White Paper

CAST
ACHIEVE INSIGHT. DELIVER EXCELLENCE.

How to Monetize Your Application Technical Debt

A Data-Driven Approach to Balance Delivery Agility with Business Risk

While there are many ways to define and measure Technical Debt, one thing is clear—it has been growing exponentially as maintenance is starved and development teams are forced to cut corners to meet increasingly unrealistic delivery schedules. CAST clearly defines Technical Debt as the cost of fixing the structural quality problems in an application that, if left unfixed, are highly likely to cause major disruption and put the business at serious risk. Once Technical Debt is measured, it can be juxtaposed with the business value of applications to inform critical tradeoffs between delivery agility and business risk.

Executive Summary

While there are many ways to define and measure Technical Debt, one thing is clear—it has been growing exponentially as maintenance is starved and development teams are forced to cut corners to meet increasingly unrealistic delivery schedules.

In *Measure and Manage Your IT Debt*, Andy Kyte, Gartner VP and Fellow, eloquently illustrates the systemic risk in the application portfolio caused by the accumulation of Technical Debt over the last decade. Gartner estimates that Fortune 2000 businesses and large public sector agencies have an average IT debt of more than $200 million, so dealing with IT debt must be a priority for the coming decade.

CAST clearly defines Technical Debt as the cost of fixing the structural quality problems in an application that, if left unfixed, are highly likely to cause major disruption and put the business at serious risk. *CAST analysis shows that even a conservative calculation of Technical Debt in the typical business application tops $1 Million.*

A fundamental element in the calculation of Technical Debt is a "violation", which occurs when an application fails to accord with one or more rules of software engineering. Violations are the root causes of software cost and risk; in other words, they are the fundamental metric of structural quality. Putting a dollar figure on structural quality translates structural quality into a language the business can understand. It enables apples-to-apples comparisons with other monetized figures, driving informed tradeoffs between Technical Debt, business value, and IT investment.

When should you measure Technical Debt? CAST recommends you make it a part of the periodic review of your mission-critical applications.

1. Count and compare the number of structural quality violations once a quarter. Establish an automated process to measure the structural quality of an application.
2. Create an Acceptance Quality Gate as a threshold for accepting any application before it is put into production, so you can clearly communicate quality targets to internal teams and external service providers.
3. Strengthen your systemic risk reduction processes. Integrate the practice of measuring violations into your delivery model to remediate risks before they increase your Technical Debt.

Once Technical Debt is measured, juxtapose it with the business value of applications to inform critical tradeoffs between delivery agility and business risk. Set the appropriate threshold for Technical Debt and monitor critical applications against this threshold to keep the right balance between agility and business risk as IT and business conditions evolve.

Contents

I. Introduction

In *Measure and Manage Your IT Debt*, Andy Kyte, Gartner VP and Fellow, eloquently illustrates the systemic risk in the application portfolio caused by the accumulation of Technical Debt over the last decade. Gartner estimates that Fortune 2000 businesses and large public sector agencies have an average IT debt of more than $200 million, so dealing with IT debt must be a priority for the coming decade. His call to action is to collect reliable data about the scale of the problem.

At CAST Research Labs, our data repository of software structural quality data—**Appmarq**—provides a unique foundation for quantifying the scale of Technical Debt in businesses worldwide. In its current state, Appmarq contains data on software size, complexity, and structural quality from 75 IT organizations from around the world. There are 288 applications in the data set and each application is measured along 27 distinct attributes, resulting in a total of approximately 8,000 data points. This white paper builds on the summary of results presented in the CAST Worldwide Application Software Study–2010.

In this white paper we explain how Appmarq is used to monetize the Technical Debt of an application. A fundamental element in the calculation of Technical Debt is a "violation". Violations, as we will explain in more detail, are at the root of an application's structural quality. Hence, our monetization of Technical Debt is based on reliably collecting and quantifying the root causes of the systemic risks in an application. *Our results show that even a conservative calculation of Technical Debt in the typical business application tops $1 Million.* There is substantial systemic risk in applications but also a substantial opportunity for improvement.

We begin with a definition of Technical Debt and the result of our calculation of Technical Debt in a typical application. We then present the details behind this calculation – the fundamental elements in the calculation and how they are put together to generate the result. We conclude with recommendations for when Technical Debt should be measured and the actions CIOs should take once Technical Debt is monetized.

Highlights

We define Technical Debt as the cost of fixing the structural quality problems in an application that, if left unfixed, put the business at serious risk.

II. The Definition of Technical Debt and How It's Calculated

There are many ways to define and calculate Technical Debt, so let's begin with our definition and its merits. We define Technical Debt as the cost of fixing the structural quality problems in an application that, if left unfixed, put the business at serious risk. Technical Debt includes only those application structural quality problems that are highly likely to cause business disruption and hence put the business at risk; it does not include all problems, just the serious ones.

Under this definition of Technical Debt, we find that a typical application of 374,000 lines of code (KLOC) has more than $1 Million of Technical Debt. Technical Debt does vary by application technology/language. For hypotheses as to why and for more details, please see the CAST Worldwide Application Software Study–2010.

Given our definition of Technical Debt, measuring it requires us to quantify an application's structural quality problems that put the business at risk. This is where Appmarq comes in. Appmarq contains data on the structural quality of business applications (as opposed to data on the process by which these applications are built). Application structural quality measures how well an application is designed and how well it is implemented (the quality of the coding practices and the degree of compliance with the best practices of software engineering that promote security, reliability, and maintainability).

The basic measure of application structural quality in Appmarq is the number of violations per thousands of lines of code (violations per KLOC). Violations are instances when an application fails to accord with one or more rules of software engineering. Violations can be grouped according to their potential customer impact in terms of the business disruption they create if left unresolved: the higher the level of business disruption, the higher the severity of the violation. The most severe violations are categorized as "critical violations."

The number of violations per KLOC for each application is not obtained from surveys of project/program managers; rather, it is measured using the repeatable, automated CAST Application Intelligence Platform. Our approach therefore rests on the foundation of objective, repeatably-measured quantities. It is not susceptible to the subjectivity and inconsistencies that undermine survey-driven data collection. Moreover, the size of the data set is large enough to make robust estimates of the number of low-, medium-, and high-severity violations per KLOC in the universe of all business applications.

Highlights

We find that a typical application of 374,000 lines of code (KLOC) has more than $1 Million of Technical Debt.

We have independently verified the strong correlation between violations and business disruption events in a number of real-world field tests of mission-critical systems. By focusing solely on violations, this calculation of Technical Debt takes into account only the problems that we know will cause business disruption. We also apply this conservative approach to quantifying the cost and time it takes to fix violations (all assumptions are stated clearly below).

In defining and calculating Technical Debt as we do, we err on the side of a conservative estimate of the scale of Technical Debt. The actual Technical Debt is likely to be higher and our aim is to simply set the value for the floor – the lowest value it is likely to be. We think this is the right direction to err when it comes to monetizing Technical Debt.

III. Four Steps for Calculating Technical Debt

Step 1. The density of violations per thousand lines of code (KLOC) is derived from source code analysis using the CAST Application Intelligence Platform.

Step 2. Violations are categorized into low, medium and high severity. The Technical Debt calculation assumes that only 50% of high-severity violations, 25% of medium-severity violations, and 10% of low-severity violations require fixing to prevent business disruption.

Step 3. We conservatively assume that each violation, no matter its level of severity, takes 1 hour to fix at a fully-burdened cost of $75 per hour. Although these numbers could be a lot higher, especially when the fix is applied during operation, we assume these values to produce a conservative estimate.

Step 4. The formula for Technical Debt:

- L = Number of Low-Severity Violations per KLOC
- M = Number of Medium-Severity Violations per KLOC
- H = Number of High-Severity Violations per KLOC
- S = Average Application Size (KLOC)
- C = Cost to Fix a Violation ($ per Hour)
- T = Time to Fix a Violation (Number of Hours)

Technical Debt per Application = [(10% * L) + (20% * M) + (50% * H)] * C * T * S

Using Appmarq data to arrive at **the values for L, M, H, and S, the amount of Technical Debt in a typical business application of 374 KLOC is over $1 Million.**

Highlights

The monetization of Technical Debt translates structural quality into money, the universal language of business. It enables apples-to-apples comparisons that were not possible before.

Once Technical Debt is monetized, what next? In the next section we explain the steps that CIOs and Application delivery and maintenance heads should take once they have measured and monetized the Technical Debt of their business-critical applications.

IV. Setting a Technical Debt Threshold

Getting a handle on the systemic risk in an application begins with an assessment of its Technical Debt. This measurement is a way to monetize the quality of the application – it puts a dollar figure on the quality of an application. This monetization is critical because it translates structural quality into money, the universal language of business. It enables apples-to-apples comparisons that were not possible before.

We all know that application quality is important. Being able to monetize quality means we can now ask a further, critical question, namely, how much quality is enough? Or to put it another way, how much should we invest in this application to manage its systemic risk?

Figure 1 is a conceptual diagram that illustrates the tradeoff between Technical Debt and business value. Please keep in mind that the diagram is illustrative and uses no actual data.

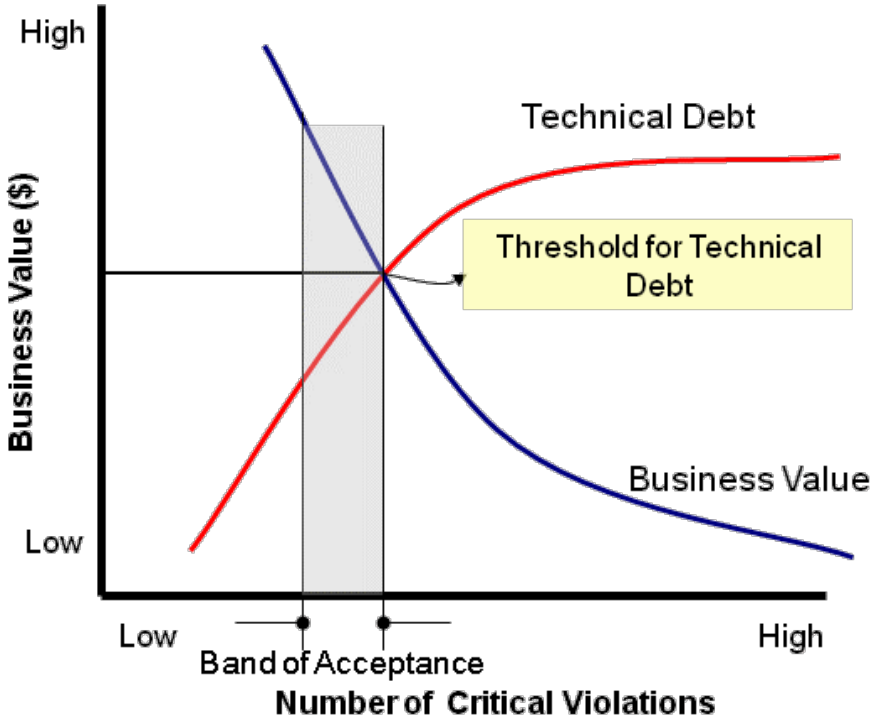

Figure 1. Application Technical Debt and Business Value as a Function of Structural Quality Violations (Conceptual)

Highlights

Three use cases to get started measuring Technical Debt are: Periodic Count of Structural Quality Violations; Acceptance Quality Gate; and Industrialization of Systemic Risk Reduction Processes.

The increase in Technical Debt as the number of violations rise is shown by the red line in Figure 1. The blue line shows the declining business value as the number of violations rise. The point of intersection at which the curves meet marks the maximum Technical Debt that can be tolerated by the application. Anything to the right of that means a precipitous drop in business value and a simultaneous rise in the cost to operate the application.

The goal is to keep the number of structural quality violations well to the left of the intersection of the curves. The range of acceptable values of Technical Debt below the threshold can vary based on the exact nature of the Technical Debt and the Business Value curves. This is indicated by the gray shaded area. Moving left beyond the shaded area might be too much of a good thing – there is a point beyond which improving quality has diminishing marginal improvement in business value.

V. From Monetization to Action–Three Use Cases

We recommend that CIOs and heads of Applications use an automated system to evaluate the structural quality of their three to five mission-critical applications. As each of these applications is being built, measure its structural quality at every major release. When the applications are in operation, measure their structural quality every quarter.

In particular, keep a watchful eye on the violation count; monitor the changes in the violation count and calculate the Technical Debt of the application after each quality assessment. Once you have a dollar figure on Technical Debt, use Figure 1 to determine how much Technical Debt is too much and how much is acceptable based on the marginal return on business value. For a framework for calculating the loss of business value due to structural quality violations, please see, The Business Value of Application Internal Quality by Dr. Bill Curtis.

Use Case 1: Periodic Count of Structural Quality Violations

While an application is being developed or being operated, establish an automated process for periodically measuring the structural quality of the application based on the number and the trend of structural quality violations. Use this information to make the right tradeoffs between delivery speed, application quality, and business value.

Use Case 2: Acceptance Quality Gate

Before you accept an application for production, measure its Technical Debt against a pre-set threshold for acceptance. Use this objective measure to clearly communicate your IT and business goals to your internal teams and to your service providers.

Highlights

Once Technical Debt is measured, you can juxtapose it with the business value of applications to inform critical tradeoffs between delivery agility and business risk.

Use Case 3: Industrialization of Systemic Risk Reduction Processes

Integrate the practice of measuring Technical Debt into your delivery model. Involve your developers, architects, QA, and DevOps to take immediate actions to reduce Technical Debt rather than wait until it might be too late (or too expensive). The cycle of measurement and structural quality improvement improves team learning, performance, and morale. Moreover, these improvements in the team's productivity can be quantified in terms of the same metrics that are used to measure structural quality.

VI. Conclusion

As Gartner analyst Andy Kyte recommends, the first step to getting a handle on the systemic risks in your portfolio is to measure the scale of Technical Debt in your applications. Measurement is the first step, but it is an important step. To ensure objective, cost-effective measurement, use an automated system to evaluate the structural quality of your business-critical applications. Make sure that your assessment of Technical Debt is grounded on a key driver of software structural quality.

The analysis in this white paper is grounded in objective counts of violations which have been verified in numerous field tests to be the key drivers of application costs and risks in organizations worldwide. The power of this Technical Debt calculation is not in its mechanics (which we have purposefully kept very simple) but in the fundamental bits of data on which it is based. The independent confirmation that these fundamental elements (structural quality lapses measured as number of low-, medium-, and high-severity violations) play a significant role in the business productivity of companies worldwide further strengthens the objectivity and accuracy of the calculation.

Once Technical Debt is measured, juxtapose it with the business value of applications to inform critical tradeoffs between delivery agility and business risk. Set the appropriate threshold for Technical Debt and monitor critical applications against this threshold to ensure that the right balance between agility and business risk is maintained as IT and business conditions evolve.

About CAST

CAST is a pioneer and world leader in Software Analysis and Measurement, with unique technology resulting from more than $90 million in R&D investment. CAST provides IT and business executives with precise analytics and automated software measurement to transform application development into a management discipline. More than 650 companies across all industry sectors and geographies rely on CAST to prevent business disruption while reducing hard IT costs. CAST is an integral part of software delivery and maintenance at the world's leading IT service providers such as IBM and Capgemini.

Founded in 1990, CAST is listed in NYSE-Euronext (Euronext: CAS) and services IT intensive enterprises worldwide with a network of offices in North America, Europe, and India.

www.castsoftware.com

North America
373 Park Avenue South
New York, NY 10016
Phone: +1 212-871-3330

Europe
3, rue Marcel Allegot
92190 Meudon - France
Phone: +33 1 46 90 21 00