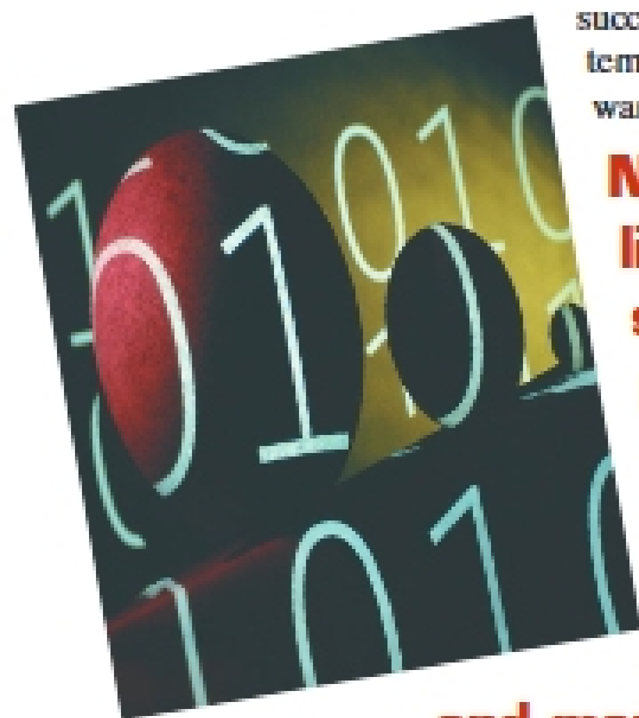
Sizing Up Today's Lightweight Software Processes

Granville Miller



New lightweight software processes promise faster time to market and more efficient development. Find out which one is right for your project and team.

There is a buzz about the new lightweight software development processes. These processes are certainly capable of creating successful software systems faster. As a software developer, which one should you choose for your project? Indeed, today's software development processes bring interesting dynamics to the projects to which developers apply them. But should process drive the project or should the project determine the process? I examine one characteristic of these processes, which can help you decide what process is best for you.

A VIEW OF OUR INDUSTRY

A couple of years ago, I sat on a panel for the Triangle Smalltalk Users Group on the differences between the various object-oriented languages. Having worked with both Smalltalk and C++, I was asked to be the C++ advocate. The moderator posed a question to the panel regarding the use of C++ and Smalltalk on large projects. Could the languages scale? Having built a C++ system with thousands of classes (with a very good team of dedicated individuals), I can honestly say that C++ is scalable. However, the approach used to build large C++ projects can be very different from that of its Smalltalk or Java brethren.

If you follow today's industry trends, you will realize that software developers have made tremendous progress since the days when they used to argue about which language was the absolute best. We now acknowledge that each language has its definite merits, and have even added a new member—Java—to the ranks of object-oriented programming languages. Our discussions of methodology have yielded a standard in the Unified Modeling Language (UML). However, advancements in these areas have simply moved the question to the often time problematic area of software development processes.

A question of development process

Several contenders direct the field of software development processes, including the Rational Unified Process, Extreme Programming, Crystal, and Feature-Driven Development. (The "Resources" sidebar lists sources for detailed information about these processes.) Each of these processes prescribes a set of activities that may (and in some cases, must) be used to understand the system's requirements and transform them into software. Each process has its merits; each process could be the right one to successfully deliver your project.

To understand the dynamics of these processes, let's look at a generic software development process. Developers think of software development as consisting of five phases: requirements, analysis, design, construction, and test (in that order). The three phases before construction are called the *front end* of the software development process. The two final phases, construction and test, form the *back end*.

Via the activities that a process prescribes, it defines how much time developers spend (in relative terms) in each phase. For example, if a process requires several different requirements specifications, context diagrams, and use-case models, you will need to commit a significant amount of project time to the requirements phase. On the other hand, if your process prescribes no requirements, analysis, or design activities, development will spend the majority of time in the construction and/or test phases.

1520-9202/01/$10.00 © 2001 IEEE

A process that prescribes the majority of its effort and creativity in the first three phases (requirements, analysis, and design) is called a *front-loaded* process. The goal of the ultimate front-loaded process is to construct a manufacturing process where software is mechanically churned out. The Rational Unified Process is considered by many people to be front loaded.

The opposite of a front-loaded process is the back-loaded process. As you might expect, most of the effort in a back-loaded process is spent in construction and test. With the ultimate back-loaded process, requirements, architecture, and design occur in the process of constructing the system in a somewhat spontaneous manner. The trend today is more toward back-loaded processes, such as Extreme Programming.

Finally, a process that strives to spread its effort evenly between the back and front ends is called *balanced*. Balanced processes do not always create an exact balance. However, they define some activities on the front end to balance the necessary development on the back end. These processes use a think-first-then-act-quickly approach. Crystal and Feature-Driven Development are examples of balanced processes.

CHOOSING THE RIGHT PROCESS

A balanced process sounds like it would be the ideal choice for everyone, however, this is not necessarily the case. For example, in telecommunications, standards provide the requirements for many systems. Much of the "what" is mapped out by the standards and even a bit of the "how." Construction is the actual mechanical process of bringing these standards to life. (Although optimizing system performance, a characteristic often used to obtain competitive advantage, is rarely mechanical.) These types of projects map well to front-loaded processes.

How about the frequently seen project whose requirements are as vague as "OO is good, let's build something with it!" Writing a full-blown use-case model around this requirement could be a

Software Development Processes

A software development process defines a set of activities necessary to deliver a software system. All activities culminate in an artifact (or refinement of an existing artifact). The artifacts represent the result of working on the activities. The goal of a software development process is to facilitate people in their quest to effectively produce a software system.

Extreme Programming (XP)

Extreme Programming has been the pioneer in the modern movement toward lightweight processes. XP emphasizes a single major artifact, the code itself. This process uses 3" x 5" cards to capture requirements in user *stories* and design via CRC (class, responsibilities, and collaboration) cards, the minor artifacts of the process. XP is much more than user stories, CRC cards, and coding, however. Testing frameworks and innovative practices such as pair programming (working in groups of two people) make XP an interesting addition to the field of software development processes.

Crystal

Crystal is a lightweight process that contains 20 artifacts. This might sound like a heavier process than XP but most of the artifacts are informal and can take the form of "chalk talks" (working problems out on a chalk board), conversations, and e-mails. Of these 20 artifacts, only the final system, the test cases, and the documentation are formal. Crystal divides its artifacts into levels of precision (20,000-foot view, 5,000-foot view, 10-foot view) to allow developers to focus on their objectives.

Feature-Driven Development

Feature-Driven Development is an incremental approach that uses as few as four artifacts (feature list, class diagram, sequence charts, and code). The FDD process focuses development using two-week iterations to show quick tangible results. Among the contributions this process provides is a semantic-based class diagram template—called the domain neutral component, which differentiates types of classes by color—to aid class designers in developing a domain model.

Rational Unified Process

The Rational Unified Process collects many of the best practices of OO analysis and design to form a process framework with 38 different artifacts. RUP is not generally considered lightweight, although a lightweight configuration called dx ("xp" turned upside down) exists. Of course, not all 38 artifacts are required in either RUP or dx. In fact, the process framework is configurable to as few as two (use cases and code) artifacts. However, the general RUP-based process uses quite a few requirements, analysis, and design artifacts because its developers based this process on the activities of the OOA/D movement.

tremendous challenge. Yet, one of the most popular integrated development environments (IDE) was created from this very idea. A back-loaded process was necessary to allow the people who submitted this requirement to "see" the power of object technology.

There are many factors that determine which type of process would best suit your project. Some factors to consider include size, domain, team member personalities and experience levels, quality requirements, uncertainty, programming language, and mission criticality. Some projects seem to dictate obvious choices in every case. For example, it is the general perception that airplane autopilots should be created with front- loaded processes.

Front-loaded processes

Certain types of people favor one approach over another. People who like to plan things before working on them tend to prefer the front-loaded processes. Methodically preparing and communicating what needs to be done can save a lot of time and frustration later on. On the other hand, a possible drawback of the front-loaded process is anti-pattern, "analysis paralysis" (William J. Brown and colleagues, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley & Sons, New York, 1998). For these projects, a front-loaded process can waste time on information that cannot be known or details that cannot be discovered in the early phases of the software development process. Additionally, delivering the system's initial versions takes longer because the early phases take time and yield models but not a tangible system.

Back-loaded processes

Action-oriented people tend to prefer back-loaded processes. These processes allow them to sink their teeth into the code right away. They perform requirements analysis and system design in the process of creating the system itself.

Back-loaded processes deliver tangible results by delivering the system's initial versions quickly. The potential downside includes a possible lack of a long-term, sustainable architecture and spaghetti code—code that has no overall structure.

Additionally, this type of process can create situations where a vice president, after seeing one of those initial versions, suggests shipping it.

Balanced processes

Balanced processes are suited for projects that are well defined in one area while completely ill defined in others. For example, the business process of order fulfillment is certainly well understood, however, applying new enterprise resource planning technology can require quite a bit of back-end work before developers can successfully understand its use on the project.

Balanced processes are generally well suited for development teams made up of several different personality types. A balanced approach can provide the compromise necessary for the strategic thinker to do some planning, analysis, and design before the action-oriented developer jumps into coding. The drawback of a balanced process is that it may never truly satisfy either of these types.

Putting your favorite process into one of these three buckets may not be entirely necessary. Many of today's software development processes are actually process frameworks. They suggest a set of activities but allow you to configure the subset that meets your needs. A few allow you to change from front- to back-loaded to balanced as the needs of your project change.

Any process may need adaptation to meet the needs of your organization and project. But don't get lost in process customization. Completion of the project—not the process—is what ultimately determines success. Processes are means to an end.

Ultimately, you are assigned the task of ensuring that your project succeeds. Many processes may be adequate when combined with a good, veteran software development team.

Resources

- *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, William J. Brown and colleagues, John Wiley & Sons, New York, 1998.
- *Crystal Clear: A Human-Powered Software Development Methodology for Small Teams* (to be published), Alistair Cockburn, Addison Wesley Longman, Reading, Mass., 2001. A preliminary version of this book appears at http://members.aol.com/humansandt/crystal/clear/.
- *Extreme Programming Explained*, Kent Beck, Addison Wesley Longman, Reading, Mass., 2000.
- *Java Modeling in Color with UML*, Peter Coad, Eric Lefebvre, and Jeff DeLuca, Prentice Hall, Upper Saddle River, N.J., 1999.
- *The Rational Unified Process: An Introduction*, Philippe Kruchten, Addison Wesley Longman, Reading, Mass., 2000.