Editor: **Diomidis Spinellis**
Athens University of Economics and Business, dds@aueb.gr

Agility Drivers

Diomidis Spinellis

When the facts change, I change my mind. What do you do, sir? —*John Maynard Keynes*

A MANAGEMENT PRACTICE is mature when even government bureaucracies decide to adopt it. The March 2011 publication of the UK's information and communications technology strategy marks this moment by advocating that "the application of agile ICT delivery methods [...] will improve government's capability to deliver projects successfully and realise benefits faster" (www.cabinetoffice.gov.uk/content/government-ict-strategy). This begs a question: Were we misguided in the decades when we advocated stringent control of requirements and a tightly milestone-driven development process? Interestingly, no—we were right then, and we're right now. Things have changed, which is why we can today smugly apply agile practices and reap impressive dividends.

Numerous new factors drive agility by increasing our productivity. Our growing ability to swiftly put together sophisticated software affords us the luxury to listen to our customers, to try out new things, to collaborate across formal boundaries, to make mistakes, to redesign as we move along—in short, to be agile. Knowing these factors helps us realize when we can afford to be agile and when not. (Hint: agile development of a plane's flight control software from the ground up is still not a good idea.)

Post your comments online by visiting the column's blog:
www.spinellis.gr/tools

Technology

On the technological front, two key drivers come from the system software we can use: the operating system and database management systems. Although I admire the conceptual clarity and simplicity of 1970s-era Unix, the truth is that a modern operating system distribution offers many facilities that transform past major projects into a weekend job. These facilities include standardized networking, interprocess communication, graphical user interface support, sophisticated typeface and 3D rendering, and load balancing across many CPUs. Database management systems (perhaps thankfully) have not advanced as much on the functionality they offer, but they have become ubiquitous. If you're carrying a smartphone, you probably have one or more relational database management systems (RDBMSs) running inside your pocket. Therefore, if a program requires structured access to persistent data, it can readily call a database; developers no longer need to waste time laboriously crafting bespoke data formats and access methods.

One level up come libraries. Gone are the days when each self-respecting program contained one or more sorting subroutines and other "utility" functions. Now the Java platform comes with no less than 3,776 classes and interfaces. From an `AbstractAnnotationValueVisitor` to a `ZipOutputStream`, it's all there. Other platforms, from Perl to .NET, are similarly feature-rich, making many software development tasks a simple matter of gluing together existing classes. If a class isn't part of the platform, it's likely to be available as an add-on component. If the task at hand requires a large dataset (say, the Earth's map) or relies on constantly changing facts (think flight information), someone will have probably developed a corresponding Web service we can use.

This brings me to application interoperability. After a few false starts (remember DCOM and Corba?) we seem to have sorted out how one application can readily communicate with another when they're a continent apart. Without laborious prearrangements, the Internet and RESTful interfaces provide us hassle-free access to data, from the movement of stocks to that of tectonic plates. This paradigm is so prevalent that public-facing applications failing to provide a Web service interface are perceived as crippled. Libraries (again) make using these services a joy. With less than a screenful of code, I can obtain a book's cover image from Amazon.com, send out a tweet to thousands of readers, and even use a SOAP interface.

A standardized presentation layer for our applications is also slowly evolving. While Ajax is by no means a perfect application development

...continued on p. 95

0740-7459/11/$26.00 © 2011 IEEE

…continued from p. 96

framework, it offers a neat way to run applications from any desktop, and the establishment of HTML5 is likely to improve things even further.

Programming languages have kept pace with our requirements, providing impressive features that used to be the subject of academic research—threads, nimble type systems, duck typing, modules, metaprogramming, and support for object-oriented and functional programming are only some of the goodies that let us churn out more sophisticated and powerful code each day. Further pushing the productivity envelope are scripting languages, which allow us to express in a couple of lines what used to be a student's final-year project.

But all these impressive technologies I've outlined here come at a cost. They require resources to develop, effort to master, muscle to run, and patience to endure their (inevitable) shortfalls. Yet even in this regard, we're blessed with many countervailing forces.

CPU power clock rates and memory capacities have increased by many orders of magnitude. The first IBM PC came with a 4.77-MHz CPU, a 360-Kbyte floppy disk, and 64 Kbytes of RAM. During the past decade, having satisfied the needs of the most demanding office worker, the resources available on a modern desktop now provide us the power to run all the agility-enabling technologies I've outlined and also the tools to conquer their complexity. These include IDEs that provide refactoring support and online help for all methods of Java's 3776 classes, optimizing compilers and runtime systems that make our readable code run efficiently, unit-testing frameworks that take the fear out of refactoring, light markup methods like Javadoc that finally materialize literate programming, and languages with an interactive top level that promote experimentation and bottom-up development. Tools such as wikis, instant messaging applications, version control systems, and bug trackers smooth the collaboration with colleagues across the hall or around the world. In addition, various cloud platforms allow the rapid and scalable deployment of user applications and developer facilities at a reasonable cost.

… and the Environment

Technology isn't the only force driving agile development practices. Social factors are equally important. Software developers are no longer retrained mathematicians or physicists, but computer science graduates who are also increasingly participating in cross-disciplinary programs. It's only natural to expect more from such people, including the ability to closely collaborate with clients on solutions that address their real needs. Changes in the management culture, with flatter hierarchies, less formal reporting, and emphasis on teamwork, also help shift the software project management focus from the milestone-based handover of responsibility into collaboration with colleagues, in-house business and marketing departments, and customers.

Moreover, developers can tap into a variety of resources for help. Programming-by-googling is becoming the standard way of writing code as developers use the Web to find code examples, suggestions for handling cryptic error messages, peer advice from professional forums, and, most importantly, open source code. This last resource is an agility enabler through the vast number of readily accessible, useful components coupled with their (sometimes deceptively) hassle-free procurement and licensing. Do you want to add PDF generation or speech output to your application? You can download and link such components to your code in a matter of minutes.

Then come two mutually opposing forces. On the one hand, the ubiquitous availability of user-friendly, polished, and versatile consumer-oriented IT applications has increased the expectations of all users. They want their ERP to be as fast as Google's search and its interface as intuitive as their iPhone's. That's a tall order for a traditional development method and a clear call for agility. On the other hand, these and other consumer applications are lowering users' expectations regarding features and reliability. Millions are discovering that sending 140-character messages over a platform that occasionally overloads (indicated by a whale lifted aloft by birds) is a perfectly acceptable, even addictive, way to communicate. Agile practices are ideal for swiftly delivering such good-enough solutions in an environment that can tolerate the occasional failure. Finally, the ubiquitous availability of IT infrastructures makes the scale of today's deployments considerably larger. IT support that used to target accounting and management is today covering all the organization's employees and customers. This increases the risk of centrally planned, rigidly executed projects, further driving the need for agility.

I'm sure that by now you're convinced that the landscape of modern software development has changed vastly over the years in ways we haven't yet fully appreciated. It's therefore natural to adjust the way we develop software. Where agility drivers are present, it's a shame to preserve the status quo. Realizing that better outcomes are possible, we must adjust our development processes, demand more from our software suppliers, and develop in-house capacity to organically grow applications and services that will delight and even captivate our users and customers.

DIOMIDIS SPINELLIS is a professor in the Department of Management Science and Technology at the Athens University of Economics and Business. Currently he is serving as the Secretary General responsible for information systems at the Greek Ministry of Finance. Contact him at dds@aueb.gr.