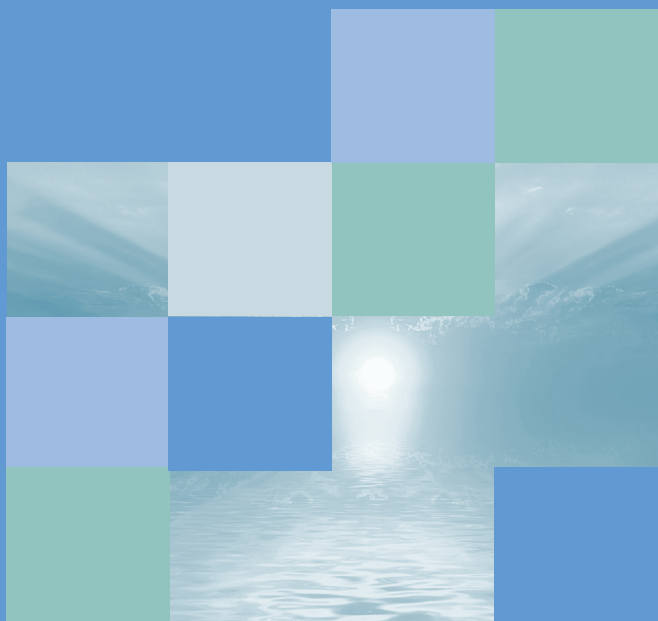
CAST

How to Monetize Application Technical Debt

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



Featuring research from

Gartner

Excerpt from:

Measure and Manage Your IT Debt

The following section is excerpted from *Measure and Manage Your IT Debt* by Gartner analyst Andy Kyte. To view the full note, click anywhere in the section.

When budgets are tight, maintenance gets cut. After a decade of tight budgets, the scale of the maintenance backlog has created systemic risk, particularly for large organizations. CIOs and IT management teams need to begin a new type of dialogue with the rest of the business about maintaining the integrity of critical application assets.

Key Findings

- The lack of a detailed application inventory, coupled with the lack of a strategic application life cycle planning discipline in most organizations, means that IT management teams have very little awareness of the scale of IT debt in their organizations.
- The consolidation in the IT industry, coupled with technology innovations, means that the next upgrade for many major applications will be a rearchitecting upgrade – a significantly more expensive and risky venture than many have experienced over the last 10 years.
- Current global IT debt is estimated to stand at $500 billion, with the potential to rise to $1 trillion by 2015.

Recommendations

- Implement the management disciplines associated with application portfolio management to gain an accurate and reliable inventory, and to understand the current costs of running the existing applications.
- Develop strategic plans for each application in the inventory to identify whether and when major maintenance activities will be required to maintain the engineering integrity of the portfolio.
- Publish a simple and accessible application status report at least annually, targeted at the managers in the rest of the business, with information that highlights the growing scale of the IT debt problem.

- Ensure that any business case for investment in a new system contains a lifetime total cost of ownership (TCO) estimate that takes into account the probable maintenance activities over the life of the system. Then put in place IT funding that recognizes the long-term commitment.

WHAT YOU NEED TO KNOW

The management of IT debt is an unwelcome addition to the challenges that beset an IT management team, but the problem is real, and it will not go away. The IT management team needs to make an honest assessment of the integrity of its entire application portfolio and estimate the budget that is likely to be needed to maintain the integrity of the application portfolio at an acceptable level for the needs of the business. This means ensuring that there is an accurate and reliable inventory of applications, with each application having a management plan that shows whether and when maintenance or replacement investment is likely to be needed. The results of such an assessment will be, at best, unsettling and, at worst, truly shocking. Whatever the results, the IT management team must make dealing with IT debt a priority for the coming decade.

STRATEGIC PLANNING ASSUMPTION(S)

Global IT debt is estimated to be $500 billion, with the potential to grow to $1 trillion by 2015.

ANALYSIS

The Sources of IT Debt

A modern enterprise or public sector organization is likely to be critically dependent on a number of business applications. These applications are engineering artifacts – that is to say, they are not just abstract concepts, but real collections of data and business logic encapsulated in programming instructions and myriad platform components, such as operating systems, databases, hardware and network infrastructure. These engineering artifact components are not immutable objects: Each one of them is at a particular point in a complex life cycle; each one of them is slowly but inevitably diverging from its ideal state toward a suboptimal state, and potentially toward obsolescence or failure. Through judicious investment in application maintenance, an application team can fight off the ravages of time and reverse some of the effects of aging – but such investments can be tough to justify in a tight economy when precious investment funds need to be used to deliver short-term, visible business benefits.

Over the past decade, the net effect of these conflicting forces of "application decay," and the shortage of investment funds to deal with this decay, has been that important maintenance activities have been repeatedly postponed in favor of important "business value add" projects. Alongside this internal finance trend, there have been some significant shifts in the technology markets that deliver applications, and the platforms that sustain applications, so that while the current version of an application may be running well, the next upgrade for many applications is going to involve a substantial shift in the underlying platform infrastructure – or portends the need for a replacement system. These decisions can be delayed for a while, but they cannot be indefinitely postponed. This combination of internal budget tightness and technology market changes means that the vast majority of organizations have built up a substantial backlog of "essential" maintenance activities that are becoming more important to address and more expensive to address as the years go by. One way to characterize this backlog of deferred liability is to see it as a debt incurred in previous years that will need to be paid off at some time. This "IT debt" is a hidden risk for many organizations, and, given continued tight economic conditions – meaning little or no money available for maintenance activities – this IT debt is growing, and the associated business risk is growing. In fact, as the business continues to invest in business value-add projects that add more functionality and complexity into the existing and aging portfolio, the size of the IT debt grows as well, since the additional functionality and complexity will need to be maintained and upgraded to a more reliable state at some point in the future.

The Scale of IT Debt

At this point, you might be saying: "Hasn't it always been like this?" While it is true that there has never been an IT organization without a backlog of maintenance activity, the scale of the problem is significantly greater than it has ever been. It is not unreasonable to suggest that the maintenance backlog – and, therefore, the IT debt – was probably at an all-time low on 31 December 1999, when every IT organization had spent a significant amount of money upgrading or replacing systems in order to deal with Y2K. Since then, however, the demands for IT investment to deliver real business value through running, growing or transforming the business have drained the maintenance coffers year after year, so year after year the IT debt has grown. And dealing with this burgeoning IT debt is a classic example of the management problem of trying to resolve the conflict between short-term business imperatives and long-term investment strategies.

How big is the problem of IT debt? In order to answer this question in a single organization, it's necessary to look at the inventory of all business applications, and then evaluate the potential costs for bringing all the elements in the portfolio up to a reasonable standard of engineering integrity, or replace them. The challenge here is

that very few IT organizations have reliable inventories of business applications, and even fewer have any meaningful estimates of the likely cost of delayed maintenance activities. However, based on research conducted with some client organizations with effective application portfolio management processes, we estimate that Fortune 2000 businesses and large public sector agencies have an average IT debt of more than $200 million – meaning that, globally, IT debt stands conservatively at $500 billion. Furthermore, as IT budgets are tightened and technology markets consolidate, the scale and complexity of major upgrades or replacement projects is growing, meaning that, without substantial effort to reduce it, global IT debt is likely to reach $1 trillion by 2015.

Managing IT Debt

So what can the CIO and the IT management team do to address the problem of IT debt? The starting point for any activity must be the collection of reliable data about the scale of the problem. This means developing an inventory of applications and a process to ensure that the inventory is maintained. It means calculating the cost of running every application in the portfolio, and then estimating the costs likely to be incurred in bringing the application up to an acceptable level of engineering integrity. (For a description of this process, please see "How the CIO Can Increase the Value of the Application Portfolio").

The key here is to express the impact of deferred maintenance in terms of the increase in risk to business processes and the degradation of efficiency of business processes. The issue must never be the applications themselves – it is what they enable that is important. This initial assessment is the beginning of knowledge, but the knowledge must be translated into action. The challenge here is that action will mean investment, and the rest of the business quite rightly wants to see tangible business results from IT investments. So addressing IT debt involves beginning a new type of discussion between the IT management team and the rest of the business – a discussion about the medium- to long-term viability and integrity of the application portfolio. This discussion might usefully start by engaging all the application stakeholders in the nuanced decisions that have to be made in the "cash flow versus risk" analysis. This is not an "instead of" discussion about how IT can contribute value through helping to run, grow and transform the business, it is additive to the existing dialogues, not an alternative.

Source: Gartner

Application Technical Debt: A Data-Driven Approach to Balance Delivery Agility with Business Risk

Introduction

Andy Kyte eloquently captures the systemic risk in the application portfolio caused by the Technical Debt that applications have accrued in last 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 article builds on the summary of results presented in the *CAST Worldwide Application Software Study – 2010*.

In this article 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.

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). To read more about how software structural quality metrics can be used to control IT costs and risks, please see the Gartner Newsletter *Software Risk Management: A Primer for IT Executives*.

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.

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.

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.**

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.

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.

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.

From Monetization to Action – Three Use Cases

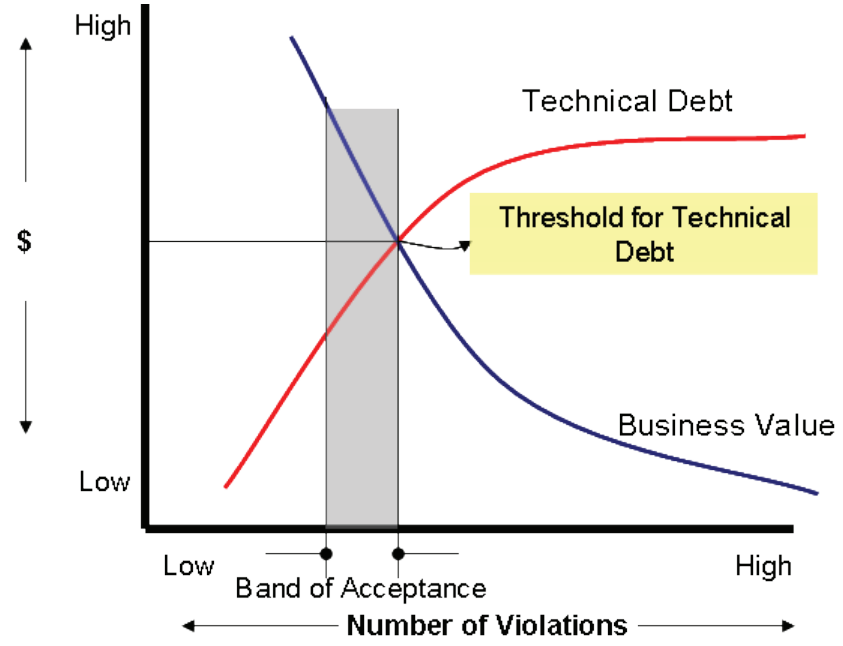
We recommend that CIOs and heads of Applications use an automated system to evaluate the structural quality of their 3 to 5 mission-critical applications. As each of these applications is being built, measure its structural quality at every major release. When the applications are in operation, measure 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.

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



Source: CAST

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.

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.

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 article 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.

Source: CAST

About CAST

The CAST Application Intelligence Platform is the only enterprise-grade software quality assessment and performance measurement solution available in the market today. The CAST solution inspects the source code, identifies and tracks quality issues, and provides the data to monitor development performance.

CAST can read, analyze and semantically understand most kinds of source code, including scripting and interface languages, 3GLs, 4GLs, web and mainframe technologies, across all layers of an application (UI, logic and data). By analyzing all tiers of a complex application, CAST measures quality and adherence to architectural and coding standards, while providing visual specification models. Managers get real time access to this information via a web interface by which they can proactively monitor, measure and improve application health and development team performance.

CAST's unique technology is the result of more than $80 million in R&D investment. Top engineering talent, dedicated to building the best technology for assessing the structural quality of mission-critical applications, has made CAST the leader in Automated Application Intelligence. CAST's mission is to use software measurement to transform application development into a management discipline.

Founded in 1990, CAST has helped more than 650 organizations worldwide accelerate IT delivery to the business, mitigate risks in production, improve customer experience, and reduce the total cost of application ownership. CAST is listed on NYSE-Euronext (Euronext: CAS) and serves Global 2000 organizations worldwide with a global network of locations in the US and Europe.

www.castsoftware.com
CAST Headquarters
North America: +1 212-871-8330
Europe: +33 1 46 90 21 00