The Quality of Requirements in Extreme Programming

Richard Duncan
Mississippi State University
richard.duncan@ieee.org

Abstract

Extreme Programming (XP) is a software process methodology that nominates writing code as the key activity throughout the development process. While at first glance this sounds chaotic, a disciplined group utilizing XP performs sufficient requirements engineering. This paper describes and evaluates the quality of requirements generated by an ideal group using XP and discusses how the XP process can assist or hinder proper requirements engineering.

Introduction

Extreme Programming (XP) is a hot, new software process methodology for medium to small sized organizations (generally around 10 people). It is designed with requirements drift as a fundamental occurrence to be embraced, rather than dealing with it as a necessary evil. XP nominates coding as the key activity throughout the development process, yet the methodology is based on economics (1).

Barry Boehm presented that the cost of change grows exponentially as the project progresses through its lifecycle (2), and Faulk reiterates this by stating that the relative repair cost is 200 times greater in the maintenance phase than if it is caught in the requirements phase (6). XP challenges that this is no longer the case. While it is more expensive to modify code than to modify a prose description, with modern languages and

development techniques it is not an exponential increase. Instead, Beck asserts that the cost for change levels out. Rather than spend extra effort in the requirements analysis phase to nail down all requirements (some of which will become obsolete through requirements drift anyway), accept that changes due to incomplete requirements will be dealt with later. XP assumes that lost resources in rework will be less than the lost resources in analyzing or developing to incomplete requirements (1).

The primary vehicle for requirements elicitation in XP is the addition of a member of the customer's organization to the team. This customer representative works full time with the team, writing stories (similar to UML Use Cases), developing system acceptance tests, and prioritizing requirements (8). The specification is not a single monolithic document; instead, it is a collection of user stories, the acceptance tests written by the customer, and the unit tests written for each module. Since the customer is present throughout the development, that customer can be considered part of the specification since he or she is available to answer questions and clear up ambiguity.

The XP lifecycle is evolutionary in nature, but the increments are made as small as possible. This allows the customer (and management) to see concrete progress throughout the development cycle and is capable of responding to requirements changes faster. There is less work involved in each release, therefore the time-consuming stages of stabilization before releases take less time. With a longer iteration time it may take a year to incorporate a new idea, with XP this can happen in less than a week (1).

A fundamental of XP is testing. The customer specifies system tests, the developers write unit tests. This test code serves as part of the requirements definition – a coded test

case is an unambiguous medium in which to record a requirement. XP calls for the test cases to be written first and then the simplest amount of code to be written to specify the test case. This means that the test cases will exercise all relevant functionality of the system, and irrelevant functionality should not make it into the system (1).

This paper describes and evaluates the requirements engineering processes associated with the Extreme Programming paradigm.

The XP Requirements Engineering Process

Harwell et al. break requirements into two types – product parameters and program parameters. A product parameter applies to the product under development, while a program parameter deals with the managerial efforts that enable development to take place (7). The customer who becomes a member of the XP team defines both product and program parameters. The product parameters are defined through stories and acceptance tests, while the program parameters are dealt with in release and iteration planning.

The product parameters are chiefly communicated through *stories*. These stories are similar to Use Cases defined in UML, but are much simpler in scope (8). Developing a comprehensive written specification is a very costly process, so XP uses a less formal approach. The requirements need not be written to answer every possible question, since the customer will always be there to answer questions as they come up. This technique would quickly spiral out of control for a large development effort, but for small-to-medium sized teams it can offer a substantial cost savings. It should be noted, however, that an inexperienced customer representative would jeopardize this property.

The programmers then take each story and estimate how long they think it will take to implement it. Scope is controlled at this point – if a programmer thinks that the story, in isolation, will take more than two weeks to implement the customer is asked to split the story. If the programmers do not understand the story they can always interact directly with the customer. Once the stories are estimated, the customer selects which stories will be implemented for the upcoming release, thereby driving development from business interests. At each release, the customer can evaluate if the next release will bring business value to the organization (1).

Each story to be implemented is broken up into tasks. A pair of programmers will work to solve one task at a time. The first step in solving a task (after understanding, of course) is to write a test case for it. The test cases will define exactly what needs to be coded for this task. Once the test cases pass, the coding is complete (1). Therefore, the unit tests may be considered a form of requirements as well. Every test (across the entire system) must pass before new code may be integrated, so these unit-test requirements are persistent. This is not to say that simple unit testing counts as an executable specification – but XP's test-driven software development does record the specific requirements of each task into test cases.

The final specification medium for product requirements is the customer acceptance tests. The customer selects scenarios to test when a user story has been correctly implemented. These are black-box system tests and it is the customer's responsibility to ensure that the scenarios are complete and that they sufficiently exercise the system (5). These

acceptance tests serve as an unambiguous determiner as to when the code meets the customer's expectations.

How XP rates

The XP requirements engineering process can be analyzed by considering the 24 quality attributes for software requirements specification proposed by (4). Davis et al. propose that a quality SRS is one that exhibits the 24 attributes listed in Table 1. Rather than applying these metrics to a given document, they are used here to measure the requirements that theoretically come out of the XP process. Of course, a quality SRS is mostly dependent on the discipline used by the people associated with the project, but specific features of XP can influence the quality of a SRS.

1. Unambiguous	+	13. Electronically Scored	+/-
2. Complete	-	14. Executable/Interpretable	+/-
3. Correct	+	15. Annotated by Relative Importance	+
4. Understandable	+	16. Annotated by Relative Stability	?
5. Verifiable	+	17. Annotated by Version	+
6. Internally Consistent	+/-	18. Not Redundant	-
7. Externally Consistent	+/-	19. At the Right Level of Detail	?
8. Achievable	+	20. Precise	?
9. Concise	+	21. Reusable	?
10. Design Independent	+/-	22. Traced	?
11. Traceable	?	23. Organized	?
12. Modifiable	+	24. Cross-Referenced	?

Table 1: The 24 quality attributes proposed in (4). A '+' indicates XP may assist in this area, a '-' that it degrades this area, a '+/-' indicates that it both assists and degrades, and a '?' indicates XP has little bearing on the area.

A specification created with XP would appear to score very well across most of these attributes, but fare poorly on others. Those qualities with a '+' symbol indicate that the subsequent paragraphs argue the XP process can lead to an improvement in the area, a '-' that XP detracts from the quality. The '+/-' annotation indicates that XP partially helps

and partially harms a specification in achieving the quality. Many of the qualities are not addressed by XP and are hence annotated with a '?,' for these qualities a groups organization, discipline, and specific project needs will decide. It should be noted that to religiously follow XP requires a great deal of discipline, so this discipline should be expected to carry over into the other qualities.

Unambiguous, Correct, and Understandable. Since the customer is present, ambiguity and problems understanding the requirements are generally minimal and easily solvable (1). Requirements are correct if and only if each represents an actual requirement of the system to be built. Since the customer writes the stories from business interests, the requirements should all be correct. With so much responsibility and freedom, clearly the selection of an appropriate customer representative is crucial to the success of the project. Even if the customer does not know exactly what he or she desires at the start of the project the evolutionary nature of XP development leads to a system more in line with the customer's needs.

Modifiable. The XP lifecycle allows changes to the requirements specification at nearly any point in system development. The specification exists as a collection of user stories, so the customer can switch out one future story for another with little impact on existing work. Since the planning, tests, and integration are all performed incrementally, XP should receive highest marks in modifiability. Of course work may be lost in this changeover, but with XP the programmers should be able to estimate how much a change will cost.

Unambiguous, Verifiable. Since the customer writes acceptance tests (with the assistance of programmers), it could be argued that the functional specification is recorded in an unambiguous format. Furthermore, the first activity performed by a programming pair to solve a task is to write test cases for it, and these test cases become a permanent part of the specification/test suite. Customers (with the help of the XP coach) will also make sure that the specification is verifiable, since they knows that they will have to write test cases for it.

Annotated by Relative Importance. The customer defines which user stories they wish implemented in each release. Hence, each requirement is annotated by relative importance at this time – the customer should for ask the highest-priority stories to be implemented first and the programmers are never left guessing priorities.

Achievable. Since each release provides some business value, a portion of the system found to be unachievable should not leave the customer with a very expensive yet unusable piece of technology. If the high-risk piece is important, it will be implemented first, in which case the unachievable component should be found quickly and the project aborted relatively inexpensively. If it is less important, then the system may be delivered in useful form without it.

Design Independent. Design independence is a classic goal for requirements, but today's object-oriented development methods recognize that design independent requirements are often impractical. Portions of the requirements (such as the user stories) can be very design independent, but the unit tests that are archived as part of the requirements and used to crosscheck new modules may depend heavily on the actual system.

Electronically Stored. XP calls for the stories to be written on index cards, so this portion of the requirements is not electronically stored. While the stories could be placed in a word processor, Jeffries et al. assert that handwritten index cards produce less feelings of permanence and allow the customer to more freely change the system (8). The customer is also available as a requirements resource, obviously not electronically stored. However, the requirements are written on individual cards so modifications can often be localized to a single card if rewriting is necessary. Furthermore, the customer codifies the system requirements with acceptance tests, so it could be argued that the most important part of the specification is stored.

Complete, Concise. XP stresses programming as the most important development activity, hence little effort is spent on creating documents, therefore the specification is very concise. The cost may be a lack of completeness, however. Since little up front analysis takes place, there may very well be holes in the system. Yet the customer drives what functionality is implemented and in what order, so true functionality should not be left out. Furthermore, since the XP process accommodates change, it should be possible to compensate for these holes later in the development lifecycle.

Security Assurances

Since the XP development methodology does not progress from a verified requirements document, how might a system developed with XP rate on a security evaluation? The Common Criteria has 7 evaluation assurance levels (EAL1-EAL7). For EAL5 and above the Common Evaluation Methodology calls for the system to be semi formally designed and tested (3). This leaves two questions to be addressed. First, can a project use formal

methods with XP? Second, without formal methods, how trusted can a system developed under XP be?

The XP process screams informality in many respects. The name alone conjures images of snowboarders with laptops, and even the books about XP are written in a conversational tone. Nevertheless, what would happen if the customer writes stories and they are annotated with a formal specification? Clearly, this would entail a large cost in training personnel, writing the specifications, and verifying the specifications. This also reduces the agility of the XP product – since more money is spent on specification the cost of change will increase. But if each story were rewritten in a formal notation it would be possible to formally verify the specification and design.

Formal methods aside, the way an XP project progresses does offer many assurances of trust. First, all code is written directly from the user stories (the specification). All functionality is tested in the unit tests and all integrated code is required to pass all tests all the time. While testing does not guaranty the absence of errors, many security holes come from poorly tested software. Hence, the test-oriented nature of XP may be a great step forward. A strong security feature of XP is pair programming. The observer in a pair constantly evaluates the code being written by his or her partner. This programmer can help reduce the probability of coding errors that might later be exploited (e.g., buffer overruns). XP also adds counterbalances to reduce the impact of a single malicious coder (either in a truly malevolent sense or inadvertently opening holes as Easter Eggs[1] side effects) through the pairing process. Rather than just inserting code into the system, one

---

[1] An unsolicited, undocumented piece of code a programmers inserts into software, generally for his or her own amusement.

programmer would have to convince the other of a rationale for why the code was being inserted. Due to collective code ownership, it is entirely possible that the next pair in the course of refactoring would catch malicious code. Pair programming and collective code ownership add further assurance that the code is written exactly to the specification.

Conclusions

Extreme Programming performs requirements engineering throughout the lifecycle in small informal stages. The customer joins the development team full time to write user stories, develop system acceptance tests, set priorities, and answer questions about the requirements. The stories are simpler in scope to use cases because the customer need not answer every conceivable question. The informal stories are then translated into unit and system acceptance tests, which have some properties of an executable specification.

Of the 24 quality attributes of a software specification, the XP process leads to higher points in nine attributes and lowers the score in two. The most noteworthy gains are in ambiguity and understandability, since the customer is always present to answer questions and clear up problems. Furthermore, since the customer is also responsible for developing test scenarios he or she will create more verifiable requirements. The discipline enforced by the XP process should also carry over into the other areas of requirements engineering.

References

1. Beck, Kent. *Extreme Programming Explained: Embrace Change*. Boston, Addison Wesley, 2000.

2. Boehm, Barry. *Software Engineering Economics*. Prentice Hall, 1981.

3. "Common Criteria," International Standard (IS) 15408, Available at
   http://csrc.nist.gov/cc/ccv20/ccv2list.htm (Accessed 4 January 2000), September
   2000.

4. Davis, Alan, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi,
   Anhtuan Dinh, Gary Kincaid, Glen Ledeboer, Patricia Reynolds, Pradhip Sitaram,
   Anh Ta, & Mary Theofanos. "Identifying and Measuring Quality in Software
   Requirements Specification," *Proceedings of the First International Software
   Metrics Symposium*, 1993, pp. 141-152.

5. "Extreme Programming: A Gentle Introduction." Available at
   http://www.ExtremeProgramming.org (Accessed 21 November 2000).

6. Faulk, Stuart. "Software Requirements: A Tutorial". Software Engineering, M.
   Dorfman and R. H. Thayer, Eds. 1996, pp. 82-103.

7. Harwell, Richard, Erik Aslaksen, Ivy Hooks, Roy Mengot, & Ken Ptack, "What is
   a Requirement." *Proceedings of the Third Annual International Symposium Nat'l
   Council Systems Eng*., 1993, pp. 17-24.

8. Jeffries, Ron, Ann Anderson, & Chet Hendrickson. *Extreme Programming
   Installed*. Boston, Addison Wesley, 2001.

Author Biography

Richard Duncan is pursuing a Master's degree in Computer Science at Mississippi State University with emphasis in software engineering. He has held summer internships at Microsoft, AT&T Labs Research and NIST. His current research interests involve applying software engineering process to the development of a public domain speech recognition system at the Institute for Signal and Information Processing (ISIP) at MSU.

Contact Information

Author name: Richard Jennings Duncan
Author's business organization: Mississippi State University
Business address: PO Box 9571, Mississippi State, Mississippi, 39762
Voice number: 662-325-8335
Fax number: 662-325-2292
Email address: Richard.Duncan@ieee.org
Internet address: n/a

Copyrights

Releases

No releases are necessary from my organization for publication

Response to editor comments

1. What is the range of a small-to-medium program?

Added numbers.

2. Where can reader reference Common Criteria and Orange Book? Can we get a brief description of these? What does a rating of B2 mean?

There is a reference to the Common Criteria. I removed the reference to the Orange book, it is redundant with the new standard.

3. What is a lightweight software process methodology?

Explaining this term would break the flow of the introduction, so I removed it. It isn't really needed.

4. What is an Easter Egg in this reference?

Added a short definition

5. Do you have any data on real results or quality?

Personally, no. The proof of concept project for XP was the C3 payroll system for Chrysler (Extreme Programming Installed describes this somewhat). How much of this should I put in?

6. I question Table 1's claim that this process help with correctness (item 3) of a project.

I added another sentence to clarify.

7. Subsequent paragraphs do not easily tie back to Table 1 as promised.
Maybe putting appropriate subheadings on the paragraphs would help with the
clarity.

8. "Many of the qualities have little to do with XP..." Do they have
little to do with XP or does XP have little to do with them?

Reworded the sentence.

9. Third paragraph under "How XP Rates." These are standard problems
that I don't believe XP will fix. When requirements change the project is
impacted because of changes to tests, rework, integration, and planning.

Reworded to specifically address this.

10. Sixth paragraph under "How XP Rates." This paragraph is not clear
and could be better explained.

I broke it up and simplified the English somewhat.

11. Eighth paragraph under "How XP Rates." Why can't this be made into
a word processor file for electronic access?

Addressed this counterpoint.

12. Last paragraph before "Conclusion" Many projects pass their tests,
but then release bugs to the users. Passing tests means little.

I didn't say or imply that passing tests means bugs will not be released to the user, all I said was that a poorly tested system is more likely to have security holes than a rigorously tested one.

13. This paper does not clearly define Requirements Engineering. It
should if it is to keep its current title. A title more along the lines of
"XPs Approach to Customer Requirements" is in line with this paper.

I changed the title to something more appropriate.

14. This paper (and XP literature in general) only alludes to the
importance of selecting the right customer representative (s).

I added a sentence to this effect.

15. Does the author have examples of real projects that have used XP and
their results?

Personally, no. The proof of concept project for XP was the C3 payroll system for Chrysler (<u>Extreme Programming Installed</u> describes this somewhat). How much of this should I put in?