Technical Debt,… It's your problem too

Doug Riley, Sr. Architect 07/25/2017

Debt is something most of us understand in a monetary sense.

- Debt: A duty or obligation to pay money, deliver goods, or render service under an express or implied agreement. (BusinessDictionary.com)

Technical debt isn't as easy to define. It is not easy to get rid of. It is not easy to avoid. It is essentially the future cost (in labor and hardware) to eliminate a perceived debt.

In today's "everything was due yesterday" world much of technical debt is incurred by taking short cuts. Why? Because everything was due yesterday, yesterday was Sunday, and there was a great football game on TV. Monday the project manager stops by (or stands by if you are Agile) and helps you out by suggesting that everything is fine if you just cut a couple corners here and there. I mean, really, testing (for example) is a waste of time. If there are problems in the code the users will find it eventually and you can fix it then. One way of looking at technical debt is that you promised to deliver something but what you delivered is less than what you promised. The difference between the two is a form of technical debt. Another common form of technical debt is when older or existing technology is used because it is cheaper, already exists, or is easier to write. This is technical debt consciously created by development decisions. In an increasingly cost conscious word technical debt is sometimes introduced when less experienced developers are given tasks that should have been given to more experienced (and expensive) developers. The result can be debt by poorly written code.

However, there are almost as many ways to incur technical debt as there are stars in the sky. One example that comes to mind is the "killer application". These come along every few years and revolutionize the way computer code/systems/projects are accomplished. They have been amazing. They doubled or tripled our productivity, changed whole paradigms, made the rest of the world stand up and notice. The Borland C development environment, Lotus Notes, PowerBuilder, dBase IV, Java, just to name a few. Now many of them would be found on lists of technical debt. It is the way the world works.

The steam locomotive was an amazing piece of engineering. I live across from two sets of railroad tracks. I haven't seen any steam locomotives. But, I have seen plenty of PowerBuilder (PB) GUI screens. I am old enough to remember the first demo I saw for PB. When the power behind the product hit me it was monumental. This was a game changer. Soon knowing PB was a useful and marketable skill. Major systems were written with PB front ends. There was soon a bevy of competing products. The client server revolution was underway. I picked PB as an example because I remember its beginnings. I also remember why it is no longer the huge force it was. Does the phrase "fat client" ring a bell? Too much work on the client level that can be done better by an application server. A fat client produces code that runs on the client operating system. A Unix front end and a Windows

front end had to use different code even if they looked the same. Then came the crushing blow; web browser front ends and Java. Write once run everywhere became the order of the day. There is nothing particularly bad about PB. I used it merely as an example of the kind of technical debt that started out as a revolutionary idea and now, while it still works fine, is rarely chosen for new projects. The skill pool is shrinking and many of the former PB gurus have moved on.

Remember! Only you can prevent technical debt. Well,… Ok, You can't really prevent it either. But, make it an important part of your projects to try to dodge debt caused by short cuts and other avoidable ways of introducing technical debt into your environments. Plan for elimination of technical debt when it is discovered or knowingly created. As for those killer apps that ended up running out of steam, replace them as soon as it is practical. Everything has a cost. Keeping around technical debt might have a larger cost than you think.