7 Steps to Pay Down the Interest on Your IT Technical Debt

AVOID FAILURE OF CRITICAL APPLICATIONS

1. Set Priorities
2. Measure Tech Debt
3. Establish Targets
4. Plan Actions
5. Remediate Violations
6. Track Results
7. Report

CAST
Software Intelligence for Digital Leaders

www.castsoftware.com
# Contents

I. **Introduction** 1

II. **Measuring Technical Debt** 3

III. **The 7-Step Technical Debt Management Cycle** 7

IV. **The Real World of Technical Debt** 15
   - Case Study: Large insurer 16
   - Case Study: Telco provider 18

V. **Conclusion** 20

VI. **About CAST** 21
We all make mistakes.

That point could not have been clearer in 2011, a year in which experts generally believe that more IT system failures, outages and data breaches occurred than any previous year since the dawn of the age of technology. And while some of these issues could be traced to intentional and malicious undermining of systems, many, if not most, had some measure of application software failure or weakness at their root.

These failures or weaknesses in application software stem from two areas. One is when developers write software that violates good architectural or coding practices causing structural flaws in the code. The other is found in legacy code – lines of code that are carried over from previous versions of application software – which either do not work properly with the new code being written, are no longer valid or carry defects that were not previously detected, thereby exposing the new software to failure or breach.

Whatever the cause of application software failure, the cost of fixing the resulting structural quality problems in production code to control development costs or avoid operational problems is called Technical Debt.
As depicted in Figure 1, the causes of Technical Debt vary from simple explanations like an inadvertent coding mistake to intentional shortcuts based on business needs. Regardless of the origin, organizations that do not address the resulting structural flaws before application software is deployed expose themselves to significant risk and will inevitably find themselves addressing the issues when they manifest themselves as application failures. **This is how Technical Debt is amassed.**

Left unaddressed, Technical Debt can so degrade an application that its benefits to the business cannot be justified by its growing costs or operational risks. Consequently, managing Technical Debt is an executive liability for those responsible for governing the costs and risks of application portfolios.

**Figure 1. Reasons for Technical Debt**
Although Technical Debt represents liability, it also represents a way to apply financial terms to the business risks associated with operational performance of critical business applications.

Technical Debt’s usefulness has been limited historically because most organizations have not known how to measure it. As a result, these organizations have been incapable of using Technical Debt to estimate the costs of application ownership or to guide management decisions about how much to invest in application quality, which, when left unfunded, leads to more Technical Debt being accrued.

“Technical Debt’s usefulness has been limited historically because most organizations have not known how to measure it.”
Recent advances in software analysis and measurement, however, have solved this problem. Automated analysis and measurement is far more accurate than previously employed manual assessments and is therefore more reliable and efficient at detecting software quality issues – meaning companies now have an objective measurement of an application’s Technical Debt, making it a critical and effective tool for application management.

**Still, a single method of measuring Technical Debt does not exist.** Generally speaking, Technical Debt can be calculated by analyzing the structural quality of an application, rating the severity of each problem, and identifying the “must-fix” problems. Using this roadmap, CAST has worked extensively with industry leaders to research and devise a method for calculating Technical Debt, as shown in Figure 2, based on the number of must-fix problems in the code, the time needed to correct the problems and the cost to fix the problems.

“...companies now have an objective measurement of an application’s Technical Debt, making it a critical and effective tool for application management.”
**Number of must-fix problems:** The first step in calculating Technical Debt is to determine which issues will be addressed. Based on its research, CAST assumes that only 50% of high severity problems, 25% of moderate severity problems, and 10% of low severity problems will ultimately be corrected in the normal course of operating the application.

**Time to correct problems:** The next element in the calculation is the time it takes for remediation. The time to fix a structural quality problem includes the time to analyze the problem, understand the code and determine a correction, evaluate potential side-effects, implement and test the correction, and release the correction into operations.

**Cost to fix problems:** The final element in calculating Technical Debt is the cost of the time spent by a developer to remediate the issues. CAST found that the data on this subject varies widely from company to company, but based on its research it determined that $75 per hour was a conservative estimate of the average rate for developers. This figure, however, should be treated like a variable rather than a constant and companies calculating their own Technical Debt should use whatever the average rate for their developers is.

Using each of these elements, the formula developed by CAST to calculate Technical Debt is depicted in Figure 3.

\[
(0.1 \times \text{LSV} + 0.25 \times \text{MSV} + 0.5 \times \text{HSV}) \times 1 \text{ hour} \times 75
\]

Where, LSV = Low Severity Defects
MSV = Medium Severity Defects
HSV = High Severity Defects

**Figure 3.** CAST’s Formula for Calculating Technical Debt
It is important to note that the Technical Debt formula presented only provides a basis for benchmarking. Organizations can and should adjust the parameters in the formula to fit their own maintenance and structural quality objectives, experiences and costs.

CAST employed the formula on the previous page in its 2011-2012 CRASH Report (CAST Report on Application Software Health), based on analysis of 745 applications containing 365 million lines of code (11.3 million backfired function points) submitted between 2008 and 2011 by 160 organizations located primarily in the United States, Europe and India.

In doing so, it determined that the average Technical Debt accrued per line of code (LOC) for all applications reviewed was $3.61. That means even a slightly below average sized application containing 300,000 LOC carries more than $1 million in Technical Debt.
The Technical Debt Management Cycle is a seven-step process for analyzing and measuring Technical Debt (presented in Figure 4), that relies on comparing it to both IT and business priorities. This process begins by translating executive business priorities into Technical Debt targets for each application. Just as individuals make decisions on how to pay down personal debt, organizations need to determine how they will retire Technical Debt.

Trying to do this on a global level can be overwhelming and it is often difficult to visualize the expected payoff. However, the analysis and measurement of Technical Debt can guide critical management decisions about how to allocate resources for reducing business risk and IT costs. Executives can set specific reduction targets based on the strategic quality priorities of their organizations, weighing the costs of remediation against the expectation of achieved benefits. This is what is known as the Technical Debt Management Cycle.

It is vital that actions for achieving these targets are determined in advance, and to track the progress at each release and periodically report this progress to the business.
Set strategic quality priorities

Since application budgets and time are constrained, IT executives must determine the most critical structural quality objectives each application service offers to the business and provide clear-cut guidance to application managers. In many cases, they will need to set these priorities in concert with the application managers since executives will generally lean toward business risk factors, and because structural quality priorities typically vary depending upon the purpose of the application software. For instance, in highly regulated industries the most critical issue may be protecting customer data, which would make Security the highest priority. In highly competitive markets, business agility may be the most critical issue so Changeability would be a priority because of its impact on the speed of delivering new functionality. And an online retail application may need to satisfy several priorities, such as Robustness and Security.

Applications with large and growing levels of Technical Debt related to IT costs may need to prioritize these factors over business risk issues since the application architecture can degrade to the point that it becomes unable to serve its business mission. In these situations, IT executives must determine the balance between business risk factors and their own internal IT cost factors. Fortunately, structural quality analysis and measurement provides the data needed to optimize the balance among strategic priorities.

“In highly competitive markets, business agility may be the most critical issue so Changeability would be a priority because of its impact on the speed of delivering new functionality.”

The 7-Step Technical Debt Management Cycle

1. Set strategic quality priorities

8 Steps to Pay Down the Interest on Your IT Technical Debt
In much the same way that developers use freeware static analysis tools to measure Technical Debt as part of their unit testing regimen before submitting code to a build, structural quality flaws that cause Technical Debt should also be measured at numerous points in the software life cycle. The difference between the two, however, is that analysis at the testing level cannot detect serious architectural flaws that may span multiple layers of the application and may be written in different, sometimes incompatible languages. These multi-component flaws are often the most devastating during operations and are the most time-consuming to fix. Consequently, the most effective times to measure structural quality are either after each phase of a build or immediately before release in order to determine the efficacy of the application as a whole.

The build team or quality assurance typically performs structural quality analysis of application software, although organizations may conduct it in an Application Intelligence (AI) Center dedicated to this function. Whoever carries out the analysis should feed the results back to the development team along with a list of the violations that yielded the measures. These analyses should be retained as a baseline to assess progress toward strategic structural quality priorities.
To establish a basis for decision-making, application managers need to translate structural quality priorities into specific targets for their application. They can accomplish this in a few ways including stating them as thresholds, such as “no high severity performance defects in the operational code base,” or “sustain Transferability scores at or above 2.8 across releases.” The strategic priorities passed down by executive management should provide the guidance needed to allocate resources toward the most critical targets.

That, however, is where the tension arises. With every release, there is a “tug-of-war” over the needs of the organization to provide new functionality quickly to the business and the need to reduce Technical Debt in priority areas. As a result, structural quality targets can rarely be achieved in the span of one or two releases. Instead, Technical Debt reduction needs to be treated as a standard component of maintenance or sprint planning. Application managers must allocate time for structural quality improvement into their application maintenance plans, or in an agile environment allocate a percentage of stories in each sprint to remediating structural quality problems (a determination that needs to be factored in as part of the design of the agile process). Since developers frequently participate in these planning activities, they should be aware of the strategic structural quality priorities against which they are expected to plan.
Plan remediation actions

At the beginning of each release planning cycle or sprint, the application manager and development team can use the structural quality analysis from the previous release to prioritize a list of violations for remediation that will help achieve the application’s structural quality targets. The team should select as many high-priority violations to remediate as can be addressed by the resources allocated during the cycle or sprint, and it is important to make sure Technical Debt does not increase because of development activities that are separate from those being conducted for the remediation of Technical Debt.

To ensure the process of reducing Technical Debt remains consistent from one phase to the next, the list of prioritized violations should be revised and updated after each release. The rate of progress in remediating the backlog of violations should also be compared to executive priorities and timelines to determine if the amount of resources devoted to remediation needs to be adjusted in future cycles or sprints. This will ensure that Technical Debt planning continues throughout the application lifespan as part of the longer strategic plan.
Remediate violations

Development teams should address violations tagged for remediation within the cycle or sprint as part of their normal development or maintenance activities. These remediation activities may involve refactoring poorly designed code or correcting specific coding weaknesses. Testing and static analysis should then be used to verify that the flaw has been successfully remediated. The team should keep track of the time required to remediate structural flaws to help with more accurate estimates on the number of remediation tasks that can be undertaken in future cycles or sprints.

If a specific type of flaw has been detected numerous times across the code base, the development team should, as time allows, identify the root cause of the flaw and take steps to eliminate it; with agile in particular, this is an appropriate topic for retrospectives at the end of a sprint. Results of these root cause analyses should be reported back to the central team conducting the structural quality analyses so they can detect trends across the organization.

“If a specific type of flaw has been detected numerous times across the code base, the development team should, as time allows, identify the root cause of the flaw and take steps to eliminate it...”
Track results

The application manager should track the ongoing results of Technical Debt remediation efforts, compare the results against established targets and share them with the development team. Significant deviations against expected progress should be addressed when planning upcoming cycles or sprints to develop achievable commitments based on the resources allocated.

Periodically the application manager and IT executives should review their progress in retiring Technical Debt and reaffirm application targets against the strategic structural quality priorities; this allows them to adjust the targets if strategic priorities have changed. Technical Debt data can be used as input for estimating and tracking the long-term cost of ownership for the application, while structural quality data needed for upcoming management decisions regarding the application should be discussed.

“Technical Debt data can be used as input for estimating and tracking the long-term cost of ownership for the application...”
Following review with the application managers, IT executives should report the status of Technical Debt to the business. Because it highlights a direct relationship between aspects of either business risk or IT cost, Technical Debt can be translated into terms easily understood by those on the business side. Terms like reductions in risk, limiting exposure to security breaches or outages, increased capability for business agility or reductions in IT’s long-term costs all can be illustrated by Technical Debt and resonate with the business side. This information can also be reported to corporate auditors responsible for assessing risks to the business.

Translating Technical Debt into business risks, however, requires the business to articulate its costs and the losses experienced when applications suffer outages, performance degradation, security breaches, data corruption and similar events. The discussion of Technical Debt and its potential liabilities initiates a dialogue that provides businesses with greater understanding of the IT risks and costs and how they relate to business issues. Consequently, the Technical Debt Management Cycle provides a vehicle for greater alignment between the business and IT.

“The discussion of Technical Debt and its potential liabilities initiates a dialogue that provides businesses with greater understanding of the IT risks and costs and how they relate to business issues.”
While it is one thing to discuss Technical Debt, how to measure it and how to manage it, it is another thing all together to put it to use in actual practice.

The following are two cases where the determination of Technical Debt was used to improve not only the structural quality of application software, but also the conduct of business.
Case Study

Large insurer cuts maintenance costs 20%

A large insurer had a claims management system with 700,000 lines of code that managed 8 million claims per year from a client base of 10 million customers. Development of new functionality suffered as a result of substantial Technical Debt, and was typically accompanied by high defect rates.

To gain control over Technical Debt and reduce maintenance costs, the executive in charge of this application mandated that future development would be subjected to structural quality measurement and baseline targets were set for maintainability. Development teams were held accountable for making steady progress toward the maintainability targets release by release, and were presented the results of each structural quality analysis.

Figure 5. Reduction in Defect Rates for a Large Insurance Application
Over the course of four years, **they saw the following results:**

- Maintainability targets were stabilized against the baseline target even though the application code base grew by 40%.
- Defect rates in system test and operations fell 56% as Technical Debt was remediated and the application became easier to understand and maintain, as shown in *Figure 5*.
- Delivery time was reduced by 60% as a result of more maintainable code and less interference from rework.
- Maintenance costs for this application were reduced by 20% within the first three years.
Case Study

Telco provider reduces outages to save $2.7 million

CAST performed structural analysis of Technical Debt on the billing system of a large telecommunications company. Management’s objectives were to reduce rework and outages, as well as gain control over outsourced maintenance costs. If managing Technical Debt addressed the objectives, then the process would be deployed to reduce Technical Debt in other critical applications.

Technical Debt was measured at an initial release and critical violations of good architectural and coding practice (anti-patterns) were targeted for remediation. Over the subsequent three releases, the anti-patterns constituting Technical Debt were steadily reduced, as shown in Figure 6. Along with this reduction in Technical Debt was a highly correlated reduction in system test and operational defects. A decline in operational problems was also observed, which was correlated with reductions in defect rates, although the data were not available to prove a causal relationship.

Based on the successful pilot, structural analysis was deployed to a portfolio of critical applications to reduce and control Technical Debt. Figure 7 shows structural quality analysis was initiated on this particular application at release 8.

The effort to remediate Technical Debt is correlated with a reduction in system test and operational defects over the next 9 releases until it appears to stabilize at release 11.2. The same pattern was observed with other applications.
Reducing rework by lowering the number of defects resulted in a substantial reduction in maintenance costs.

IT estimated cost savings of $2.7 million in the first year on this particular application as a result of rework and outage reductions as well as better management of outsourced services.

**Figure 7.** Defect Volume Reduction across Releases
**Technical Debt** provides a way to apply financial terms to the risks inherent in applications that have structural flaws in code caused by violations of good architectural and coding practices. For those with executive responsibility for governing the costs and risks of an application portfolio, the Technical Debt Management Cycle provides a clear process for measuring and managing Technical Debt that can help in balancing IT and business priorities, and over time reduce the risk of failure of critical applications.

By following this 7-step process, an organization can weigh the costs of remediation against the expectation of achieved benefits—and ultimately pay down the interest of the overall liability inherent in the portfolio.

**Author**
Dr. Bill Curtis, Senior Vice President and Chief Scientist, CAST
VI. About CAST

CAST is the pioneer and category leader in Software Intelligence, providing insight into the structural condition of software assets. CAST technology is renowned as the most accurate "MRI for Software", which delivers actionable insights into software composition, architectures, database structures, critical flaws, quality grades, cloud readiness levels and work effort metrics. It is used globally by thousands of forward-looking digital leaders to make objective decisions, accelerate modernization and raise the security and resiliency of mission critical software., visit www.castsoftware.com

Call: 877-852-2278
Email: info@castsoftware.com
Visit our Web site: www.castsoftware.com

7 Steps to Pay Down the Interest on Your IT Technical Debt