HOME IMPROVEMENT TECH DEBT

Overview: Organizations would do well to apply the same thought processes homeowners do to tackling home repairs to resolving technical debt: Do it sooner rather than later.

In our common area at Compuware there is a large TV that shows home improvement shows (among other programming). I'll watch them sometimes when I eat my lunch. The common thread in these programs is the "surprise" of some bad wiring or plumbing, maybe termites… something that needs to be dealt with. The homeowners are then faced with a choice. They really want that updated bathroom but there is only so much time and money. Something has to give and… the repairs really need to get done.

I've never seen the homeowners turn down the necessary work in favor of the improvement they wanted. They want a safe, long-lasting house, and they understand it's more cost effective to take care of the repair right away than waiting.

Technical Debt
Why don't we apply that same thinking to our technical debt? Technical debt reflects the cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer.

This can happen when we prepare a prototype or sample code and test it out. We garner feedback, tweak it, get more feedback, tweak it, until it works. The code may not be the most efficient—perhaps we should do better SQL calls or write an API to get the information in one call—but it works. Very little thought, though, is given to how it could scale or how difficult it will be to maintain over the long term.

Why do we kick the proverbial can down the road? What makes our thought process different from a homeowner?

Just like them we want new features; we see the vision. What we don't seem to respect is that we are building on a weak foundation. We are adding more and more onto an application that is still using that limited, easy solution. Just like the homeowners, we need a compromise to make it work and provide for a safe infrastructure on which to build.

Here are some ways to consider tech debt to make it more visible so it's more likely to get addressed. First, and foremost, always consider resolving tech debt as an investment in the future of your application and not as a cost. Then:

--Consider the cost of delay. Maybe you feel you can delay resolving your technical debt, but you must balance the cost of how it will limit your future deliverables against what you would prefer to add to the product now. If you don't pay attention to technical debt, you won't be able to pull user requirements quickly through your pipeline, which will hurt you in the end.
--Don't be duped. People think tech debt isn't visible to end user, but it actually can be. Tech debt limits what you can provide in the future or, even more relevant to the end user, makes the application more fragile and prone to failure. Make sure this is clear as the decisions are being made.
--Do a rough sizing for resolving the tech debt. It is easy to overlook a fix if it isn't sized. Once you have rough estimates you are in a better position to balance doing the work now against other initiatives.

Here are a few ways to better understand the issues and provide quicker, more accurate estimates:

--Use Compuware Topaz for Program Analysis to review the logic graphically. It's said a picture is worth a thousand words and this is certainly true when you can graphically show an end user the logic and why it needs to be refactored first, before starting the improvements it needs.
--Use metrics and KPI's to understand the impact. There may be some code which is poorly written, difficult to update thus very rarely ever updated. Balance that against similar code which is frequently updated. Obtain metrics on how often the code has been changed in the past; its complexity (Halstead and McCabe Metrics); and how frequently it fails. Using these metrics will give you a better picture of how you are being affected by this code. A source of these metrics is zAdviser and Topaz for Program Analysis.

Just like a house needs a solid foundation to build improvements on, so do your applications. It is tempting to pass over needed work to provide some benefit now, but you need a long-term view of the health of the application to make sure it can support your plans.