

# Development Management & Practices

## 1. Faster Development

We're deeply concerned with *fast* development. We pay attention to development effort when designing [information security](#), [enterprise architecture](#) (persistence, security, management, etc.), user interfaces, or whatever. If it can't be realized within time and budget constraints, then a design is useless.

We also note that all time is not equal. Efficient and effective development processes multiply development capacity of everyone on the team. Effective team development management practices similarly reduce the effort it takes to complete a task - by keeping every team member focus on what needs to be done to complete the project.

## 2. Development Practices

Through experience, we've compiled a set of development practices that increases developer efficiency. Each practice attempts to either automate a time-consuming task or avoid it. Concentrating on common, time-consuming tasks by reducing the associated effort necessarily reduces development time. Some of the development practices each member of the team can use to work more efficiently include:

- *Source code repository + ANT*. Being able to share code and components is key to co-operative development. A repository alone is not enough: an automated build and deployment process is needed to share re-usable components. The Apache ANT tool is the best build and deploy toolkit for general developers we've found thus far.
- *Use different development tools for different tasks*. Some tools are really great at helping to build UIs, while others help with XML development, and still others really speed things up when doing general-purpose development. Use the right tool for the right job, obviously. A key corollary: avoid forcing each developer on the team to use the same development environment.
- *Prototype and build tests*. A throwaway proof-of-concept piece will help you build the final version. The proof-of-concept class or component should be thrown together in the simplest way possible to help you prove you assumptions, and flesh-out required behavior. A repeatable, automated test will guarantee your finished product fulfills the same requirements.
- *Comment and document*. Every non-trivial method and callable interface needs to have a

detailed behavioral description, with pre-conditions and post-conditions if appropriate. The code is *not* the documentation, despite what some people say. The documentation is the documentation, and it defines what the code is supposed to do.

- *Re-use existing code and components.* Its amazing how many developers will spend weeks of effort in order to save themselves hours of research. Only build a new class when no existing one can do an adequate job.

### **3. Development Management and Planning**

We are great believers in [SCRUM](#) development management, having used its practices to efficiently complete projects. The SCRUM process attempts to keep all members of the team focused on the end-goal: completing a set of requirements within a fixed amount of time.

The SCRUM process is an agile methodology concentrating on management and planning. A project customer and team members meet regularly to summarize and review tasks to be completed within a short amount of time, known as a "sprint". Team members estimate effort for each task in the customer's list. The customer prioritizes the tasks; the tasks accepted for development include all the tasks that can be completed within the sprint's time limit.

The key is to invigorate each developer by allowing him to select his tasks (within reason), and basing time budgets on his estimates of effort. The specific terminology and practices of SCRUM are not crucial, though experience has taught that they do work. What is crucial is that the team is focused on completing the requirements that the customer needs to be completed, and that through self-selection the team tends to be much more efficient at completing tasks.