The Power of Stories

Rachel Davies
Developer
Connextra
Studio 312, 53-79 Highgate Road
London NW5 1TL, UK
+44 20 7692 9898
rachel@connextra.com

ABSTRACT

This paper explores the differences between Use Cases and Stories in Extreme Programming (XP) in incremental software development. The objective of this comparison is to dispel misconceptions about the definition of Use Cases and XP Stories. Better understanding of those differences will lead to their more effective use in the context of incremental development.

In classic software development processes, software requirements are specified "upfront" as documents feeding into a linear "waterfall" of development activity. Both Use Cases and XP Stories provide a means to separate out independent functional requirements, which is an essential step to developing software incrementally.

It appears that Use Cases and XP Stories have a common purpose, to describe functional requirements. For this reason, it may appear that the chief difference between these methods of requirements capture is the level of detail (or precision) in their respective textual descriptions. On this basis of comparison, an XP Story written on an index card might be considered to be a "light-weight" or "cut-down" description of a Use Case scenario, with fewer words and less formal constraints.

However, the purpose of the XP Story is not to document requirements but to enable incremental software development to proceed in an environment where requirements change is expected. This paper asserts that the key differentiating factor between Use Cases and XP Stories is the way that their scope is determined and not the level of detail in their description.

In XP, Stories are by definition time-bounded (in estimated development time) to enable their complete implementation in a single iteration. In contrast, the scope of a Use Case depends on applying an abstract definition, concerning system interaction with external actors, to the development domain. The definition that is used to identify Use Cases is independent of any time considerations, such as estimated development time or development iteration planning.

This paper suggests that it is the "time-boxed" aspect of XP Stories that makes the activity of writing them, in the context of the Planning Game, such a powerful driver in the planning of software development iterations.

Keywords

Use case, story, requirements, RUP, XP.

1 REQUIREMENTS

Customer requirements specify system behaviour (or capabilities) prior to the procurement of a system that provides this required behaviour. A system implementation can be shown to meet requirements by functional tests. Both Use Cases and XP Stories provide a means to capture functional requirements for subsequent software development.

The Rational Unified Process (RUP) supports iterative software development for the same reasons as XP, to reduce risk and gain benefit from early user feedback. However, both these development methods process requirements very differently.

2 DEFINITIONS

Use Case Definition

Use Cases were introduced by Ivar Jacobson [3] in the Objectory process. A definition of a Use Case is part of the standard definition of Unified Modeling Language (UML): "A use case describes the interactions between the users and the entity as well as the responses performed by the entity, as these responses are perceived from the outside of the entity. A use case also includes possible variants of this sequence (e.g., alternative sequences, exceptional behavior, error handling, etc.)." where the term "entity" refers to the system [6].

A Use Case is an abstraction, a generalization of specific instances of system interactions. Additionally in UML, relationships between Use Cases may be expressed using three types of association with other Use Cases: *generalization*, *includes* and *extends*.

The Objectory, OMT and Booch methods have since fused in the evolution of the Rational Unified Process [4,5]. However, Use Cases remain at the heart of RUP. Central to RUP is the production of models of the system prior to code (although not to the exclusion of some software prototyping). Use Cases break down functional requirements into a suitable form for analysis and design models to be developed. By focusing on individual event sequences, a designer may populate an object model. The objective of producing models is to ensure the consistency and completeness of a set of requirements and to provide documentation that can be used to understand what the system is meant to do.

Scope
The scope of an individual Use Case is determined by whether the description meets the definition of a Use Case and is complete with respect to variations of the sequence of interactions with the system.

Form
The formal definition of Use Cases as an abstraction increases the technical jargon and mechanisms of representation, such as Use Case templates and several types of UML diagrams that the customer needs to understand to write or agree to Use Cases.

Ownership
In RUP, Stakeholders for the system under development are identified; examples are an end user, a purchaser, a system administrator, etc. Requirements are elicited from these stakeholders but the processing of those requirements is done by the project team and not by the customer. Typically, a set of requirements is prioritized by an Architect, and iteration plans are maintained by a Project Manager.

Limitations
RUP makes it clear that Use Cases capture only functional requirements; "non-functional" requirements relating to factors such as performance, reliability, etc are captured in other RUP artifacts. "Supplementary Specifications are an important complement to the Use-Case Model, because together they capture all software requirements (functional and non-functional) that need to be described to serve as a complete Software Requirements Specification."[5].

Story Definition
In his first book on Extreme Programming, Kent Beck defines a Story as: "One thing the customer wants the system to do. Stories should be estimable at between one to five ideal programming weeks. Stories should be testable."[1]. Further to this definition "Stories need to be of a size that you can build a few of them in each iteration"[2].

Scope
A customer story is limited in XP by estimated development time. Following XP principles, a developer can only provide reliable estimates based on the measurement of past velocity in the context of an iteration. Stories which are too big to estimate must be split into smaller Stories of no more functionality than can be implemented in a single iteration. XP Stories are thus limited in scope by time. Stories can be completed in a single iteration, which ensures that feedback from their early implementation is gained before further requirements are costed.

Form
The definition of XP Stories does not specify a particular form of expression beyond that it should be possible to write them in natural language on an index card. This liberates the customer from having to understand formal definitions or special notation. The detail recorded on a story card needs only to be the estimate and a title to differentiate the story from others in the same iteration, with some words or sketches to recall the discussion between the customer and the developer.

Ownership
In XP, the customer owns the Stories. It is the customer rather than a project manager who controls the content of development iterations. There is only one means of allocating requirements to an iteration, via the Planning Game, which enables the on-site customer to prioritize the Stories in a release plan.

Limitations
XP Stories are not limited by the same formal constraints as Use Cases, and so Stories can be written to detail some types of functional requirements that cannot be categorized as Use Cases. As a general rule, if a functional test can be expressed to verify that a system conforms to a requirement, then it should be possible to express the requirement using an XP Story.

3 COMPARISON OF RELATED ARTIFACTS

RUP uses the term "artifact" to refer to documentation, models or software used or produced by a software development process. The essence of Use Cases and XP Stories may be understood better by a comparison of their artifacts.

A requirement may be represented in several different artifacts. A Use Case description can be identified as a process artifact in RUP workflows. Similarly a Story card can be identified as a process artifact of XP. However, a requirement may also be represented in a process by other equally important artifacts such as Functional Tests.

It should also be recognized that direct verbal communication is an alternative to the capture of information as artifacts. In XP, a greater emphasis is placed on verbal rather than written communication. "A user story is nothing more than agreement that the customer and developers will talk together about a feature"[2]. For example, an XP Story card will probably not include notes on every aspect of the story discussed between the customer and developer when it was estimated. It does not need to do so, if the developer who estimated the card can implement the engineering tasks associated with this story, with the customer on-site and available for any clarification required.

In RUP, Use Cases models and descriptions are usually considered to be artifacts of documentation, to be kept after software release to aid software maintenance. In XP, Story cards are not preserved as software documentation.

Story cards should be disposed of following a software release. Although Stories are written down on cards, these cards act as tokens within a release plan rather than descriptions of Stories; they represent customer requirements rather than document them. XP Stories are like bars on a Gantt chart, when the plan is complete the Stories serve no further purpose. In XP, it is the job of Acceptance Tests, not Story cards, to document and preserve the accumulated set of requirements on the system [2].

RUP provides an extensible process framework [5] so the kinds of artifacts produced under workflows within a RUP software development are more varied than using XP (which is more strictly defined as a software development process).

In a typical RUP development, the artifacts generated for a set of Use Cases could be: a Use Case model (UML diagrams and supporting textual descriptions), a design model, software development plan, software components, a test plan and test cases.

The artifacts produced in XP for each Story are as follows: a Story card in a release plan, engineering tasks in an iteration plan, source code with associated unit tests, acceptance tests and a software release.

4 ISSUES IN ITERATIVE DEVELOPMENT

Consistency

Use cases provide a mechanism to attack problems of inconsistency. The objective of Use Case analysis is primarily to expand initial requirements to a complete set (without holes) including many alternate scenarios. This ensures that development effort is not spent on unnecessary implementation of ill-thought-out scenarios.

When using XP, gaps in requirements emerge by the delivery of software to a customer that conforms to their requests, made in the form of Stories. The delivered software only meets those requirements discussed with the customer. It is the responsibility of the customer to take the time to ensure that the Stories they select for the Planning Game are consistent with one another. It is easier for the customer to do this where requirements are written in their own terms. Any unplanned behaviour can be addressed by writing new Stories and which may be planned into future iterations.

Planning

Iteration plans are developed on the basis of development estimates. Estimates are more reliable where metrics are available to support them; they are *most* reliable when comparing like with like. This is one reason why there is a benefit to maintaining an iteration cycle with the same iteration period. To enable the planning of iterations of the same duration "You must define only enough work to fill the iteration. You are scheduling by time and not volume."[5]. As an analysis tool, Use Cases expand requirements; this tendency to expansion may stand in opposition to the restriction of scope needed for a unit of requirements to be used in planning incremental development.

When planning iterations, units of work need to be identified and development estimates applied to them. A Use Case provides a unit of requirements. But is the Use Case a suitable unit of requirements to base development plans on? This depends on two factors: the complexity of Use Cases identifiable in the development domain, and the iteration period to be adopted.

Although it is possible for a software development to have requirements that can be broken down into Use Cases of a size that allows their complete implementation in single iterations. Where Use Cases are more complex, this may not be possible and only some scenarios of a Use Case can be implemented in an iteration. Also, particularly in early iterations of a development, it is common practice in incremental development to exclude non-essential steps in scenarios, such as error reporting, performance measurement, time-outs, security checks, recovery or rollback action. So an iteration plan may be adopted where the work units exclude some scenarios or steps in scenarios of a Use Case.

In planning iterations, it is important to be able to distinguish what requirements have been implemented and what requirements remain. Ideally, an iteration plan should reference units of requirements that can be completely implemented in that iteration. If Use Cases are too large to be implemented in a single iteration, partial implementation in successive iterations may be necessary. Where this is done, additional work will be needed to keep track of those parts of a Use Case that have been implemented. If it does become necessary to track requirements at the level of steps in a Use Case scenario (and maintain correlation with requirements documented in supplementary specifications) then this can become difficult to manage in subsequent iterations, especially where requirements change between iterations.

An alternative approach is to increase the duration of iterations, to allow whole Use Cases to be completed. However, this increases the time that development proceeds without the feedback derived from the completion of the iteration. This practice of adapting the iteration length may also weaken the accuracy of estimates, where they are based on measurements taken within iterations of irregular length, as it becomes harder to compare like with like.

In contrast, XP Stories by definition must allow completion in a single iteration; this frees project management from the extra work of tracking requirements. Iteration completion is determined by functional test passes. At the start of each iteration, the customer is expected to introduce new requirements; this is the norm not the exception.

Change Management

Where face-to-face verbal communication is easy, fewer process artifacts may be needed. However, large teams or distributed teams make such direct communication difficult, so more artifacts need to be produced. In these circumstances, such artifacts are likely to be subject to change management procedures, so that they are stored safely and kept up to date. For this reason, duplication of information across artifacts should be avoided to prevent artifacts going out of phase and to control the overhead of the extra work needed to keep them aligned.

XP avoids the problem of keeping requirements aligned across artifacts by increasing direct communication and avoiding duplication.

RUP provides change management mechanisms and recommends the use of tools to support the automation of such process workflows.

5 CONCLUSIONS

This paper does not suggest that Use Cases are better than XP Stories, or vice versa. It attempts to understand the forces which should influence an informed decision to select them for the capture of requirements in incremental software development.

Due to the need for direct communication, XP is only viable for small co-located teams with access to an on-site customer. Large distributed teams may need to rely on more documentation and adopt RUP or other less agile processes.

In XP, Stories provide a time-boxed unit of requirements with the advantage that, on the completion of an iteration, requirements can be completely implemented. The practice of rejecting Stories with large estimates and splitting them into other Stories (rather than another unit of requirements) allows a fixed iteration length to be adopted, supporting future development estimates.

A development project may also have small enough Use Cases for them to be completed in a single iteration. But in some developments, Use Cases can only be identified which are larger than can be implemented within an iteration period, that can only be split into “subunits” of Use Cases, such as scenarios or steps within those scenarios, but not into other smaller Use Cases. In such developments, it may be necessary either to track the implementation of partial Use Cases or adopt irregular iteration periods. The former adds to management overhead costs and the latter reduces the benefit of early feedback and accurate measurements on which to base future estimates.

6 INFORMATION AND QUESTIONS

For more information, contact *rachel@connextra.com*

ACKNOWLEDGEMENTS

This paper draws on experience gained by the author in employment with Marconi and CMG using Use Cases for requirements capture in incremental development cycles and in her current role at Connextra using XP Stories.

REFERENCES

1. Beck, K. *Extreme Programming Explained,* Addison Wesley, 2000.
2. Beck, K., Fowler, M., *Planning Extreme Programming*, Addison Wesley, 2000.
3. Jacobson, I. et al, *Object-oriented Software Engineering,* Addison Wesley, 1992.
4. Jacobson, I., Booch, G. and Rumbaugh, J. *The Unified Software Development Process*, Addison Wesley, 1998.
5. Krutchen, P. *The Rational Unified Process An Introduction*, Addison Wesley, 2000.
6. Object Management Group, *Unified Modeling Language Specification*, v1.3, 2000.