Agile Development Checklist

The purpose of this article is to define a set of ideal practices for an agile software development project. The idea for this article came to me after discussing CMMI-type processes and realizing that there is no agile equivalent. I encourage you to leave comments about this article using the discussions module at the bottom of this page. Please note that the practices listed are the practices that I believe are essential to a good agile development project; they do not necessarilly have anything to do with being agile. I have tried to list the practices in descending order of importance.

**Practice 1 - Aggressive Refactoring In my opinion:** Refactoring is the most overlooked skill for a software developer. A well refactored application has a much higher value to the project sponser than does a poorly refactored application. The most common sign of code in need of refactoring is excessively long methods. I try to keep methods to less than 100 lines. Other common code smells are misleading or meaningless variable names, and code duplication. Static code analysis tools, such as FxCop, can provide a useful measure of code quality.

**Practice 2 - Testing:** Firstly, their should BE some developer testing. All code that is written should be testable and have tests written for it. It is acceptable to modify your program to facilitate good testing. I believe that the traditional testing terms; unit tests, integration tests and system tests have become outdated. Instead I prefer the terms developer tests, functional tests and non-functional tests. Non-functional tests are things like performance testing, functional tests are tests that the customer cares about like use case tests or business transaction tests, and developer tests are everything else that the developer needs to test to prove to herself that the code is correct. As much testing as possible should be automated and run as part of continuous integration. If code coverage analysis is included in the automated testing it provides a nice indication of the health of the system at any point in time.

**Practice 3 - Automated Build and Deployment:** The project should have an automated build, an automated deployment and ideally automated testing. In the optimal situation a developer can click a button and the build process will build the latest source, deploy, test and report on the result. Automating these processes not only saves time but it also eliminates a huge number of bugs and time wasters.

**Practice 4 - Continuous Integration:** If a project has automated build, deployment and testing then continous integration is really just a simple matter of automating the kick-off of that build, deploy test cycle. Every checkin should result in a new build and test, on a separate build server. The results of this should be reported to every team member and it should be established team practice to immediately fix the build. A working build is everyone's top priority. People should not be made to feel bad if they break the build, as this decreases their courage.

**Practice 5 - Source Control:** A source control system should be used to store all project artifacts including: code, non-code documentation, build scripts, database schema and data scripts, and tests. Code should not be checked into the build until it compiles, has tests and is passing its tests.

**Practice 6 - Communication Plan:** There should be a defined, direct communication channel between the developers and the customers. This can be (best to worst): on demand face-to-face communication, daily or weekly face-face communication, contact phone numbers, instant messaging,email mailing list, intermediary (BA or PM). These communication channels can and should be combined.

**Practice 7 - Task Tracking:** There should be a defined technique for recording and prioritizing development tasks and bugs. The system should make it possible to assign responsibility for tasks to individuals. If tasks are tracked against estimates then the estimate should be performed by the person who will do the task.

**Practice 8 - Self Documenting Code:** Code comments should be subject to the same quality requirements as the code itself. Everytrhing possible should be done to ensure that no other technical documentation is required. When non-code technical documentation is required it should be subject to the following restrictions: referenced from the code, always up-to-date (change when the code changes), only one version per baseline, stored in source control.

**Practice 9 - Peer Review:** There must be some form of peer review, such as code review of pair programming. If the developers are subject to performance reviews then the peer reviews they do should be an input to that process. This helps to avoid the temptation to approve everything to avoid confrontation. Make sure that the reviews are for quality, not just correctness.

**Practice 10 - Work-in-progress:** A working version of the latest iteration should always be available for customer feedback. The advantage of this is that customers see very quickly when something has been developed different to what they had in mind. Shortening this feedback loop decreases the cost of change.

**Practice 11 - Feedback Mechanism:** There should be a defined mechanism for project team members, including the customer, to provide feedback on the projects processes. A suggestion is to hold a short meeting at the end of each iteration.