always be actively avoided by periodically refactoring any given system.

Below is a chart of the main forms of technical debt as well as a description and the root cause for each. This table serves to give a general overview of the main forms of technical debt as well as a simple explanation as to who is responsible for the different types.

|  |  |  |  |
| --- | --- | --- | --- |
| **Term** | **Example** | **Who’s Responsible** | **How to Manage** |
| Deliberate | Intentional shortcuts to decrease time to market. | Stakeholders and product owners since this type of debt is caused by business decisions. | Keep a backlog to track all deliberate instances that need to be addressed at a later point. |
| Accidental or Outdated | Constantly changing system requirements, bad design based on new specifications. | Team leads and Product Owners because debt results from business decisions and design decisions. | Refactoring code is unavoidable in some instances but updating old/over complex code every year or so when the app is in “steady” state is recommended. |
| Bit Rot | System has become overly complex due to incremental changes, usually made by multiple developers and is made worse by styles overlapping. | Development team because this type of debt is incurred by individual developers. | Should always be avoided and is decreased by continuous refactoring of code. |

(FirstMark, 2020)

Identifying Technical Debt is a big concern in defining and classifying it. Focusing on the user is the first method of identifying technical debt, and often one of the easiest to implement as thinking from the user’s perspective is a general practice in software development. Another commonly used method is through simply monitoring your software’s output information. Speed tests, load speeds, or other performance issues or errors can indicate that a software is not performing up to standard. These can quickly point out areas of issue and help eliminate coding habits that generate technical debt, enhancing your understanding of the types of technical debt as well as how it occurs to remedy such. Spreading this knowledge beyond just the tech savvy individuals of a firm can increase the breadth of which tech debt is assessed and addressed in an organization.

The long-term goal for many IT leaders is to move assets away from legacy systems and apply money and time to more innovative work. Technical debt is not always a bad thing. Just like regular debt, sometimes organizations might need to do or acquire something immediately. In these scenarios the short-term debt incurred outweighs the downsides if it paves for greater successes down the road. Nonetheless, decisions that are made for things in the short run might have negative future repercussions.

**How Other Organizations Address and Handle Technical Debt**

The FAA would like the team to research how other organizations, be it government agencies or others in the private industry, address and handle their technical debt. On this subject, the team managed to interview Senior Software Engineer for Home Depot, Mr. William Meeks.

Mr. Meeks works on systems and applications regarding customer loyalty and has built an application from the ground up in that role. Although Mr. Meeks had only put in a few hundred lines of code into that project initially, it has grown to nearly a million lines in the span of 4 years from the contributions of numerous other developers. Based on his experience working on the mentioned project, tackling technical debt requires numerous special considerations. If something in the program is classified as debt, Home Depot has to consider many factors such as the technology, stakeholders, other units involved, end users, contractors, and future projects. Furthermore, multiple systems might depend on a legacy system that is a source of technical debt. These factors must be considered in the decision-making process to address the technical debt.

For Home Depot, its enterprise involves thousands of employees and stakeholders in most decisions due to interdependent systems. Mr. Meeks posits that it is extremely challenging for the organization to net positive in an enterprise-wide reduction of technical debt. At an enterprise level, decisions often create some form of technical debt despite their best efforts. This occurs since there are many factors that can contribute to technical debt. Exacerbating this is the fact that decisions made by management in order to address technical debt can often have potential cascading effects that pose difficulties in other areas.

The following encompasses the key considerations taken by Mr. Meeks from his experience at Home Depot when handling technical debt:

When replacing a system,

* What is new that is needed for the project to succeed?
* What did not work well in the last system?
* What can be added to prevent technical debt in the future?
* What type of resources will be needed?
* Will the new system be cloud-based? If so, what additional requirements does that pose for successful project completion?
* Finally, what worked decently well on the legacy system, but had bugs, or required adjustments over time and could have been better if implemented another way?

When retiring a legacy system for a new project,

* What are other applications that depend on the legacy system?
* Will retiring the legacy system force the retirement of other useful tools or systems in good standing?

The notes taken from Mr. Meeks interview can be found in Appendix B.

Unfortunately, the team was unable to uncover any other matrices, formulas, or specific and conclusive data on that matter in the pursuing research. This is attributed to the fact that being able to manage technical debt can be a core competency which creates competitive advantages for firms with best practices already in place.

According to Stripe’s 2018 report, *The Developer Coefficient*, developers waste more than 17 hours of their 41.1 total hours average work week on dealing with “bad code” and errors, debugging, refactoring, and modifying. This translates to roughly 42.1% of their time addressing technical debt and bad code. Figure 1 below illustrates the above (The Developer Coefficient, 2018).

Chart, pie chart

Description automatically generated

Figure 1: Average Weekly Developer Time Spent on Technical Debt and Bad Code

This time spent dealing with technical debt results in a loss in efficiency of developers. Furthermore, it can demoralize teams, and is estimated to cost companies $85 billion annually (The Developer Coefficient, 2018). As such, successfully managing technical debt is viewed as an advantage and, therefore, relevant data on the topic relating specifically to government agencies or private industry leaders is not made available.

**Calculating Technical Debt**

When it comes down to trying to measure technical debt in a codebase or system it is important to be able to quantify the ratio of bad code and technical debt compared to the rest of the system. Calculating the technical debt ratio (TDR) is done using the following formula:

**Technical Debt Ratio = (Remediation Cost / Development Cost) x 100%**

To find the remediation cost, the organization must quantify the amount of time and money it would cost to clean up the systems in question. For development cost, you can multiply the number of lines of code by an estimate of the number of hours each line would take to write to determine the development cost for a program in question. The lower the percentage is in the technical debt ratio, the better. Ideally, organizations want to have less than 5% of technical debt in their IT systems where possible. (Okwufulueze, 2020)

Accounting for the size and scale of systems operated in the FAA, it would be exceedingly difficult to manually calculate the TDR. The FAA could utilize several software tools that would assist in automating and reporting technical debt on a regular basis to management.

One example of a software tool that can achieve this is called SonarQube. This tool is a free, open-source code quality management tool that allows developer teams to track, manage, and eventually resolve their technical debt. It contains features such as dashboards and reporting tools that will track and calculate a variety of useful metrics in a large enterprise system. SonarQube will also categorize the code it determines to be debt into a group of buckets called the technical debt pyramid. Once analysis is complete, SonarQube will assign a letter grade to each application or system it is focused on based on the TDR.

* **Technical Debt (TD) Ratio <=10%, the rating is A**
* **11%<= TD Ratio <=20%, the rating is B**
* **21%<= TD Ratio <=50%, the rating is C**
* **51%<= TD Ratio <=100%, the rating is D**
* **over 100%, the rating is E**

In addition to software tools and the TDR, an organization can also benefit from categorizing their systems in question to determine the appropriate level of priority needed for each individual application. Our research suggested the following basic categorizations:

* **Local.** This is technical debt that’s confined within a class or a module. Changing it would have either minimal or no impact on other classes or modules it works with. It’s relatively localized, e.g., replacement of a logging system or CI steps improvement. This is the simplest to work on.
* **Global.** Fixing this technical debt will involve the entire software, e.g., changing the codebase from Java to Kotlin or introducing global architecture change. Some of such changes can be compartmentalized to make them local changes, though.
* **System.** This change is beyond your software and involves your external dependencies changes, e.g., a revamp of an entire analytic system of a mobile app from client-base to server-base, where the change involves both client and server. This is the most complex to work on.

**Preventative Measures and How to Manage Technical Debt**

Although the team was unable to find relevant data on how other organizations handle their technical debt, we were instead able to discover recommendations for preventing and managing technical debt. From our research, we compiled a list of tips from numerous articles written by IS experts, based on their experience, in dealing with this matter.

According to one of the articles found during research, *Erasing Tech Debt* written in 2020, a simple method would be to focus on creating the right approach. It suggests three initial steps on prioritizing how you manage your technical debt.

1. Manage technical debt the same way as financial debt: In other words, stop spending or creating more debt, and pay the debt off over time.
2. Use metrics to quantify the business impact of technical debt: Managers should use these metrics to determine when and where their team is being slowed by technical debt.
3. Assess product life cycles: Managers should conduct a risk analysis for both creating and addressing technical debt.

Next, it provides a four-pronged approach to indoctrinate an effective culture to manage technical debt.

1. Make sure to have buy-in from engineers first: So, developers need to support and understand the importance of addressing technical debt.
2. Reframe “technical debt” as “continuous product health”: This will improve workers’ mindsets about technical debt, helping them to think about it as the health of the work they produce.
3. Make addressing technical debt an everyday practice
4. Make “fixing work” and “investment work” first-class citizens: In other words, let those who enjoy mending systems mend, and let those who like creating systems create. (Erasing tech debt, 2020)

It is important for IT leaders to determine what decisions will contribute to technical debt throughout their development cycle. This allows them to allocate the proper resources to code that requires refactoring. The authors of a study done by the Software Engineering Institute at Carnegie Mellon University developed a flowchart that is shown below (Bellomo et al., 2016). This serves to determine technical debt using issue trackers. The issue trackers were taken from Chromium and CONNECT which are Health IT Exchange open-source projects and two government IT projects. By utilizing issue trackers, the development team can help decrease the effects of deliberate and accidental or outdated technical debt. (Bellomo et al., 2016)

Diagram

Description automatically generated

Figure 2: Flow Chart to Determine Technical Debt Using Issue Trackers (Bellomo et al., 2016)

Another method of managing technical debt proposed by researchers is the use of gamification. This method was developed after a request from the Pole Emploi, an agency of the French government, to help minimize technical debt (Blanc et al., 2018). It works to address technical debt using elements of a game in a non-game context. These researchers created a gamification tool named Themis that can integrate with the SonarQube tool mentioned previously. Themis provides features such as points, leaderboards, and challenges to encourage developers to fix and remove technical debt. In addition to this, it allows managers to monitor developers’ technical debt contributions using a view that shows the actions of their developers. In other words, they can see what technical debt was added or removed, and by whom. Many developers may find software maintenance non-rewarding, so Themis aims to create more excitement for this type of work through friendly competition. In the conclusion of their study, the researchers found that in the three-week timespan, developers and managers both noticed an improvement in the reduction of technical debt. (Blanc et al., 2018)

Yet another method is proposed by Gabor Zöld and includes tips for both preventing and managing technical debt (2021). His advice for preventing technical debt includes:

* Stay up to date: Tools, frameworks, and libraries should always be updated.
* Document: Make sure to keep track of everything that needs fixed.
* Conduct code reviews: Another set of eyes can help to catch mistakes before they’re made.
* Use automated testing: This should be used regularly to catch problems with code before too much is accumulated.
* Use Agile architecture: Agile architecture requires code to be flexible and maintainable, so it also helps to prevent technical debt buildup.
* Conduct blame-free post-mortem research: The focus of these analyses should be on understanding the problem and its cause to prevent reoccurrences.

Zöld also gives additional advice on managing technical debt (2021):

* Refactoring high-interest technical debt first: The debt that might create the most problems should be addressed first.
* Boy Scout rule - Continuous refactoring: In general, technical debt should be fixed whenever it is found. Although, this can be very time consuming, so set aside a percentage of time (recommends 5-33%) for each sprint to do so.
* Repay debt while performing valuable customer work: Designate a maximum amount of time during each sprint for technical debt refactoring.
* Just ignore it: Not all technical debt needs to be fixed. Don’t waste time and effort.

Finally, although SonarQube can be useful in evaluating technical debt, there are other ways to analyze particular kinds of debt. These may be useful to help determine specific issues developers are facing. One study focuses on detecting deliberate debt, otherwise known as ‘self-admitted’ debt. Researcher Aljohani made use of a Natural Language Processing (NLP) classifier to pull developer comments from their code. Next, he used a tool called SATD-detector to pull the comments that were categorized as self-admitted technical debt. This list contains 62 common words and phrases that indicate self-admitted technical debt. By using this list, Aljohani was able to narrow down the original list of comments to only those indicating technical debt. By doing this process multiple times considering different commits, it also allowed him to see how much of the self-admitted debt was eventually resolved. (Aljohani, 2019)

**Section IV: Software Solutions**

The extent of the FAA's technical debt was unknown to the team, therefore we included a multitude of options of potential software solutions for managing technical debt. At the end of this section we have included a chart of potential software solutions by SalesForce and SonarQube, the team found that can help manage technical debt.

**Gartner**

Gartner is a technology research company and consulting firm that provides services such as research, executive programs, consulting, and conferences. According to their website, the team could not access their extensive services unless we booked a consulting appointment with them to customize solutions for specific problems. Nonetheless we found an unbiased review about the services offered by Gartner. According to the article, using information provided by Gartner is an industry standard and many companies do not question the insights they bring. Critics of Gartner bring up some of their past failed predictions such as their prediction on how both SAP Fiori and IBM Watson would be much more successful than they were. Some critics believe that Gartner is not looking to make proper forecasts, rather, they are trying to make a splash with their prediction or satisfy a large vendor with its projections says Shaun Snapp.

A standard subscription to Gartner is $30,000 for basic analysis. The price may be lower depending on how often you have questions for analysts and if you can have online access to their article databases. Gartner purposefully gives less information at the lower levels, so their clients are encouraged to pay even more for more precise information in whatever field they are looking into. If you are looking into working with Gartner it is important to know that the more money you put into Gartner products and services, the more information you will be provided with. Gartner is in a dominant position in the IT analyst market so they can charge a high price for the services they offer. If you have the money to spend with Gartner, it can be a rewarding experience to have information and services provided by their specialists. However, if your budget is tight and does not have enough capital to spend for additional services with Gartner specialists, we would not recommend working with them because you will not be provided with full background information of what you are looking into. (Snapp, 2017)

**Third Party Solutions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tool/Service** | **Identifies** | **SalesForce Offering** | **Cost** | **Description** |
| StaticCode Analyzer | Code Quality | No | PMD is free; others involve cost | StaticCode Analyzers help identify code quality and security. Different tools are required for analyzing Apex and [LWC (JavaScript)](https://www.softwaretestingmagazine.com/tools/open-source-javascript-code-analysis/). |
| [Security Scanner](https://security.secure.force.com/security/tools/forcecom/scanner) | Security | Yes | Free | This tool is used to perform a static analysis scan of all unpackaged code in your organization. |
| [Salesforce Optimizer](https://help.salesforce.com/articleView?id=sf.optimizer_kick_off.htm&type=5) | Holistic High Level Org Assessment | Yes | Free | The Salesforce Optimizer app analyzes more features than the existing PDF report and in a more interactive and actionable format. |
| [FieldTrip](https://help.salesforce.com/articleView?id=000321289&type=1&mode=1) | Field Usage | No | Free | FieldTrip is a common AppExchange tool for the discovery of unused fields and data quality. |
| Code Review from Premier Success | Code Quality | Yes | Free with Premier/Signature Success offerings | This tool provides in-depth code analysis and review; it is available via MCS/Premier Success. |
| Expert Coaching Session (formerly called Accelerators) | Variety of Issues | Yes | Free with Premier/Signature Success offerings | Accelerators are free Premier Success offerings that customers can sign up for. Check out the [Accelerator Library](https://www.salesforce.com/content/dam/web/en_us/www/documents/accelerators/accelerator-library.pdf) and review the [Request Process](https://help.salesforce.com/articleView?id=000337601&type=1&mode=1) to get started. |
| [Proactive Monitoring](https://help.salesforce.com/articleView?id=000313873&type=1&mode=1) | Performance, Usage, Limits | Yes | Free with Premier/Signature Success offerings | Proactive Monitoring provides continuous, 24/7 monitoring of customers' key Salesforce solutions, helping to predict and prevent issues. |
| [Org Health Assessment](https://help.salesforce.com/articleView?id=000319484&type=1&mode=1) | Holistic High Level Org Assessment | Yes | Free with Premier/Signature Success offerings | The Org Health Assessment helps you check your system’s overall health and provides recommendations on how to improve it. |
| [Eagle Eyes](https://appexchange.salesforce.com/appxListingDetail?listingId=a0N3u00000OMjSSEA1) | Performance, Limits | Yes | Free | This Salesforce Labs AppExchange tool provides a way to visualize an organization's performance data using Event Monitoring data and Einstein Analytics. |
| [Maturity Assessment](https://help.salesforce.com/articleView?id=000358359&type=1&mode=1) | Variety of Issues | Yes | Free with Premier/Signature Success offerings | This Salesforce program is designed to determine the organization's maturity for a specific cloud/business unit or DevSecOps. |
| [Salesforce Professional Services](https://www.salesforce.com/services/overview/) | All of the above, and more | Yes | Paid | Salesforce Professional Services can offer additional help identifying and measuring technical debt with proprietary tools and offerings. In addition to identifying technical debt and system best practices, Salesforce Professional Services can provide advisory and consulting services to review processes and recommend alignment to industry standards with innovation and customer experience in mind. |
| [SonarQube](https://docs.sonarqube.org/latest/) | Identifies bugs and detects code smells | No | Free | Provides necessary analytics, tools and metrics to manage and track different sources of technical debt. |
| [StepSize](https://www.stepsize.com/)  Figure 3: Software Solutions provided by (Salesforce Architects, 2021)  ) | Tracks unresolved issues in code | No | $25 a month per user | Tracks and prioritizes the most important technical debt within codes |

**Appendix A: Signed Statement of Work**

Statement of Work

Analyzing Technical Debt

Prepared for the FAA

10/18/2021

**Organization Description:**

The Federal Aviation Administration (FAA) is a government organization that has a wide array of responsibilities concerning civil aviation and the U.S. commercial airspace. They promote air safety including but not limited to the manufacturing, operation, and maintenance of aircraft. They also develop and maintain the systems for air traffic control. Additionally, they communicate with foreign authorities to share information, organize air traffic, and set agreements.

The FAA has many applications, as such, they would like to analyze and evaluate technical debt. Here, they hope to find ways to reduce their technical debt by studying other industries and finding solutions. This will enable the FAA to perform IT functions more efficiently by saving time, capital, and labor in the long run.

**Problem Statement:**

The FAA has an interest in figuring out how to handle their technical debt. They would like us to investigate ways to measure their technical debt, reduce it, and compare themselves to similar industries. This will not only help them to perform more effectively but will also let them analyze areas where they need to improve. Specifically, they would like us to research how an organization the size of the FAA with similar technical debt compares to both other government entities and private industry leaders’ standards or methods for addressing and handling technical debt. The FAA would also like to know more about potential third party vendor’s that offer a solution for solving technical debt. The three main types of debt we are focusing on is Deliberate, Accidental or Outdated, and Bit Rot.

**Work Product:**

The goal of this project is to provide the FAA with general insights into how other organizations measure and handle their technical debt with the intent of the IT department utilizing the information in order to revise and improve their current information systems. Our team plans on investigating the three main types of technical debt, focusing on industries with similar Information Systems. We plan on looking into the current best practices for handling technical debt, as well as industry solutions that currently exist for handling similar issues.

Our project team will deliver the following:

* Detailed documentation of the project and our findings such as:
  + Statement of Work
  + Project Definition report
  + Final Report
* Meetings with the Client(s) to further define requirements to ensure the intended scope of our research is met.
* Project definition Presentation:
  + Present the documentation we have created thus far, allowing us to make changes and adjust our focus in potential problem areas or direction
* Proposed solution walkthrough
* Final proposed solution presentation that will consist of elements such as:
  + Review strategic alignment emphasizing the intended impact of new system according to the FAA’s values
  + Summary of interest areas
  + Visualizations to aid the delivery of key points
  + Government entity comparisons
  + Private industry methodology for handling technical debt
  + Details on relevant 3rd party alternatives
  + Suggested course of action for handling technical debt

**Constraints:**

Despite this project largely being based on our team’s research, there are potential constraints that we will face. The first constraint is that it will likely be difficult to find information about how other government entities deal with technical debt. Another potential constraint is that Covid-19 is still a large issue. It is possible that our clients or the team could contract Covid, impacting the timeliness of our communication. Lastly, there have been threats of a government shutdown, which would also disrupt our ability to effectively deliver a solution to the client.

**Resources required:**

* Our client’s time to provide insight and direction.
* Permission to interview other employees.
* Nothing should have to be purchased unless there is an article that is not free to read but contains information useful to the project.

**Milestone Dates:**

* Problem Definition Presentation: Tuesday, Oct. 26th
* Progress Meeting 1: Wednesday, Oct. 13th
* Progress Meeting 2: Wednesday, Oct. 20th
* Progress Meeting 3: Wednesday, Nov. 3rd
* Final Project Document: Thursday, Nov. 18th
* Final Presentation: Thursday, Dec. 2nd

**Signatures:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name:** | **Title:** | **Signature:** | **Date:** |
| **Dr. Bob Beatty** | **Professor** |  |  |
| **Mr. Kim McCord** | **Lead EA** |  |  |
| **Mr. Keith Burlison** | **ESC CTO** |  |  |

**Appendix B and Following**

Interview: William “Rex” Meeks from Home Depot

Updated: 11/15/2021

**William “Rex” Meeks**

**Senior Software engineer for Home Depot**

Works on systems and applications surrounding customer loyalty.

Around 4 years of experience

First project: Owned an application that is used as an orchestrator to retire a legacy system which involved handling the technical debt associated with the legacy system.

**Petting zoo analogy:**

Imagine you were on a team of architects and the higher up executives give you a relatively clear set of instructions to build that pen. At the start of the project, they tell you they want it to be a circle and needs to be big enough to hold up to 8 petting zoo animals (pigs, mini horses, alpacas etc.) and they want you to build it in 6 months. You say great and start working. You choose your materials and get to work. You decide to build it out of wood, because it’s cost effective and begin making other foundational judgments to build that pen and start implementing those decisions. 4 months pass and the pen is almost complete, and you have a meeting with the executives on the progress of the project just for them to tell you that new information has come out surrounding the project and you need to make it an oblong shape now and 2 of the animals are allergic to wood, and it now needs to be able to hold an elephant, you’re kind of screwed because you already started down this path and made critical judgements along the way that aren’t easy to simply change in a way that meets the new project requirements, and that in a sense of developing a system or application, automatically becomes tech debt. Because you now have all this work completed that is now useless because of the requirements of the project changing and you haven’t gained any business value out of those 4 months of work and are faced with changing almost everything starting from square one. Now you’ll have an unreliable new pen that’s patched together from what you could salvage of the work the team already did, which complicates new requirements that become known in the future, forcing you to create more technical debt in future projects.

Technical debt has many factors beyond just the technology. You must consider Stakeholders, other business units involved, end users, contractors in the loyalty system scenario, future projects that will depend on compatibility with the new system and so on.

You can have one system that has multiple other systems or applications that depend on the legacy system that’s in question. You must factor that into your decision-making process on the routes you can take to address the technical debt.

**Example at Home Depot:**

Another scenario is sheer size of code, rex's application he built from the ground up his first year on the job, only put a few hundred lines into the project initially, and in 4 years has grown to nearly a million lines of code from the contributions of multiple other developers over time. That means in a tech debt sense that if something in that program is classified as debt, you have to consider if someone who built that section of the application works there still, or perhaps only understands an elevated level of their section of code and not the inner workings of key areas, these types of considerations can also play a significant role in resolving that debt with the legacy system.

**Is it easy to solve technical debt issues?**

At a small startup level with say 40-100 employees, it's entirely possible to fix certain forms of technical debt and net less total technical debt from the solutions to the old debt. However, at an enterprise level such as Home Depot with thousands of employees and stakeholders involved in every decision with dependent systems, it becomes extremely hard to net positive in an enterprises reduction of technical debt, due to the difficult positions management is often faced with considering the bigger potential cascading effects of making a decision in addressing the debt. At an enterprise level, decisions often create technical debt in some form despite best efforts because there are so many ways to contribute to technical debt in the first place.

**Considerations:**

If you’re going to replace a system some key factors to consider would be what is new that is needed for the project to succeed, what didn’t work well in the last system, what can be added to prevent technical debt in the future, and what type of resources will you need, such as servers, is it going to be cloud based in the new system? If so, what additional requirements does that pose for successful project completion? Looking into legacy systems, how they worked well, and where they fell behind is a good methodology to begin working.

In the option to retire a legacy system for creating a new project, one must consider that due to interdependencies from a variety of applications (e.x. an old data warehouse,) if you retire that one thing, you may also have to retire a possible multitude of other applications or systems that may still be in good standing.

**Annotated Bibliography**

*3 main types of technical debt and how to manage them*. (2020, October 18). Hacker Noon. Retrieved November 15, 2021, from https://hackernoon.com/there-are-3-main-types-of-technical-debt-heres-how-to-manage-them-4a3328a4c50c.

This article defines and provides examples of the different forms of technical debt that are referenced in this report. This article also includes information as to how each type of debt is caused and who should be held responsible.

Aikat, K. (2021, June 29). *Amazon announces AWS BUGBUST, an international coding event to pay off Tech-debt worth $100 million*. Content.TechGig.com. Retrieved November 15, 2021, from https://content.techgig.com/amazon-announces-aws-bugbust-an-international-coding-event-to-pay-off-technical-debt-worth-100-million/articleshow/83916911.cms.

Amazon announces an international competition, AWS BugBust, calling on developers to solve 1 million software flaws and pay off $100 million in technical debt. AWS BugBust aims to raise awareness for their AI-powered tool Amazon CodeGuru to identify and solve companies’ most expensive lines of code.

Aljohani, A. (2019). *An empirical study on discovering a new self-admitted technical debt type - API-debt*. Available from ProQuest Dissertations & Theses Global. (2239303068). https://login.ezproxy.lib.ou.edu/login?url=https://www.proquest.com/dissertations-theses/empirical-study-on-discovering-new-self-admitted/docview/2239303068/se-2?accountid=12964

In this paper, Alijohani conducted a study using multiple test sessions to further explore one way to help diminish self-admitted technical debt (SATD). His method uses a tool called Natural Language Processing (NLP) classifier that pulls comments from codes. Next, he uses another tool called SATD-detector that pulls the SATD comments from the total comments. He also breaks down SATD into categories and defines a new category called API debt. Lastly, he uses a method that shows how much SATD is actually removed from the final code.

Bellomo, S. (2016, June 27). *Got technical debt? Track technical debt to improve your*

*development practices*. Insights.sei.cmu.edu. Retrieved November 15, 2021,

from https://insights.sei.cmu.edu/blog/got-technical-debt-track-technical-debt-to

improve-your-development-practices/.

In this article, the author references a study that was conducted by SEI at Carnegie Mellon University, where they utilized issue trackers to classify technical debt. The article includes a flow chart of whether certain issues contribute to technical debt or not.

Bellomo, S.,  Nord, R. L., Ozkaya, I., and Popeck, M. (2016). *Got Technical Debt?*

*Surfacing Elusive Technical Debt in Issue Trackers*. 2016 IEEE/ACM 13th

Working Conference on Mining Software Repositories (MSR), pp. 327-338.

This is the study that is referenced in the above article that describes how issue tracker can be utilized to identify and manage different types of technical debt.

Blanc, X., Foucault, M., Storey, M. A., et al. (2018, February 8). *Gamification: a Game*

*Changer for Managing Technical Debt? A Design Study*. arxiv.org. Retrieved

November 14, 2021, from https://arxiv.org/abs/1802.02693.

The authors in this paper provide their suggestions for managing technical debt (TD). Their proposed solution to this challenge involved the creation of a gamification tool, they named Themis, to address TD. In this case, they used game elements in a non-game context, such as leaderboards, points, & challenges, to encourage developers to fix technical debt.

Buchanan, S., &amp; Ganly, C. (2020, August 17). *Manage technology debt to create*

*technology wealth*. Gartner.com. Retrieved November 15, 2021, from

https://www.gartner.com/en/documents/3989188/manage-technology-debt-to

create-technology-wealth.

Casey, Kevin (2020, June 3) How to Explain Technical Debt in Plain English. Retrieved November 10, 2021, from https://enterprisersproject.com/article/2020/6/technical-debt- explained-plain-english. This article provides industry leader’s definitions of Technical Debt and these are referenced in the Findings section.

Codabux, Zadia & Williams, Byron. (2013). *Managing technical debt: An industrial case*

*study*. 2013 4th International Workshop on Managing Technical

Debt. https://www.researchgate.net/publication/261075398\_Managing\_technical

debt\_An\_industrial\_case\_study

Doty, E., &amp; Dunn, P. (2021, July 29). *How federal agencies can identify and*

*address technical debt*. fedtechmagazine.com. Retrieved November 15, 2021,

from https://fedtechmagazine.com/article/2021/09/how-federal-agencies-can

identify-and-address-technical-debt.

CIOs and their teams should poll mission units and program managers to identify the most critical applications or systems they operate. They, then, must figure out if the former identified items can be retired. The idea is to shift towards addressing technical debt in a proactive manner.

Efimova, P. (2021, January 26). *Tools to track and manage technical debt*.

Stepsize.com. Retrieved November 15, 2021, from

https://www.stepsize.com/blog/tools-to-track-and-manage-technical-debt.

Eliav, R. (2020, November 19). *Prioritizing technical debt the agile way*. Panaya.

Retrieved November 15, 2021, from https://www.panaya.com/blog/modern

alm/technical-debt-agile-way/.

Elye. (2020, June 23). *Practical tech debt prioritization*. BetterProgramming.pub.

Retrieved November 15, 2021, from https://betterprogramming.pub/practical

tech-debts-prioritization-443aa7c43e21.

*Erasing tech debt: A leader's guide to getting in the black*. Pluralsight.com. (2020,

February 4). Retrieved November 15, 2021, from

https://www.pluralsight.com/blog/software-development/erasing-tech-debt.

Technical debt chart. Technical debt’s impact on engineers, team, and organizations. The article gives examples of how technical debt can occur in terms of automation, cognitive load, and absence of context in daily practices. After which, technical debt can be addressed by reframing it as continuous product health, and making fixing work and investment work a priority. When prioritizing how to manage the debt, organizations must use metrics to quantify the impact of the debt and assess product life cycles.

Ernst, N. (2015, July 27). *A field study of technical debt*. Insights.sei.cmu.edu.

Retrieved November 15, 2021, from https://insights.sei.cmu.edu/blog/a-field

study-of-technical-debt/. This case study over Technical Debt goes in depth into defining and analyzing its different properties.

Garg, A., Narayan, B., &amp; Tran, S. (2021, July 22). *Defining, identifying, and*

*measuring technical debt*. Medium.com. Retrieved November 15, 2021, from

https://medium.com/salesforce-architects/defining-identifying-and-measuring

technical-debt-5f783e2b381d.

13 different types of technical debt and some tools to identify and access the debt. These include static code analyzers to identify code quality and security, FieldTrip to discover unused fields and data quality, Salesforce Eagle Eyes to visualize an organization’s performance data using Event Monitoring data.

Guo, Y., et al. (n.d.). Tracking technical debt — An exploratory case study. *2011 27th*

*IEEE International Conference on Software Maintenance (ICSM)*, pp. 528-

531. https://ieeexplore.ieee.org/document/6080824

Homme, K. (2018, January 31). *How can agencies address technical debt?*

DevBlogs.Microsoft.com. Retrieved November 15, 2021, from

https://devblogs.microsoft.com/azuregov/how-can-agencies-address-technical

debt/.

Agencies should shift from a capital expenditure (capex) dominant model to one more focused on operating expenditure (opex). CAPEX are purchases that will be used to improve an organization’s performance in the future where benefits are defined by longevity. However, with the increasing costs to maintain legacy systems, OPEX enables the use of as-a-service offerings so agencies can shed non-differentiated services and offload commodity workloads.

Marko, K. (2020, September 15). *How to manage technical debt in IT organizations*.

SearchITOperations.TechTarget.com. Retrieved November 15, 2021, from

https://searchitoperations.techtarget.com/tip/How-to-manage-technical-debt-in

IT-organizations.

2020 Charts for percentage of IT time spent on technical debt, polls of real costs incurred by technical debt, and techniques employed within organizations to manage technical debt. To create a debt reduction plan, an organization must assess existing IT infrastructure, estimate support and maintenance costs, report on the state of technical debt, create a strategy and update schedule to address technical debt, maintain a standard for all IT infrastructure to prevent new problems.

Merrill, M. (2020, September 24). *How to identify technical debt in your software*.

DevSquad. Retrieved November 15, 2021, from https://devsquad.com/blog/how

to-identify-technical-debt/.  This article gives industry specific advice as to how to identify technical debt. This gives perspective into common practices for identifying different types.

Miller, J. (2018, January 11). *How agencies can get out from under $7 billion in*

*technical debt*. FederalNewsNetwork.com. Retrieved November 15, 2021, from

https://federalnewsnetwork.com/cio-news/2018/01/how-agencies-can-get-out

from-under-7-billion-in-technical-debt/.

Consortium for IT Software Quality (CISQ) is an organization which has worked with experts to develop a new set of standards based on severe known problems in codes that are quintessential to fix. In an interview with CISQ CEO, Curtis, he said that CISQ looks at the complexity of a defect through a static analysis tool to adjust the level of effort and come up with a cost to fix the defect. The static analysis software looks at structure aspects of software to ensure that code is well constructed, securable, scalable and represents good engineering approaches. Finally, Curtis also said that agencies should start with understanding their technical debt with a policy change or improvement rather than jumping right into standards implementation.

O'Keeffe, D. (2017). *An Empirical Case Study of Technical Debt Management: A*

*Software Services Provider Perspective* (dissertation).

Okwufulueze, D. (n.d.). What is Technical Debt and How Can It Be

Measured? Andela.com. Retrieved November 15, 2021, from

https://andela.com/insights/what-is-technical-debt-and-how-can-it-be-measured/.

Omeyer, A. (2021, July 27). The perfect process to manage Tech Debt. Dzone.com.

Retrieved November 15, 2021, from https://dzone.com/articles/the-perfect

process-to-manage-tech-debt.

Technical debt according to Omeyer, Co-founder of Stepzone, a SaaS company to measure and manage technical debt, can be successfully managed by developing appropriate processes that are fully integrated into the usual Agile framework and have become habits. Essentially the organization must be clear on how to deal with small, medium, and large pieces of technical debt. For small pieces of technical debt, engineers must feel empowered to fix it as soon as spotted without the need for anyone else’s approval. This culture is important for a healthy codebase and there are tools to aid this. For medium-sized technical debt, organizations should identify debt that hinders crucial initiatives, or costs the organization dearly in terms of engineer productivity. Documenting and quantifying its costs allows the organization to prioritize the debts that, if addressed, will deliver value. For large technical debt, teams have to review the data necessary to understand the cost of the technical debt whereby leadership will sequence the debt accordingly based on the organization’s priorities. This debt will be approved and scheduled into the roadmap like feature work.

Papapetrou, P. (n.d.). *How to calculate technical debt and express it clearly*.

ThinkApps.com. Retrieved November 15, 2021, from

http://thinkapps.com/blog/development/technical-debt-calculation/.

In this article, the author discusses the value of reporting technical debt in way that can be easily interpreted and understood by both technical and non-technical team members. It describes the difficulties large code bases have in manually calculating technical debt and recommends 3rd party software tools.

SalesForce Architects. (2021, July 22). *Defining, identifying, and measuring technical debt*.

Medium. Retrieved November 17, 2021, from https://medium.com/salesforce-architects/defining-identifying-and-measuring-technical-debt-5f783e2b381d.

This article cites a study that identified 13 different types of technical debt and includes the definition of each. The article also includes a chart that displays a list of SalesForce Technical debt management product offerings that is included in our report.

Schwartz, M. (2020, December 16). *The CIO-CFO Conversation: Technical Debt—An*

*Apt Term?*AWS.Amazon.com. Retrieved November 15, 2021, from

https://aws.amazon.com/blogs/enterprise-strategy/the-cio-cfo-conversation

technical-debt-an-apt-term/.

This article describes ways in which technical debt should be communicated to other stakeholders. They prefer to use the term technical delta and functional delta to describe the differences between where the code is at and where they want it to be ideally. Tech delta relates more to code quality and design, while functional delta relates to the business decisions or new features of the proposed system.

Snapp, S. (2017, June 15). *How to Understand Whether Gartner is Worth the Investment*.

Brightwork Research & Analysis. Retrieved November 11, 2021, from https://www.brightworkresearch.com/gartner-worth-investment/.

This article weighs out the pros and cons of Gartner for users to decide if it is worth the investment.

*The Developer Coefficient*. (2018, September). Stripe.com. Retrieved November 14, 2021, from https://stripe.com/files/reports/the-developer-coefficient.pdf.

Stripe partnered with Harris Poll to survey developers, technical leaders and C-level executives about their organizations’ business challenges, software development practices, and future investments to determine the role that developer productivity plays in their success. In this report, they found that technical debt and bad code takes up a significant amount of developer time which hurts morale and presents an opportunity cost to organizations. It further outlines the impact and corresponding poll results from the respondents.

*Technical debt*. (2021, September 2). ProductPlan. Retrieved November

15, 2021, from https://www.productplan.com/glossary/technical-debt/.

This source gives a general definition of technical debt. It also discusses some

of the different types of technical debt and answers the question, “Is technical

debt bad?”.

Zöld, G. (2021, June 29). *Technical debt: All you need to know about managing it*.

Codingsans.com. Retrieved November 15, 2021, from

https://codingsans.com/blog/technical-debt.

This article presents some best practices to prevent and manage technical debt. To prevent it, organizations must be up-to-date on tools and frameworks, maintain documentation on technical debt, conduct code reviews, use automated testing, implement agile architecture, and conduct blame-free post-mortems. To effectively manage technical debt, organizations must prioritize the refactoring of high-interest technical debt first, engage in continuous refactoring (boy scout rule), repay debt while performing valuable customer work, and ignoring it.