Challenges for Analysts on a Large XP Project

Dr. Gregory Schalliol
ThoughtWorks, Inc.
651 W. Washington Blvd., 6th Floor
Chicago, IL 60661 USA
+1 312 373 8661
glschall@thoughtworks.com

ABSTRACT
The author recounts the fundamental issues encountered by a team of 8 analysts on a 50-person, multi-year development project that converted to an XP process. Those include the importance of comprehending the whole application in addition to the parts, the art of dividing the whole into meaningful story cards, the sometimes complicated role of the customer, and the place of more traditional analysis. In describing the team's responses to these issues, the author suggests what challenges one might expect for analysis and analysts in large XP projects.

Keywords
Analysis, Analyst Role in XP, Project Size

1 INTRODUCTION

In Kent Beck's groundbreaking book *Extreme Programming Explained*, the word *analysis* does not appear in the index, and Beck explicitly warns against trying to use the methodology for large development projects [1]. I, however, am an analyst using XP on a large development project, and the point of this paper is to show that I am not completely foolish for having played such a role.

The case I wish to discuss is a large J2EE development project that switched to an XP approach about halfway through its three-year life. In the 50-person team on this project, there are about 30 developers, 8 quality assurance testers, and 8 analysts. The application being built is a comprehensive, "back-end" leasing system, namely, a product that would manage everything from the moment a lease is booked through the eventual disposal of the leased assets. This includes all aspects of accounts receivable, asset management, and lease termination, not to mention the million and one possible ways the laws allow one to fashion a lease for renting something to someone. At present, the application consists of over 500,000 lines of executable code. Our initial customer/user is the leasing arm of a traditional, Fortune 500 company, but we are partnering with that first user to offer a more generic version of the application for the leasing industry as a whole.

Traditional wisdom would say such a project is not a likely candidate for an XP approach, and the bulk of my paper will recount the difficulties we encountered with XP on this project and the ways we dealt with them. Despite these difficulties, however, and despite our deviations from some core XP practices, I wish to show how we experienced sufficient success to recommend similar methodologies for large projects in the future.

2 THE MISSING PICTURE OF THE WHOLE

The first challenge I will address is the difficulty we encountered because we had no holistic picture of the application available to everyone during the development process. Although much planning and analysis had been performed before our project switched to an XP approach, once it was adopted, our primary roadmap was the set of story cards we developed and arranged in a large spreadsheet. At best, a more holistic picture of the application existed in the minds of those analysts on our team with extensive experience in the leasing business. But for those without such experience, which included most members of the team, the application appeared as a collection of independent parts without a clear image of their connection in a whole. We refrained from developing any more up-front documentation or graphic of the whole application in order to reap the purported benefit of XP's "agility." The story cards, we thought, would be enough of a guide.

But the absence of a readily available, holistic picture in this case contributed to a number of problems. Since leasing has exceptionally complex business logic, which is often counter-intuitive, team members without direct business experience in leasing tended not to understand how their particular stories fit in with or depended on other stories in the whole scheme of things. Hence, when they were charged with implementing new stories, they often left out tests that would verify proper functioning of the newer stories in conjunction with the older ones previously developed. As the iterations accumulated and the complexity of the system grew, even our analysts with extensive leasing experience were frustrated by not knowing sufficiently how the parts were connected so as to write sufficient tests to verify completed stories. So as you might expect, new cards were often "finished" in the eyes of their owners when, in fact, they were not.

When this would become evident, we would, of course, write new story cards to cover what we had missed. But there was a general feeling of frustration among card owners when they discovered they had missed some dependency, and at some times, minimal inclination to acknowledge that their card actually included the dependencies in question. A common response from many team members to the difficult task of developing and testing a new bit of functionality in all of its entanglements with existing functionality was to quip, "Why don't we just use the XP approach?" The implication, of course, is that XP looks at the whole as a collection of atomistic stories, namely, stories that can be treated as independent of one another. Whereas that may tend to be the case in some more simple applications, that was hardly the case in this one. As the application grew in size and complexity with each new iteration, the amount of "analysis" required to compose a sufficient list of tests for each new story increased tremendously. Developers and analysts both tended to know the local functionality of a finite number of stories or sections of code, but hardly anyone had a comfortable grasp of how it was all connected, despite the extensive lines of communication among team members. For new members joining the team after the beginning of the project, the effort to understand the significance of the part without knowing the whole was especially daunting.

What was lacking, we think, was some easily accessible "picture" of the whole so that both developers and analysts adopting a new story could readily "see" what sort of connections with other parts they would have to test in order to complete the card. This would not have had to be a static picture that was produced before the first line of code was written. But at the very least, it would have had to be something that was available in an updated, cumulative form at the beginning of each iteration so that story owners could take their bearings and reliably estimate the scope of the stories and the requisite tests. No one could conceive of a useable "metaphor" to help "everyone on the project understand the basic elements and their relationships" [2], because leasing is too complex a business to be productively communicated through a simpler image. A more traditional picture or graphic was needed and would have helped us tremendously, but producing and maintaining it would have forced us to divert resources to the sort of design overhead that XP de-emphasizes for the sake of agility.

To be sure, some of the difficulties here are the fault of the nature and size of our project. Less complex business domains are more easily described with metaphoric images, and less complex applications will have many fewer dependencies among stories. Nonetheless, we found ourselves too easily lulled into the belief that most stories are independent of one another [3] and that we could find a useful metaphor for a very complex system. If XP is to be used for developing complex applications, we suggest some mechanism for constantly reminding everyone that functionality divided into distinct stories does not imply the independence of the stories. This is where a single, integrated "picture" needs to supplement a spreadsheet of separable story cards.

3 LEARNING HOW TO DIVIDE AT THE JOINTS

Related to this problem was the difficulty we encountered dividing the whole into story cards. To borrow an image from Plato, as well as from your own kitchen, one would think the division of the whole application into its story parts would be like cutting a whole chicken into its familiar pieces. Even though the joints may not be immediately visible, there are places to do the dividing that are more appropriate than others. But if you are cutting up the chicken for the first time, as I recall I once did in my parents' kitchen, you may encounter great difficulty because you do not know where the joints are.

It took us a good deal of practice to find the joints, particularly because we had many different cues for locating them. One cue we used was to divide into cards in ways that worked well in earlier iterations. Although this tended to work well early on, as the application grew in size and complexity, it became unreliable. The sort of functionality that was an independent chunk in iteration four had become, by iteration fourteen, intertwined with several other chunks that had been completed in the meantime. Hence, on more than one occasion, we discovered we had missed testing one or more interactions because the list of story cards by itself did not include any built-in guide to interactions among past and present cards. Here, again, a holistic picture of the whole application that would be updated regularly would have helped tremendously. Without this, we found ourselves devoting much time to reviewing and rewriting story cards on a continuous basis in order to try to keep track of new functional interactions in the card texts. We eventually devoted one full-time analyst to developing and managing the story card list with our customer.

Another guide we used to distinguish story cards was to divide by bits of functionality that would produce some new, visible business value in the build we would deliver to the customer at the end of each iteration. The goal of this was to insure that the customer could *see* ongoing progress. But dividing at these "joints" often turned out badly. For example, we played one story card early on entitled "Terminate Lease Billing Schedule." At first glance, this distinct mechanism seemed like a perfect candidate for a story card, because it encompassed a clear function that the customer could understand and actually use when finished. But as we began to implement it in a one-month iteration, we discovered that our desire to deliver the whole function at once led us to badly underestimate the time needed for the card. Luckily, the analyst for the card had devised her functional tests in such a way that we were able to divide

up the card into smaller cards along the lines of her functional tests. Thus, although the customer did not have the whole of the termination functionality available for use after one iteration, some testable part of it was finished. Over the course of the next two iterations, the other, more granular cards were finished. In the process, however, we learned our lesson not to divide automatically at joints of fully useable functionality. But this meant that we had to prepare our customer to be patient with partially finished business functions after certain iterations were completed.

From this experience, we perceived a precarious tension between two goals of XP. On the one hand, iterative development promotes the impression that the customer receives some level of a useable application at frequent intervals and can, as a result, decide to terminate a project at many stages and still walk away with an application having business value. On the other hand, the division of development into iterative chunks often makes it impossible to deliver functionality that the customer can actually use for his business at the end of any particular iteration. In the case of our "Terminate Lease Billing Schedule" example, the chunk we delivered at the end of the first iteration could be used and tested, but from a business perspective, it was valueless without the other chunks that were completed in subsequent iterations.

In sum, dividing story cards well means not following any one particular guide too rigidly. To return to the example of the joints of the chicken, if one insists on having a chicken quartered, some of the cuts may be easy because they happen to fall at natural joints, but that last cut through the breastplate will create much additional toil. We found that trying to adhere too rigidly to card division by deliverable functionality or by past experience often created more toil rather than less.

4 LOOKING BEYOND TODAY'S PART

Despite our awareness of the XP admonition not to implement beyond what is stated in the current card or iteration, we found ourselves constantly wondering, and often cursing, why we should not do just that. This frustration increased when we finally realized how often we would have to phase in the development of business functionality gradually over multiple iterations. If the first part of this business functionality, implemented in iteration *n*, is useless without the additional parts implemented in iterations *n*+1, *n*+2, etc., then why not bend the XP rule against pro-active design and implement certain things in iteration *n* so that you do not need to refactor in iteration *n*+2? There was (and still is) general disagreement among team members as to whether to bend the rule here or not, but we generally swallowed our frustration, knowing that refactoring would have to be done, and followed the XP line.

5 IDENTIFYING OUR CUSTOMER

Our customer/partner for this project devoted a team of its employees full time to this project, but they were not on-site with our development team. This fact contributed to expected problems in the efficiency of communication between customer and developer, but these were not the most difficult challenges that confronted us in this area. Due to the breadth and complexity of the application, it was impossible for us to have the XP ideal of a customer who was also an end user. In a typical leasing business, the person responsible for managing accounts receivable for its customers is not the person who handles end-of-lease transactions, nor the person who books the original lease. The application we were building, however, required a "customer" who was simultaneously familiar with all of these dimensions of the business, as well as familiar with how all of them needed to work together. Moreover, our customer's business was itself divided into multiple leasing divisions, and no two of them had identical business processes or requirements. To top that off, the way our customer did business often deviated from typical practices in the leasing industry as a whole.

This meant, of course, that our customer was in fact several distinct and different "customers," each having peculiar requirements that were not always compatible with one another. To be sure, much of this was due to the peculiar circumstance of our trying to build a custom product for one company and a generic product for an entire industry at the same time. Nonetheless, we suspect that more often than not, typical customers for larger applications will be more multifaceted than the ideal customer who speaks with a single voice. To handle the competing "voices" among our various customers, we instituted "issue" cards in addition to development cards. The issue card would state the particular business function that needed to be addressed, and a team of business domain experts from our team and the customer's team would meet on a periodic basis to resolve how the functionality should be developed. When some agreement was finally reached, the issue card was then turned into the appropriate story cards. Here again, though, the complexity of our project added another weight that reduced the agility of XP on this project.

6 THE ROLE OF THE CUSTOMER

The fact that our customer, despite its multifaceted nature, should determine the functionality of the system we built was never an issue, and they felt comfortable in that role. But when it came time for the customer team to develop the set of functional tests that would prove the completion of functionality they had requested, their comfort level was much lower. Part of this, we think, is due to the prevalent view among non-technical professionals that computer applications are complex and difficult, so it's OK to use them, but scary to peek at all under the covers. We made an extraordinary effort to convince our customer's team that they needed to not just specify the functionality to build, but also to develop the tests to verify its completion. They eventually did so, but only after having relied on

many, many samples from our own analysts for a long time. They were just not used to the analytic process a typical software analyst would go through when figuring out how many tests covering which functions would constitute a complete verification of this new card.

There was a clear difference, in our mind, between devising a business scenario and devising a functional test. In the former case, one makes sure that, say, when you dispose of a particular asset from inventory, the correct accounting transactions are performed. In the latter case, one verifies everything tested in the business scenario, but also verifies the proper functioning of all negative and atypical actions that could occur in the process, widget action on screens, behind-the-scenes dependencies, etc. Our customer team did not need much coaching to provide us with the business scenarios, but the functional test itself, in all of its detail, required us to do much more training with the customer than we had anticipated. In this respect, we think the typical description of the ideal XP customer working directly with the developer, although surely true in some cases, is not typical and, hence, underestimates the need for the traditional analyst intermediary.

7 IS THERE A PLACE FOR ANALYSIS IN XP?

From the experiences I have recounted above, it should be apparent that our use of XP on a large and complex development project forced us to institute roles and procedures that are not clearly envisioned in the common list of XP practices and roles. Hence, it made sense for us to include a team of traditional analysts in this project to fill these and other related roles. Of particular importance in this case was the complexity of the business logic in this particular application. The customer's team on this project provided considerable guidance in defining what was built, but they needed assistance from business experts with a broader perspective and traditional software analysts in order to articulate clearly and completely in a set of functional tests what the system needed to do.

To facilitate the avoidance of disagreements within our multifarious customer, and to articulate more completely the dependencies among parts described in story cards, we needed to produce more traditional artifacts in addition to the story cards and functional tests. For each story card, we developed a separate, more detailed description of the functionality involved, its business purpose, its impact on other parts of the application already developed, and any additional specifics needed to direct the developers. This document was accessible on-line to all parties involved and often proved helpful in resolving misunderstandings or uncertainties that could not be determined by examining the story card by itself. Were we to initiate a similar project of this size in the future, I suspect we would devote even more manpower to the production of more traditional artifacts of analysis so that the XP practices we found productive could be employed once again successfully at this scale.

8 WHERE XP WORKED WELL

Despite the various ways in which we found XP in need of supplemental procedures and artifacts for our unusual project, we came to appreciate many of its basic practices. The fact that we were forced to articulate and develop the functional tests at the beginning of the development process in an iteration was very healthy. Too often, when functionality is designed first and tests devised only much later after development, there is a disconnect between the original design and the tests. By reducing this time to a short iteration, there is less likelihood for that discrepancy to arise.

The frequency of deadlines in the iterative process tended to keep us focused and productive on the particular cards we had adopted. We tried to find the optimal iteration length for our project, starting first with one-month iterations (which seemed a bit too long), and then changing to two-week iterations (which seemed a bit too short). Our individual focus was also encouraged greatly by the fact that owners of tasks were responsible for estimating those same cards. It was much more difficult for someone to acknowledge that something could not meet a deadline when that confession would also imply the person has estimated the task badly. We soon learned that task estimation and ownership needs to extend not just to developers, but to all roles in the project.

The practice of giving individuals ownership of their own problems also made it possible for several individuals to employ their peculiar intelligences to solve many problems. One case, in particular, stands out in this regard. We attempted to implement one card dealing with a very complicated piece of functionality during iteration six, and it soon became apparent that the original strategy we had developed would be cumbersome and, in the end, perhaps unacceptable. Seeing this, we assigned time to one of our business domain analysts to "think through" the card again and propose an alternative way of implementing the functionality. He figured out a substantially more elegant and efficient way to implement the functionality on the card: something that would not have been possible had we felt obliged to implement exactly what we had been told to do.

This case led us to introduce "analysis" cards in addition to regular development cards. For particularly complex bits of functionality, typically with many dependencies, we would estimate analysis time for someone during an iteration in order to flesh out carefully all of the test cases that would be needed for implementing the card in question. During the subsequent iteration, that card would be played like any other card. The amount of time required to think through a sufficient list of functional tests for cards varied greatly from card to card, so we had to implement provisions like this to accommodate those differences.

9 CONCLUSION

From our experience on this development project, we do not mean to imply that XP fails to work for large and complex application and development. Rather, we wish only to point out that many of the basic practices of XP are quite useful in such projects, but they need to be supplemented with some "heavier" methodology in order to work well. A list of story cards, if it becomes too large and complicated, needs to be supplemented with a holistic "picture" to insure that the cards are managed, updated, and ordered well. More importantly, one must keep a careful watch on the dependencies among stories as the list and complexity of story cards grow. Metaphors can go so far, but the complexity they can communicate is limited. A "customer," if it has many facets, needs someone to facilitate communication among the camps, manage the reconciliation of incompatible voices, and provide business expertise from a global perspective. All of these examples point to the fact that one should be prepared for reduced "agility" from XP, as well as unforeseen challenges, when it is implemented for particularly complex or large application development.

ACKNOWLEDGEMENTS

It was my ThoughtWorks colleague Martin Fowler who encouraged me to communicate this experience to a wider audience, and it was my fellow analyst, Terri Hollar, who helped me articulate parts of that experience. I am indebted to both for their assistance, as well as to the rest of my colleagues on this project at ThoughtWorks for their continual support.

REFERENCES

1. Beck, Kent. *Extreme Programming Explained: Embrace Change* (Reading MA, 2000), Addison-Wesley, 157, 181-190.
2. Beck (2000), 56.
3. Beck, Kent and Fowler, Martin. *Planning Extreme Programming* (Boston, 2001), Addison-Wesley, 47, 63-64.