Legacy: Old technology that frightens developers (part 2)

(Original creator: bolarotibi)

By Clive Howard, Principal Practitioner Analyst, Creative Intellect Consulting

Read part 1 here

Coding best practice is not always common practice

Then there is Object Orientated Programming (OOP), a concept designed to avoid many of the code issues that developers hate about legacy. The core tenets of encapsulation, inheritance and polymorphism mean that code should be highly re-usable, minimalist and easily readable. All great for a developer delving back into old code and needing to understand what it does, how and make changes to it quickly. Instead much OO code is bloated, buggy and impossible to understand. There are numerous reasons for this from bad class design, to poor coding and challenges of discoverability within tooling.

As I mentioned earlier teams do not set out to create bad applications and the first version is often great. With the right amount of work put into the design and coding the application can tick all of the right boxes. A developer having to go in and make a change or fix an issue will find it relatively painless. Then comes the final problem that few applications ever escape from or survive: Change requests.

Time and Change: two obvious partners in crime, but there is a third

From Rapid Application Development of the late nineties to Agile practices of today people have come to expect very fast turnarounds on change requests. Developers are under increasing pressure to deliver new features in shorter periods of time. Whereas it was once fine to roll out a new version every 12 months now the expected delivery time is every 6 weeks (or less).

No matter how good the original design and coding of the application when changes come there are bound to be issues with the original code that were not foreseen. Due to time pressure instead of fixing the underlying problem the developer simply works around it or hacks out a solution. The resulting mess has become known as technical debt. Essentially an increasing number of underlying problems being stored up in the code that will one day bring the whole application down: unless of course the debt is paid by rectifying those lurking problems. The popularity of Agile practices – though it need not lead to it – means that this scenario now plays out much quicker.

I spent 15 years designing and cutting code for applications and was guilty of all those sins listed above and more. Like most developers I'd prefer to blame it all on the time pressures applied by clients and managers. The truth is that developers cut corners for many reasons and they know it. Most of us know when we've contributed to technical debt although we still like to complain about what those who have gone before have done.

Is there is a better way?

The troubles with design and coding practices make for grim reading and any developer knows the story all too well. Legacy makes the heart sink and everyone wants to work on Greenfield projects. These projects are of course going to be much better and not have all of the horrors of legacy apps. Except after a while they do and today's Greenfield dream is tomorrow's brownfield nightmare. Does it always have to be this way?

The answer is of course no, it does not.

Revolutionary code of conduct: A path to redemption

The solution lies partly in the patterns and practices outlined above which need to be implemented correctly. However the development community also needs to be better at adopting new tools and frameworks. So many times we see code written using the latest version of a tool or framework that does not take advantage of any of the benefits that version brings. Despite the number of database abstraction offerings such as Object Relational Mapping (ORM) how often do people actually use them? How many teams make use of dependency injection techniques? Or switch to model driven development products?

Instead as a community developers largely take new versions of tools and continue to use them in old ways. Worse still is those who start a new project using out of date tools because of the comfort created by familiarity. Development teams need to be better at adopting new products and features and thinking differently about choices that are made early in development.

The world of mobile and Cloud makes these issues even more pressing as we're forced to adopt new ideas that challenge the old ways of doing things. Agile means that the time it takes to create legacy problems is greatly accelerated and so developers are more likely to have to deal with the ramifications of their decisions rather than leaving it to another team a couple of years from now. This is not an inherent fault in agile but too frequently good practice (regular sprints dedicated to cleaning up code for example) are lost in the frenzy to push out new features. Therefore it is time to look to new tools, frameworks as well as patterns and practices that will support this pace of change.

It is important to think about how developers can focus on the code that they have to write, keep it minimal, modular and easy to understand. Where can technology help to abstract away much of the plumbing so that applications are not tied to specific databases, infrastructures or User Interfaces? Modern applications will need to be able to move between data stores, on premise and the Cloud, desktop and mobile. These tools are out there but the community needs to embrace them.

This change is clearly the responsibility of the development team and the developers and architects themselves. Management and clients are not going to dictate which tool is used but instead developers need to investigate new technologies. Where appropriate they need to push for additional training and make the business case for it based on the quantifiable costs that legacy incurs. The community cannot continue working in the same ways and complain when it results in the same old outcomes. Agile, mobile and Cloud are forcing developers to work faster and across more platforms and technologies than ever before. Now is the ideal time to look again at what's in their toolbox.