ATDD: Not as Optional as You Think

By Jennitta Andrea

As a practitioner, I prefer to define agile by its visible outcome, rather than focusing on the collection of practices performed by a team. I define agile as: “delivering running tested features to the user community on a regular basis to maximize business value and to respond to timely feedback with strategic product adaptation.” [1] I believe it’s healthy for different agile projects to make different decisions about their practices, as long as the outcome is achieved and sustained. Successful projects understand that project context (criticality, complexity, duration, team composition, etc.) is key to making good decisions about their agile process.

That said, it concerns me that far too many teams treat the acceptance-test-driven development (ATDD) practice as optional.

What is ATDD?

ATDD (a.k.a. story-test-driven development, functional-test-driven development, behavior-driven development, etc.) is the practice of expressing functional story requirements as concrete examples or expectations prior to story development. During story development, a collaborative workflow occurs in which: examples written and then automated; granular automated unit tests are developed; and the system code is written and integrated with the rest of the running, tested software. The story is ”done”—deemed ready for exploratory and other testing—when these scope-defining automated checks pass. [2]

Why should we practice ATDD?

Software processes are like an ecosystem; all of the components—activities, practices, roles, workflows, tempo, tools, and work products—depend upon and enable each other in complex and unexpected ways.

ATDD enables:

Trustworthy specifications—ATDD requirements specifications are trusted because they are concrete, thus bringing areas of ambiguity and inconsistency to light very early. They are living (automated) documents that will not get out of sync with the code. Whenever the code is changed, immediate feedback is provided about mismatches between the two. This level of trust exists for the lifetime of the system, facilitating the transition to operational support for maintenance and enhancements.

Short iterations—Given the above definition, the key to an agile project is incrementally building the system, feature by feature. Everything—requirements, design, development, integration, testing, and review—is condensed in order to fit an entire cycle into a short, fixed-sized iteration (typically two weeks). ATDD helps the team maintain a concrete, clear focus and provides an unambiguous definition of what ”done” means for each piece of work in the iteration.

Incremental development—Working incrementally means different people will revisit and integrate the same code many times. This approach is not practical unless there is a safety net of automated checks to ensure unintended side effects have not been introduced. ATDD builds such a safety net at both the business- and technology-facing levels. [3]

Enriched tester role—In the early days of agile, it was thought that ATDD would make testers obsolete. Early agile projects found out the hard way that the tester role is needed more than ever. Ironically ATDD makes the tester role much more enriched and valuable. Automated examples and unit tests take care of the ”checking,” guaranteeing a stable system for the testers to perform their creative, skilled, and value-added work of ”testing.” [4]

Highly testable system—ATDD produces a highly testable system. Having examples and unit tests drive development ensures that the system can be accessed and validated from any possible angle, facilitating future automation as well as manual testing and operational diagnosis.

But, there is a cost to achieving these ATDD benefits. ATDD depends on collaboration, skill, and discipline. Examples are

the result of the collaborative effort of many different roles (subject matter experts, analysts, testers, and developers). Skill and discipline are required to make examples locatable, readable, maintainable, and efficient. Discipline is needed to consistently embed examples and unit tests into the development and integration activities, and to apply design for testability strategies and patterns. Selecting the appropriate tool for expressing, automating, and maintaining examples is a critical success factor. [5]

What if we don't practice ATDD?

An ecosystem shows signs of distress when balance is disrupted; having too much or too little of something can have serious consequences in other areas. All of the benefits described above quickly unravel if ATDD is missing. In particular, incremental story development puts high demands on the tester role due to constant demands to manually regression test an ever-changing system. If you've experienced or heard of a "bad" agile project, chances are good that the way they performed ATDD (if at all) is one of the root causes of their problems.

What's next?

On August 8, 2010, a community of practice retrospective was held to share the successes and reflect on the challenges a variety of different teams have experienced with ATDD. [6] The goal of this workshop was to increase overall community learning so that we can effectively focus our energy on individual, team, tool, and AAFTT-program improvements. The key findings from this event will be summarized in my next contribution to Iterations (October 2010).

References

1. Inspired by Jeffries, 2004. "A Metric Leading to Agility." http://www.xprogramming.com/xpmag/jatRtsMetric.htm
2. Andrea, 2009. http://www.jennittaandrea.com/wp-content/uploads/2009/03/beautiful_testing_ch14.pdf
3. Marick, 2003. http://www.exampler.com/old-blog/2003/08/21/
4. Bolton, 2009. "Testing vs. Checking." http://www.developsense.com/blog/2009/08/testing-vs-checking/
5. Andrea, 2008. http://www.jennittaandrea.com/wp-content/uploads/2009/03/someassemblyrequired_bettersoftware_jan2008.pdf
6. AAFTT, 2010. "AAFTT Program's 4th Annual Workshop: ATDD Community of Practice Retrospective." http://aaftt2010.crowdvine.com/

About the Author
Jennitta has been a practitioner (analyst, tester, developer) and coach since 1988. She has worked on agile projects since 2000 and published numerous experience-based papers for conferences, journals, and books. Her tutorials and training cover: agile requirements, process adaptation, automated examples, and project retrospectives. Find more at www.theandreagroup.ca and @jennitta_andrea.