Unit Tests

As an agile paradigm, it is imperative for systems developed in XP to be as robust as possible in order to avoid lengthy delays caused by uncaught errors when development on the next iteration begins. The longer an error continues undetected, the costlier it becomes to fix the further we are in the development cycle. XP’s testing principles adhere to minimizing stalls in development and minimizing costs by rectifying errors as early as possible. This is all done during the unit test.

In XP unit tests are written simultaneously with the code. By writing unit tests before the code, you may miss some test cases that become necessary depending on your code implementation. Similarly, writing unit tests after the code is not a practical approach because you have to mentally keep track of what tests are needed at which sections of code, and it is highly likely a test case will be forgotten. When writing test cases simultaneously with the code, test cases are generated when they are immediately necessary (when writing methods for an object class for example). The lists of test cases are run all the time, and whenever a new test is formed, it is added to the list. When tests are performed rapidly, the location of an error is much easier to find – in most cases it would be located only within the most recent changes to the code. This is analogous to the strategy new developers use where they compile their code frequently whenever changes are made. The list of test cases grows extremely quickly, and that leads to why XP demands all tests be automated.

When unit tests consistently pass with 100% success, it is easier to integrate components together. In the long run, it will make newer iterations easier to develop since it has a stable codebase. Occasionally an acceptance test may reveal an error – this indicates that a test case was missing. Uncovering the missing test case may lead to new tests cases also. In XP, a product is not released until all unit and acceptance tests have 100% success rates.

Ron Jeffries frequently used the phrase “Everything That Could Possibly Break” (Extreme Programming Installed) for guidance in deciding what to test. There are some methods that are simply indestructible (such as accessor methods) that don’t need to be tested. Deciding what to omit from testing takes experience, but Jeffries claims “There are more things that couldn't possibly break than you might imagine” (Extreme Programming Installed, pg. 153) and an adequate amount of time might be saved from not having to write unnecessary test cases.

Analysis

The strategies of unit testing are not unique to XP; they are common testing strategies employed by various methodologies to ensure a high quality product. Even non-iterative methods such as the waterfall model employ the same testing strategies. In order to reduce costs, it is important to resolve an issue as early as possible after it is introduced, and all modern software models have accommodated this into their practices.

XP development differs from iterative development in one aspect however: the error tolerance. XP strives for quality and does not allow a system release until tests pass at 100%, but in iterative development there are situations where a product is released pre-maturely because of deadlines due to marketing strategy. The errors would then be corrected in a future update. This leads me to believe that XP is ideal when the development team is working one-to-one with a client. In contrast, traditional iterative development is more practical when your consumer base is shared among other developers. When the release of a system is not only driven by quality but also competitive edge, some XP practices may have to be broken in order to gain the advantage.