Technical Debt

Posted on November 1, 2007 1:06:PM by Steve McConnell to 10x Software Development

The term technical debt was coined by Ward Cunningham to describe the obligation that a software organization incurs when it chooses a design or construction approach that's expedient in the short term but that increases complexity and is more costly in the long term.

Ward didn't develop the metaphor in very much depth. The few other people who have discussed technical debt seem to use the metaphor mainly to communicate the concept to technical staff. I agree that it's a useful metaphor for communicating with technical staff, but I'm more interested in the metaphor's incredibly rich ability to explain a critical technical concept to non-technical project stakeholders.

What is Technical Debt? Two Basic Kinds

The first kind of technical debt is the kind that is incurred unintentionally. For example, a design approach just turns out to be error-prone or a junior programmer just writes bad code. This technical debt is the non-strategic result of doing a poor job. In some cases, this kind of debt can be incurred unknowingly, for example, your company might acquire a company that has accumulated significant technical debt that you don't identify until after the acquisition. Sometimes, ironically, this debt can be created when a team stumbles in its efforts to rewrite a debt-laden platform and inadvertently creates more debt. We'll call this general category of debt Type I.

The second kind of technical debt is the kind that is incurred intentionally. This commonly occurs when an organization makes a conscious decision to optimize for the present rather than for the future. "If we don't get this release done on time, there won't be a next release" is a common refrain—and often a compelling one. This leads to decisions like, "We don't have time to reconcile these two databases, so we'll write some glue code that keeps them synchronized for now and reconcile them after we ship." Or "We have some code written by a contractor that doesn't follow our coding standards; we'll clean that up later." Or "We didn't have time to write all the unit tests for the code we wrote the last 2 months of the project. We'll right those tests after the release." (We'll call this Type II.)

The rest of my comments focus on the kind of technical debt that's incurred for strategic reasons (Type II).

Short-Term vs. Long-Term Debt

With real debt, a company will maintain both short-term and long-term debt. You use short-term debt to cover things like gaps between your receivables (payments from customers) and expenses (payroll). You take on short term debt when you have the money, you just don't have it *now*. Short-term debt is expected to be paid off frequently. The technical equivalent seems straightforward. Short-term debt is the debt that's taken on *tactically and reactively*, usually as a late-stage measure to get a specific release out the door. (We'll call this Type II.A.)

Long term debt is the debt a company takes on *strategically and proactively--investing in new capital equipment, like a new factory, or a new corporate campus. Again, the technical equivalent seems straightforward: "We don't think we're going to need to support a second platform for at least five years, so this release can be built onthe assumption that we're supporting only one platform." (We'll call this Type II.B.)*

The implication is that short-term debt should be paid off quickly, perhaps as the first part of the next release cycle, whereas long-term debt can be carried for a few years or longer.

Incurring Technical Debt

When technical debt is incurred for strategic reasons, the fundamental reason is always that the cost of development work today is seen as more expensive than the cost will be in the future. This can be true for any of several reasons.

*Time to Market***.** When time to market is critical, incurring an extra $1 in development might equate to a loss of $10 in revenue. Even if the development cost for the same work rises to $5 later, incurring the $1 debt now is a good business decision.

Preservation of Startup Capital. In a startup environment you have a fixed amount of seed money, and every dollar counts. If you can delay an expense for a year or two you can pay for that expense out of a greater amount of money later rather than out of precious startup funds now.

*Delaying Development Expense***.** When a system is retired, all of the system's technical debt is retired with it. Once a system has been taken out of production, there's no difference between a "clean and correct" solution and a "quick and dirty" solution. Unlike financial debt, when a system is retired all its technical debt is retired

with it. Consequently near the end of a system's service life it becomes increasingly difficult to cost-justify investing in anything other than what's most expedient.

Be Sure You Are Incurring The Right Kind of Technical Debt

Some debt is taken on in large chunks: "We don't have time to implement this the right way; just hack it in and we'll fix it after we ship." Conceptually this is like buying a car—it's a large debt that can be tracked and managed. (We'll call this Type II.A.1.)

Other debt accumulates from taking hundreds or thousands of small shortcuts--generic variable names, sparse comments, creating one class in a case where you should create two, not following coding conventions, and so on. This kind of debt is like credit card debt. It's easy to incur unintentionally, it adds up faster than you think, and it's harder to track and manage after it has been incurred. (We'll call this Type II.A.2.)

Both of these kinds of debt are commonly incurred in response to the directive to "Get it out the door as quickly as possible." However, the second kind (II.A.2) doesn't pay off even in the short term of an initial development cycle and should be avoided.

Debt Service

One of the important implications of technical debt is that it must be *serviced*, i.e., once you incur a debt there will be interest charges.

If the debt grows large enough, eventually the company will spend more on servicing its debt than it invests in increasing the value of its other assets. A common example is a legacy code base in which so much work goes into keeping a production system running (i.e., "servicing the debt") that there is little time left over to add new capabilities to the system. With financial debt, analysts talk about the "debt ratio," which is equal to total debt divided by total assets. Higher debt ratios are seen as more risky, which seems true for technical debt, too.

Attitudes Toward Technical Debt

Like financial debt, different companies have different philosophies about the usefulness of debt. Some companies want to avoid taking on any debt at all; others see debt as a useful tool and just want to know how to use debt wisely.

I've found that business staff generally seems to have a higher tolerance for technical debt than technical staff does. Business executives tend to want to understand the

tradeoffs involved, whereas some technical staff seem to believe that the only correct amount of technical debt is *zero*.

The reason most often cited by technical staff for avoiding debt altogether is the challenge of communicating the existence of technical debt to business staff and the challenge of helping business staff remember the implications of the technical debt that has previously been incurred. Everyone agrees that it's a good idea to incur debt late in a release cycle, but business staff can sometimes resist accounting for the time needed to pay off the debt on the next release cycle. The main issue seems to be that, unlike financial debt, technical debt is much less visible, and so people have an easier time ignoring it.

How do You Make an Organization's Debt Load More Visible?

One organization we've worked with maintains a debt list within its defect tracking system. Each time a debt is incurred, the tasks needed to pay off that debt are entered into the system along with an estimated effort and schedule. The debt backlog is then tracked, and any unresolved debt more than 90 days old is treated as critical.

Another organization maintains its debt list as part of its Scrum product backlog, with similar estimates of effort required to pay off each debt.

Either of these approaches can be used to increase visibility into the debt load and into the debt service work that needs to occur within or across release cycles. Each also provides a useful safeguard against accumulating the "credit card debt" of a mountain of tiny shortcuts mentioned earlier. You can simply tell the team, "If the shortcut you are considering taking is too minor to add to the debt-service defect list/product backlog, then it's too minor to make a difference; don't take that shortcut. We only want to take shortcuts that we can track and repair later."

Ability to Take on Debt Safely Varies

Different teams will have different technical debt credit ratings. The credit rating reflects a team's ability to pay off technical debt after it has been incurred.

A key factor in ability to pay off technical debt is the level of debt a team takes on unintentionally, i.e., how much of its debt is Type I? The less debt a team creates for itself through unintentional low-quality work, the more debt a team can safely absorb for strategic reasons. This is true regardless of whether we're talking about taking on Type I vs. Type II debt or whether we're talking about taking on Type II.A.1 vs. Type II.A.2 debt.

One company tracks debt vs. team velocity. Once a team's velocity begins to drop as a result of servicing its technical debt, the team focuses on reducing its debt until its velocity recovers. Another approach is to track rework, and use that as a measure of how much debt a team is accumulating.

Retiring Debt

"Working off debt" can be motivational and good for team morale. A good approach when short-term debt has been incurred is to take the first development iteration after a release and devote that to paying off short-term technical debt.

The ability to pay off debt depends at least in part on the kind of software the team is working on. If a team incurs short-term debt on a web application, a new release can easily be rolled up after the team backfills its debt-reduction work. If a team incurs short-term debt in avionics firmware— the pay off of which requires replacing a box on an airplane— that team should have a higher bar for taking on *any* short-term debt. This is like a minimum payment--if your minimum payment is 3% of your balance, that's no problem. If the minimum payment is $1000 regardless of your balance, you'd think hard about taking on any debt at all.

Communicating about Technical Debt

The technical debt vocabulary provides a way to communicate with non-technical staff in an area that has traditionally suffered from a lack of transparency. Shifting the dialog from a technical vocabulary to a financial vocabulary provides a clearer, more understandable framework for these discussions. Although the technical debt terminology is not currently in widespread use, I've found that it resonates immediately with every executive I've presented it to as well as other non-technical stakeholders. It also makes sense to technical staff who are often all-too-aware of the debt load their organization is carrying.

Here are some suggestions for communicating about debt with non-technical stakeholders:

Use an organization's maintenance budget as a rough proxy for it's technical debt service load. However you will need to differentiate between maintenance that keeps a production system running vs. maintenance that extends the capabilities of a production system. Only the first category counts as technical debt.

Discuss debt in terms of money rather than in terms of features. For example, "40% of our current R&D budget is going into supporting previous releases" or "We're currently spending $2.3 million per year servicing our technical debt."

Be sure you're taking on the right kind of debt. Not all debts are equal. Some debts are the result of good business decisions; others are the result of sloppy technical practices or bad communication about what debt the business intends to take on. The only kinds that are really healthy are Types II.A.1 and II.B.

Treat the discussion about debt as an ongoing dialog rather than a single discussion. You might need several discussions before the nuances of the metaphor fully sink in.

Technical Debt Taxonomy

Here's a summary of the kinds of technical debt:

Non Debt

Feature backlog, deferred features, cut features, etc. Not all incomplete work is debt. These aren't debt, because they don't require interest payments.

Debt

I. Debt incurred unintentionally due to low quality work

II. Debt incurred intentionally

II.A. Short-term debt, usually incurred reactively, for tactical reasons

II.A.1. Individually identifiable shortcuts (like a car loan)

II.A.2. Numerous tiny shortcuts (like credit card debt)

II.B. Long-term debt, usually incurred proactively, for strategic reasons