

waters

NOVEMBER 2006

Avoiding the Deluge

How to move beyond the waterfall development model. By [LUKE FLEMMER](#)

Take a moment to consider the defining challenges of your job. If “responding to change” isn’t one of them, you’re either reading the wrong magazine or not trying hard enough.

Nowhere is this more apparent than in front- and middle-office IT organizations. Tasked with building trading and processing systems for an ever-increasing range of products in an aggressive regulatory environment, these organizations rise or fall on their ability to respond rapidly to dynamic requirements while maintaining stability. In the face of this onslaught, savvy development managers are embracing a new way of building software far better suited to these challenges.

Most software today is built using a variation of the “Waterfall Model,” so called because preceding steps “flow down” to subsequent ones. This model reflects a desire to make software development conform to other traditional engineering disciplines, such as civil and mechanical engineering. In these disciplines, a huge premium is placed on upfront specification and design. When you’re building a bridge, for example, there’s a lot of merit to this approach. You don’t want to get halfway through the project only to decide you prefer the bridge six feet to the left. These projects have such stringent processes for change management because the construction materials and processes do not lend themselves readily to change.

For this reason, the Waterfall Model delivers two very attractive capabilities: Projects can be priced accurately on a fixed-bid basis and scheduled to firm



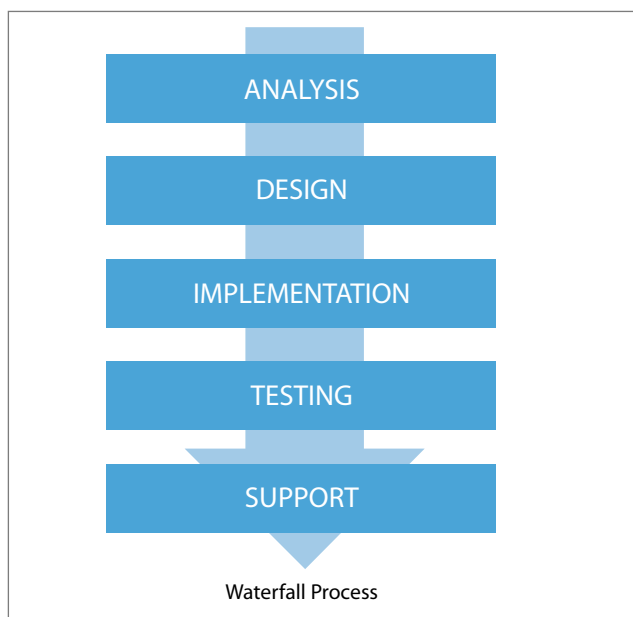
milestone dates. Although there are frequent exceptions—have you tried driving through Boston’s Big Dig recently?—the model makes a good deal of sense for these types of projects.

It is easy to understand why the Waterfall Model was adopted with enthusiasm bordering on fervor. Unfortunately, building business software—especially in an arena as dynamic as finance—is very different from building bridges. Not only has the Waterfall Model failed to deliver expected business benefits, it has actively hurt productivity in many IT-driven organizations.

The Threat of Change

The whole point of software is that it can be changed. That’s the “soft” part. You don’t need to change the physical computer, or the “hard” part. You can change its behavior just by typing in new code. This is the key enabler of the remarkable velocity of modern business. The Waterfall Model, however, is all about controlling and limiting change. In a best-case Waterfall scenario, you can capture a complete and comprehensive set of requirements up front, and those requirements never change over the course of a project. The more they do change, the more you must write up new requirements, and the more the plan must be revised. To this style of project management, change is like Kryptonite.

A new approach to building software is needed: one that sees change as a necessary and desirable aspect of the development process.



So it's not surprising that a somewhat adversarial relationship frequently develops between the client and the development teams. Clients are the source of change, while project managers must resist those changes as best they can. When the project is finally delivered after six months or a year, it is all too common to find that the needs of the business have evolved substantially.

Anyone who has ever built software for a trading desk knows this approach is flawed. Traders come to work to make money and seldom view time spent talking to IT people as contributing to that goal. If you are lucky enough to get their feedback, you will receive specifications ranging from the expansively high level—"It needs to be able to lay off DV01"—to the incredibly granular level—"You know that button in the XYZ app? Make it like that." Creating a comprehensive and complete specification from this information is non-trivial.

Even if you have a comprehensive specification, the problems are just beginning. Markets are dynamic, and volatility is directly tied to profit. Big spreads and interesting trading opportunities arise from new products and strategies. Over time, all markets become more efficient, and these opportunities evaporate. For this reason, traders must respond rapidly to market changes—be they new sources of liquidity, new derivatives or new quantitative strategies. The specification is not a clear blueprint, of which occasional minor requests are made. Rather, it is a snapshot of a dynamic and evolving set of requirements. Trying to meet the requirements of a dynamic business with a static process isn't just inefficient, it's actively damaging.

Over the last few years, some have realized that a new approach to building software is needed: one that sees change not as something to be resisted, but as a necessary and desirable aspect of the development process. This approach acknowledges that building software is not at all like building bridges. Rather, it is about narrowing the gap, at any point in time, between what the business needs and the software that it has.

The New Agility

As with most things, this new approach has produced its fair share of buzzwords. The term most commonly used to refer to the adaptive, lightweight nature of this process is "agile." Another common expression is "iterative development," which is more descriptive

What these approaches stress is a process with quick turnaround time, one that doesn't just tolerate change, but actually encourages it. The concept of "iteration" is paramount. Software is delivered at regular intervals—often as short as two weeks and seldom longer than a month—rather than on a single red-letter day.

The software that is delivered should be tested and usable—in other words, of "production quality." Early iterations will simply be sparse on features. As the iterations progress, the features added will be precisely those features the business has decided to prioritize for that iteration. In other words, the project is no longer beholden to some original specification document, but rather allowed to grow in a direction set by customers. For teams able to adopt this way of working, the rewards can be dramatic: vastly reduced time to market, lower project risk, greater transparency and, critically, less divergence between current requirements and current capabilities.

Ironically, this less proscriptive, adaptive process actually produces more reliable software, for a couple of reasons. In a traditional Waterfall Model, testing is relegated to the project's end. With such projects, there is almost always timeline slippage, and as the project progresses, pressure mounts to deliver a finished product, which reduces the time allocated to testing and, therefore, leads to less tested software. In agile development organizations, testing is woven into the process and each completed iteration is tested. Put simply, testing takes pride of place in an agile process.

Additionally, more of the delivered code is actually used. A big problem with the traditional approach is that, because change is so expensive and unwelcome, analysts try to imagine all the possible functionality that might be required. These predictions are seldom accurate—one study by research firm The Standish Group International found that 64 percent of features developed in software projects are rarely or never used. In addition to the colossal waste of resources implicit in building features people don't use, it leads to bloated applications with large amounts of barely tested code. The iterative process produces far less extraneous code because it focuses on features users actually want. These features are then extensively tested by the users who have requested them.

In our experience, London banks are far ahead of those in New York in terms of the adoption of this approach. Although some forward-thinking groups within the large US banks have adopted these principles, the word "agile" is far better understood, and adoption far more advanced, in the UK. We hope to see a continued uptake in the US financial community because we believe this development methodology will lead to better, cheaper and faster software and more competitive organizations. ●

Luke Flemmer is a managing director with Lab49, a consultancy that develops trading systems and consults on software design and development methodology.