TECHNICAL DEBT – WHY IT MATTERS TO CAPACITY MANAGERS

John Rhodes, Managing Director/CTO,
CM First Group

"One man's crappy software is another man's full time job."
Jessica Gaston

A rule of thumb (ROT) has always been that 80% of application performance derives from design choices and coding inefficiencies, leaving only 20% for performance analysts to work with. While no one really knows how true this is, there is no question that performance tuning alone cannot make a poorly designed application run as well as a thoughtfully designed one. Resource demands are also much higher when an application has been written without consideration of the impact of coding options. For capacity managers, the job is a lot harder when trying to make business transactions run efficiently and quickly at the lowest possible cost because the component with the biggest impact is beyond their control.

In a time where performance and capacity people are charged not just with 'keeping the system running,' but also have to manage costs and customer satisfaction while working faster and with fewer people than ever before, inefficient application code can make or break you. You need to understand and manage 'technical debt.'

'Technical debt' is the term coined in 1992 by Ward Cunningham, inventor of wikis and a signatory to the Agile Manifesto Doctrine, to describe application design trade-offs generally made to speed applications to production at the lowest price possible. Where once we believed that application code wouldn't last more than five years, many companies are still running code written in the '70's and '80's. Over all that time, problems accumulate; the older the code, the more likely it is brittle and will be difficult to maintain and enhance. Worse yet, it may be impossible to exploit desired new technologies such as Cloud or mobile with older code. 67% of CFOs surveyed strongly desired to modernize legacy applications; technical debt can make it difficult or impossible to achieve that goal.[i]

"Adapting old programs to fit new machines usually means adapting new machines to behave like old ones."
Alan J. Perlis

The cost to companies goes beyond performance. Availability can also be jeopardized, as can security. Security holes are more likely with older code because the knowledge we have accumulated to make code safe post-dates when the code was created. The costs can be significant. The older and more debt-ridden code is, the more it costs in people-time to maintain and the more resources it is likely to use. The end result is an impact on customer satisfaction as they find they cannot get from your company the services they demand. (Fig. 1)

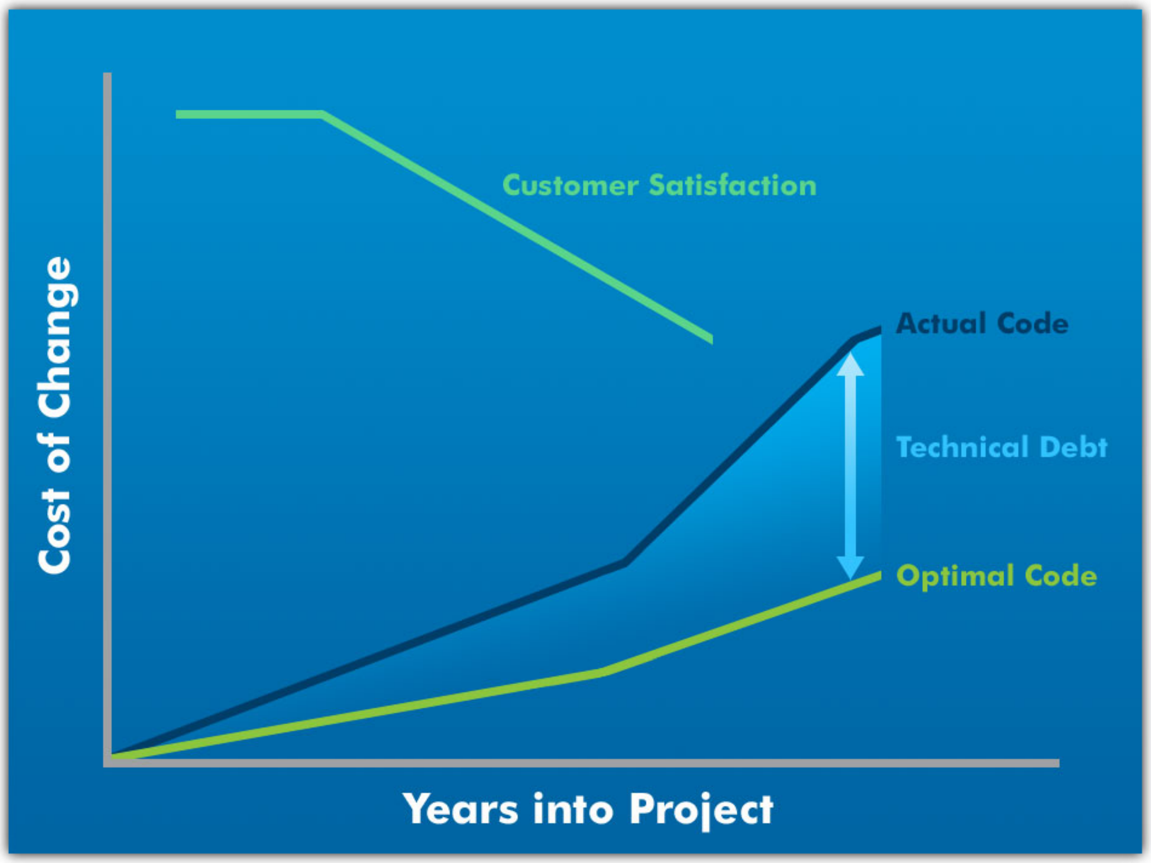

Fig. 1 - The Cost of Technical Debt

Where does ‘technical debt’ come from? The sources are as many and as varied as the people who created it. In some cases, code was written so long ago that standards and best practices simply didn’t exist. Languages can be almost obsolete, meaning maintenance programmers struggle to even understand what is written. Junior programmers made mistakes because of lack of experience; older, more experienced programmers might have taken shortcuts, rushed or created overly complex code, because they could. Code designed for a simple purpose may now be expected to support much more complex business processes and the original design may just not be optimal for that purpose. Sections of code have often been cloned and repurposed, so finding the problem code becomes much more difficult. Single platform applications now have to talk to and exchange data with other platforms. The original code probably didn’t plan for that eventuality. Outsourcing and the purchase of application software distances the coding process from the IT department charged with managing it. It’s very difficult to manage code quality when you don’t have the source. Older code may not support new business demands well. Old VSAM files may not handle the diverse requirements as well as a real database. Finally, in too many cases, lack of documentation makes it difficult to understand, accurately maintain and manage code.

Though the word ‘legacy’ makes people think ‘mainframe,’ other platforms have been around long enough to have developed their own technical debt issues. UNIX and Windows aren’t exempt, nor is Linux or iSeries. In fact, one area that many may neglect to check is the world of code generation, where the performance relies on the underlying design of the generator to produce optimized code.

When you're betting your job on performance, code matters. You simply can no longer afford to ignore it.

Though in the past, getting in on application design and having influence over purchase decisions was challenging – hardware was cheap and no one wanted to take the time – now, this problem is so sizable that it is literally everyone's problem. However, the obvious solutions – manual review of the code with an eye to fixing it or dumping the code and buying a purchased solution – aren't generally viable. Anyone who opted for a purchased solution to solve the Y2K problems knows that this is not a silver bullet. And it isn't practical to pore over code to find the problems. No one has the time and few probably have the ability. There are simply too many problems.

What are your options? Twenty years ago, you were probably stuck. Today, there exist a number of automated code analyzers that work across all varieties of hardware and software. They exploit multiple engines to work efficiently and quickly to produce a meta-analytic repository which defines exactly where the debt can be found. These engines look at code quality, maintainability, complexity and connectivity. The result is a module-by-module identification of the problems found, enabling fixes to be prioritized and quickly made. The best solutions help to find clones, so when you fix one problem, you can fix it for all iterations of that code segment. Look for the ones that provide all these features to get the best results.

Prioritization can be made based on what code executes most frequently, what code is up for major modification or is holding you back from modernization. Down the road, this process can fold into your continuous process improvement methodology, ensuring that you continually strive to fix problem code while minimizing the development of new problems.

You can't truly address performance anymore without getting the code working better. Reap the benefit of code analysis and modernization and save time and resources, while improving customer satisfaction. Work with your development teams and discover the benefits of partnership.

---

[i]

http://www.workday.com/resources/is_your_financial_system_killing_your_business.php?campid=ussm_li_no_fi_725#!