The Future of Programming May Not Involve Coding (Part 2)

(Original creator: bolarotibi)

By** Bola Rotibi**, Research Director, Creative Intellect Consulting

Part 2 in a 2 part series: Read Part 1 Here

Developers: I'm better than you and I need my code

The most significant barrier to adoption has perhaps been the developer. Within the development community there can be a snobbery about writing code. A hierarchy of developers often exists in which those who are closest to the metal are better developers. So a C++ developer is far superior to a JavaScript developer. Therefore a developer who writes no code at all is perhaps not even a developer. Aside from this unofficial cast system there has always been a desire amongst developers to write code. Becoming a developer to not write code is like choosing to be a racing driver and not driving a car. Finally there is the niggling suspicion within the minds of developers for whom the low-code approach might be appealing is that a requirement will come along that cannot be fulfilled. If you are writing the code then this is highly unlikely to happen, any requirement can be met. This has been true in some cases such as the early days of .NET when developers discovered that something which was possible pre-.NET was not possible within the framework. Anyone who remembers the challenge of needing more than one form in an ASP.NET page will know what I mean. These issues, in many cases more emotional than real, have managed to drown out the many sensible reasons for adopting low-code development tools. Had software development been any other part of a business it probably would have shifted to this approach many years ago as the business case for it is overwhelming. But, within IT the old guard of developers has stemmed the tide and most development today, especially within the enterprise, still means many, many lines of code being written, tested, deployed and maintained.

The rise of a new breed of developer

Just as technology has changed so has the developer. The 2015 Stack Overflow survey found that the average developer does not have a computer science degree (almost half are self-taught) and has only been coding for 2-5 years. They are a product of the micro-service, API, framework, born-on-the-web world. These are not the Fortran or Mainframe or even Visual Basic guys of u. The most popular language in the survey was JavaScript with C#, PHP and Python beating out C++ and C. The main driver for this generation is speed and technology choices largely flow from that. This is highlighted by many of the new coding bootcamps that have sprung up. These are very short training courses in which people with usually zero coding experience learn how to develop in a few weeks. In order to operate in these timescales they choose technologies such as Ruby on Rails or Node.js simply because the more traditional technologies take longer to teach. Then there are the micro-service based applications often built on Platform as a Service such as Heroku or Cloud Foundry. These are also popular with the bootcamps because even quite complex applications can just be a few lines of code that knit together a few services. In this world some developers do not even see themselves as developers but merely curators of services. For them the issue of whether they are a "real" developer is irrelevant. Their aim is to create an application as quickly as possible. They would rather invest time in making it a great user experience. The applications that this generation have been creating are those that have disrupted many traditional industries or notably impacted society in recent years: Facebook, Netflix, Twitter, Dropbox and so on. The result is that now traditional enterprises and SME's are having to respond in kind. Innovation, agility and time to value are the buzz words in today's business with respect to IT. Developers are having to change and that includes languages, technologies and tools. We see this reflected in the products that their traditional vendors are producing such as Microsoft and IBM's PaaS platforms.

The future demands change

Then there is the challenge of mobile and Internet of Things. Whereas for a long time developers just had to worry about building a desktop application (often for a very specific OS) they now have to worry about the web, phones, tablets and potentially many other types of device. Codebases need to be able to serve many different clients. If every line of code is going to be written by hand then building an application that works on desktop, multiple phones and tablets and beyond is going to mean a lot of lines of code across numerous languages, technologies and tools. All of which must be tested, maintained and evolved. This will push developers, development teams and budgets to their limits. And with speed being an overriding factor the old ways simply won't work. Coding and creation of applications is opening up to a wider group. The advent of the citizen developer: regular folks building mobile apps in their spare time; or people at work building apps to aid their productivity; the iOS Gold Rush saw many designers begin writing Objective-C in order to push out iPhone apps. Everyone is getting in on the act and they are not playing by the old rules or caring about the developer hierarchy. This evolution began with the web when many people (such as myself) who had no development background or training became developers at the forefront of technology. That change has accelerated and continues to do so. This is being enabled by and driving the growth in low-code tools and technologies. As I said in the beginning, it is ironic that as governments push for the better training of software coding within the younger generation, the need to actually write code is rapidly diminishing. Many software application developers should look willingly to embrace this evolution not least because of the act of development is to build to deliver value. Code is a construction asset like bricks are in the building trade. In the building trade, ultimate value comes from the finished product and the way it is architected and the function it delivers. Of course the quality of the bricks also helps to cement that value but at the end of the day it is a commodity component. The analogy works for the business of programming with many tools now are able to auto generate the building block code to a consistent standard and quality: i.e. a commoditised function. Full and lasting value must surely then lie not in the construction of code but in the application that is devised and the way it is architected or modelled to deliver the function required. The resulting experience of engagement for the user is intrinsically tied to the value perceived and this is rooted more in the application model and architecture than the underlying code.