The Role of User Stories & User Acceptance Tests in XP

In XP, requirements elicitation is done entirely though user stories. In contrast to other methods of requirements engineering, user stories allow the customer and programmer to overcome problems with understanding. When a customer “tells a story about how the system will be used” (Extreme Programming Installed, pg. 16), the requirements contain more background context allowing the programmers to understand the problem better. This allows programmers to ask more effective questions about the system to be developed to gain more insight into the problem. After the exchange of stories and questions the customer may feel that the initial story he or she started with is not satisfactory and will revise it. The continued revision of the story brings initially hidden requirements into light.

If a story is too large it is split up into smaller segments because programmers have more accurate estimates for smaller problems. As recommended by one of the founders of XP, Ron Jeffries states in his book: “stories should encompass a week or two of programmer time” (Extreme Programming Installed, pg. 18). For planning purposes this helps the development team stay as close to their estimated schedule as possible.

After the implementation is complete, acceptance tests are performed to verify the system indeed satisfies the system described in the user story. Like other development models, the purpose of acceptance tests remain the same – to allow the customer to verify the system performs as desired and to raise confidence in all the project stakeholders. However, acceptance tests also have another crucial role unique to XP. The XP model reduces the amount of documentation necessary by working in reverse; rather than writing the documentation first and deriving test cases from it, XP instead uses the test cases to derive what documentation is necessary. This model allows programmers to begin developing a system prior to heavy paperwork, and at the final stage only the relevant documentation will surface. Being agile in nature, XP demands that all tests – acceptance and unit tests – are automated.

Analysis

The method of using user stories is an interesting approach. On first impression it seems the developers may receive unnecessary information. More traditional software approaches required the customer to only specify what the system must do, and programmers were free to implement it with any desired strategy. Asking someone to “make this for me, and I don’t care how you do it” makes sense as we normally would not expect the client to have much understanding on the technical aspects of development, and those decisions would be better left for the developer to deal with. Although user stories take more commitment by both parties – customer and developer – due to the amount of communication that must be maintained, the process of writing and revising stories helps the customer understand the product truly desired and helps expose details that may have initially been omitted. By the same token, when a user story explains in detail the what, where, why, who, and how, it allows the developers to make more informed decisions during the implementation stage.

The role of user acceptance tests strongly enforces the paradigm of XP – a model that is adaptive rather than predictive. In waterfall models, development parties invest too many resources in analyzing the system and the runtime environments to discover all potential roadblocks. It is difficult or near impossible to tabulate every possible constraint, and there are likely to be constraints that are irrelevant to the current iteration of the system. By using acceptance tests at each stage to uncover the requirements, only currently relevant documentation is kept. This level of abstraction also allows the development team to focus on the current user story at hand, and this leads to a cohesive development system.

XP has a heavy focus on testing, demanding all tests be automated and running them on the system frequently. Due to XP’s nature in welcoming change, it is important to have lower level modules performing at the highest quality. This allows higher level models to be switched with another when requirements change, and developers will have confidence that the lower level system will continue to maintain its stability. That is not to say lower level modules need not be tested when the system as a whole undergoes change (take Ariadne 5 for example). But as software engineers, we generally do not “reinvent the wheel”, and we always use previously developed modules as a basis. For example, when we develop in a programming language, pinpointing the location of an error becomes easier when we can trust our compiler, because we know the error is within our own code rather than underlying systems. This optimism may not be applicable to large scale projects because of the amount of coupling necessary resulting in a more complex system, but XP is not an effective methodology when the project size is large due to face to face communication losing effectiveness.

<http://www.agilemodeling.com/essays/agileModelingXP.htm>

Extreme Programming Installed (By Ron Jeffries, Ann Anderson, Chet Hendrickson; October 16, 2000)