Unfixing the Fixed Scope Project: Using Agile Methodologies to Create Flexibility in Project Scope

Jeff Patton
Development Team Lead
Tomax Corporation
224 South 200 West
Salt Lake City, UT
Jpatton@tomax.com

Abstract

Although it seems to be common knowledge that it's impossible to succeed in a project with fixed time, quality and scope, we often continue to try anyway. This experience report discusses our successful failure at running fixed time and scope projects. I say successful failure because we actually failed to fix scope but arrived at an acceptable way to vary scope and deliver on time in an environment not normally amenable to variable scope. This paper discusses the methods used and makes recommendations on how you might unfix scope in your development environment.

Keywords

Interaction Design, Agile Methodologies, Extreme Programming, Usage-Centered Design, Scope Management, Project Planning

1. The Failure Mode

1.1 The Scene Opens

My employer, Tomax Corporation, specializes in software for medium sized chain retailers - those with 100 to 1000 physical locations. In theory our company sells shrink-wrap software. The software automates processes from point-of-sale to item maintenance, merchandising, purchasing, receiving and integration with other financial systems. After being sold, the software need only be installed and configured to work in a specific customer's environment. However, most companies who invest in our product would like some part of the product altered or extended to better suit their specific business needs. So, it's generally the case that before the software can be installed and made live in the new customers' company, a development project to complete and integrate these changes must take place. This is where the fun starts.

1.2 Tension Builds

Customers who purchase our software generally have the goal to reduce costs by automating processes that were manual in the past. They may want to replace obsolete systems that, while once adequate, have not changed to meet their current business needs. Whatever the motivation, analysts within our customer's company carefully select new features for Tomax to build that will earn them their desired return on investment. Their feature selections along with expected delivery and installation dates for those features are represented in an ROI analysis. Omitting any of these features, or failing to deliver them on their projected date will call into question this ROI analysis and the profitability of the entire project.

Installing new software requires a major investment the least part of which is the actual purchase price of the software. Companies implementing our software require significant staff to plan and execute the installation, training, process change management, pilot and eventual rollout of the software over hundreds of far flung physical locations. It's safe to say that the resources needed to accomplish those activities exceed the resources Tomax may have dedicated to building the new features requested prior to software installation and rollout.

The seasonal cycles of retail result in times of the year where business is slow and times where business is brisk such as the fall and winter holiday season. During these brisk times, introducing change within a retail organization carries particularly high risk. Consequently, pilot and rollout of new software must take place during the slower times of year.

Plans are drawn up with fixed dates for feature delivery, acceptance testing, pilot and rollout. Dates are driven by the seasonal nature of retail and the over-arching corporate goals for change on or before a given date. Massive

resources are hired or relocated within the company to be in position as dates on the project plan come up. So important are these dates that payment to Tomax is bound to them. Missing the dates may result in our company paying financial penalties to our customer. It's possible that if dates are missed our customer's losses may be so great that suing Tomax for damages is not out of the question.

1.3 We've Made Our Bed

New features have been identified. Development due dates have been fixed based on best guesses for development time for those new features. Detail design begins for these features. Sign-off on this design is important. Should the feature be implemented incorrectly, dates may be at risk and the entire project may be delayed.

We've fixed the timeline for the project.

We've fixed the scope for the project.

The timeline is generally short enough that hiring and training new resources is generally not beneficial.

Quality must be solid since the software we're building is generally mission-critical for our customer.

1.4 Predictably Unpredictable

As the project begins we can depend on surprises starting to emerge.

The detail designs Tomax put together seem to have large omissions in them. Often, some details were not considered or resolved. Even where details were right, time estimates seem to be overly optimistic. As a result development time is greater than originally predicted.

Since dates can't move, if estimates are off the additional development required to complete a feature is accomplished by working extra hours. Development is done hastily. If development does complete before the due date, less time than was originally planned is left for testing by QA. The quality suffers. When customers perform acceptance testing, they find errors. Customer confidence falls and tensions in the project rise.

If development cannot complete on time, the very bad news is conveyed to the customer. The project may be delayed for several months to fall into a different retail cycle at great expense to the customer and Tomax. If the feature in question can be removed from the product, the project may go into production without the new feature. Tomax loses credibility. The team within the customer's organization loses credibility within their company for choosing Tomax as a vendor.

This series of events occurs a little too frequently. Often heroic events on the part of project manager and developers save the project at the last minute. Often software is delivered on time, but customers rely on "work-arounds" for bugs and big usability issues. In reality projects are generally never a complete failure, but, it's often hard to claim success either.

2. Understanding The Problem

2.1 We Know What's Wrong

Commonly repeated dogma in the software development industry has project managers chanting "time, scope or quality, you can have any 2 of the 3." Sometimes "resources" gets thrown in as the fourth variable – and then you get 3 of the 4. But with resources you have Brook's Law to deal with "Adding resources to a late project makes it later." [4] But, we know what's going on here: in our world everything's fixed and there's nowhere for the predictably unpredictable scope changes to escape.

2.2 What Exactly Does "Fixed" Mean?

Expensive resources are scheduled to begin implementation work on a specific day. Customer payments are based on hitting a specific release date. This seemed to cement the *time* part of the triangle pretty firmly.

Quality – well shouldn't that be fixed? Poor quality increases the amount of verification the implementation team needs, thus increasing time. Releasing with poor quality negatively affects our reputation with the customer. At acceptance time customers who see evidence of poor quality become extra cautious, and, well, downright nit-picky about things. They are worried about the bugs they aren't finding. Since no one is willing to go on record with a decision to decrease quality in order to deliver all the scope on time, quality needs to stay as fixed as possible.

Scope pretends to be fixed initially, but inevitably ends up growing. When feature changes arrive, the changes often make sense and generally folks feel like if we'd done a little better job up-front on scoping features, these new features would have been discovered earlier.

In summary, dates are fixed, scope was initially fixed but generally grows and changes, and quality, which should be fixed, suffers. If we are going to make things work, scope is the area where we need to focus attention.

3. Inventing The Solution

3.1 The Plan to Beat the System

We hypothesized that we could rise above this whole triangle thing by getting the scope right. We'd have time,

quality and scope by doing a much better job understanding the scope so our time estimates would be accurate or alternatively we could commit to the appropriate amount of scope given the required delivery time. That sounded simple enough, but exactly how would we go about doing this?

3.2 Enter Interaction Design

Alan Cooper, in The Inmates are Running the Asylum [8], defines interaction design this way: "Almost all interaction design refers to the selection of [software] behavior and their presentation to users." It seemed clear that in most cases scope increased not because we'd done a poor job with the technical design, but that we simply omitted behaviors the system required to meet end-user expectations. Generally speaking, interaction design practices arrive at appropriate system behavior by first focusing on the people that will use the system and what their goals are. Then, given those goals, invent the smallest, easiest set of tasks that allow the people using their software to meet those goals.

Identifying the people involved in a particular business process and their goals may seem too obvious to be a revelation. Perhaps because the approach seemed too simple, we'd never considered it. In the past, analysis focused on capturing complex business rules, documenting complex process flows, trapping fields and validations that may be on a complex form. We raced straight to the difficult details of a business process amassing a lot of information along the way. That quantity of information often led us to believe we had appropriate understanding of the business problem.

While we may have captured an understanding of some interesting business rules, we often omitted entire people and their goals related to the process because those people weren't a direct participant in this particular business process. They may have supplied and maintained information used during that process. They may have had on oversight responsibility for that process. We may have failed to take into account the skills, or lack of skills of the person performing the business process. We may have failed to take into account environmental conditions, for example: does this process take place on the fast paced retail floor during business, or in the back office at a more convenient time. Simply identifying all the people involved in the business process and attempting to empathize with them helped us trap important scope items that we'd have missed in the past.

Constantine & Lockwood's Usage-Centered Design [6] provided an easy to implement process framework to practice interaction design. While attending training with Larry Constantine & Lucy Lockwood, I got the chance to practice the agile form of U-CD best described in the Constantine paper "Process Agility and Software Usability" [7]. This form in particular is highly collaborative and makes good use of card-sorting techniques [6, p83] to quickly and effectively capture the foundational user roles and tasks necessary to determine features and a finished design.

3.3 Enter Extreme Programming and Agile methodologies

Agile methodologies promise improved project performance. They certainly seemed like they'd work well with the collaborative form of interaction design we'd chosen to practice. We chose to use development practices from Extreme Programming [2]. We knew practices such as test-driven development, simple design and refactoring should improve code quality. Short iterations would give us more feedback regarding how on or off schedule we were. Scrum [10] practices such as daily standup meetings and regular customer demonstration at the end of the month would keep us focused during development and keep the customers involved at a minimum of once a month. *Information radiators* like those described in Cockburn's Agile Software Development [5] would keep interaction design information, object models and iteration plans on the wall in plain site where they could continuously inform the team.

4. Applying The Solution

4.1 Aren't We Smart!

Combining this 1-2 punch of interaction design and agile development seemed to be a winner. Initial projects tackled with this approach were delivered on time, with very high quality. Contract obligations were met, customers were happy.

Interestingly, a few unexpected benefits emerged.

- **Interaction design forced needed prioritization.**

  While we identified the people the software served, we also prioritized them. We often asked "Who are the people who must be satisfied for this software to be successful?" Then we applied the some process to tasks those people performed. We asked "What are the most important tasks those people need to perform?"

  Even better than knowing the priority, we had the confidence of watching our customer prioritize people and tasks himself during a collaborative Usage-Centered Design session. Our customer owned scope. Usage-Centered Design card-sorting techniques [6,

p83] engage the customers, make it easy for them to prioritize and relate people and tasks to each other. Like using CRC cards [1] for OO design, touching and handling the cards allows the customer to begin to visualize the people, start to develop an attachment to them. They care that those people's goals are met. The customers no longer sees scope as an unordered list of features. They care more about the high priority people and tasks, and less about the rest. The tasks became our prioritized feature list.

- **XP style estimation attached suitable value to features.**

  Putting a price tag on things changes everything. We involved developers in collaborative design sessions. We then delivered time estimates along with the features. We let customers understand price at a more granular level allowing them to help make trade-off decisions. Could we simplify or eliminate this feature and still allow the person using the software to meet their goals? How about these less important people - can we make them use less automated processes? Paper worksheets? Printed reports? What about this expensive but unimportant feature? Can it be cut completely?

- **Iterative development changes the progress report.**

  "Percent-completes" on MS project plans were replaced by interactive demonstrations. We attempted to complete some features in every iteration. Every month we'd demonstrate the working software. No one cared much anymore about this feature being 85% complete. The working features gave our customers different questions to ask: "Do I like it?", "Did I remember something we didn't discuss during collaborative design?", "Should we change scope?", "Now I understand the impact of that feature to the system."

5. Nothing's Perfect

5.1 Trouble In Paradise and Our Plan to Beat the System

In the previous section I claimed we'd delivered on time with high quality. Well, since I'm a software developer, you shouldn't be surprised to find out that that's only partly true. Initially as we proceeded to iteratively develop features we applied "yesterday's weather" forecasting [3] and quickly realized we weren't going to finish on time.

While I was attending a workshop on XP planning at XPUniverse 2002, Ron Jeffries clearly showed in his Release Results discussion and Excel Model [9] the value of delivering and placing into use the most valuable features first. If we deliver the most valuable features early and start earning that value from them, the remaining features tend to have less effect on the overall ROI of the software project. The assertion was even made that after getting the value from the initial important features that an XP customer may elect to never complete the less valuable features.

Armed with this knowledge, we diligently built highest-priority features first. Whenever we spotted a feature not absolutely necessary to the interaction design's most important people and most important tasks we pushed it to the bottom of the list. We let customers know we were deferring those less important things, and since we were deferring them using priorities they'd set, they agreed.

When we arrived at the due date, features were left unfinished. Strangely no one wanted to talk much about those features. The customer was happy with the product and considered our contractual obligations met. We'd met the schedule and let scope slip. We'd beat the system.

5.2 Fictitious Phase II

In one of our early attempts at managing the project by building higher priority features first, the particular project in question had been broken into two phases, one for urgent delivery this retail season, the next for delivery 6-8 months later. The customer considered tasks unfinished in phase I to be something we'd discuss when we collaborated on the design for phase II. Early in the contractual stage for the work, rough lists of features for phase I & II had been decided. Using interaction design principles we identified some phase II features folks really needed in phase I, and some phase I features that could be pushed back to phase II. By the delivery of phase I no one was really sure exactly what would be in phase II, just that we'd collaborate on it and it would be good. But, everyone was sure that phase I would be usable today. It seemed that knowing there was a phase II, whether we got to it or not, took the tension out of making sure all features were in the first delivery.

As a postscript, phase II has been indefinitely delayed due to budget and time constraints with our customer. It turns out phase I is sufficient for now. Ron Jeffries was right. With no shortage of work to do for this customer and others, the development team wasn't sorry to see the work pushed back.

Reflecting now on that particular project, it's not that the features in phase II didn't have value, just that their value was lower relative to those in phase I. Having two phased releases gave us the opportunity to divide scope into high and low priority buckets. It gave the customer the opportunity to evaluate a version of the product only containing those high priority features. Phase II wasn't really fictitious. It may still be built. But by virtue of it being composed of less important features, its priority

relative to other projects has fallen dramatically. All the while, the customer has the opportunity to earn value on those high priority features.

6. Was Something Else Going On?

6.1 "I do not think it means what you think it means."[1]

We believed that the combination of interaction design practices and agile development practices had enabled us to succeed, and we certainly planned to continue doing things this way. But, while explaining our methodology's success to Alistair Cockburn, he pointed out other mechanisms at work that may have been big factors.

- **Customer Trust**

  By collaborating early and continuously with the customer, we'd proved we could listen to them and let them set priorities. We demonstrated our software to them frequently so they could see we were making progress. In many situations we disagreed with our customer's suggestions and voiced our opinions. In those situations we worked together to arrive at an approach we could both support. We'd established early customer trust and maintained it.

- **Reduced Feature Coupling**

  By breaking the project into a concise set of features, then implementing them one at a time in such a way that they weren't architecturally dependent, we always had a project ready to deliver. By prioritizing most important features first, the product became usable earlier. By not assuming we would be required to deliver all the features, we didn't build in dependencies to code not yet implemented. When the inevitable time crunch came, the customer easily let go of a few low priority features in order to meet the delivery date. Since the low priority features were not tightly coupled to features already completed, it didn't jeopardize product stability to defer them. They trusted they'd prioritized things well for this phase I and would do so again on the next phase.

[1] Quote from Inigo Montoya in The Princess Bride

7. Our Methodology Today

7.1 New Strategies Based on What We Think We Know Now

Based on lessons learned from and reflection on a few projects that have used these methods we've arrived at a list of guidelines we attempt to follow when approaching new projects:

- **Keep design general and scope soft.** Identify people, tasks and priorities. Strive for suitable detail. Even when we think we know detail – keep it out of print. Especially avoid things like literal screen designs, database table designs, detailed validation or business rules. These details are distractions from identifying the necessary breadth of features and their priorities.
- **Recognize customers aren't adversaries.** When we succeed, they succeed. They want to participate and help. Take advantage of them for something other than signing off requirement and design documents.
- **Write a Collaboration Plan.** Detail the participation you require from customers during project activities such as collaborative design sessions, monthly product reviews, and testing and acceptance of delivered software. Publish this plan internally and to your customer.
- **Phase Delivery.** No matter how small the project is, break it into at least 2 phases. Require the customer take delivery of, install and perform acceptance on at least 1 delivery prior to the final delivery. A successful phase I builds confidence in the design, the team and the software. A successful phase I is easier since there's less tension about making sure all scope is present in the delivery. Working with phase I deliverable helps customer identify what scope might be unnecessary and what scope might need to be added.
- **Plan To Drop Features.** When creating release plans for the release of each phase, make sure the release includes some low priority features. Make sure the construction of the software allows for the easy removal or disabling of incomplete features. Make sure delivery is always possible.

8. Reflection

Our particular domain of software development forces unique time constraints on our project deliveries. Our customers require important custom features and demand high quality. We've not found the silver bullet approach yet that helps us break out of the dependencies of time,

scope and quality. However, by understanding and accepting that there are dependencies then working to create flexibility in scope, we're able to successfully deliver projects on time. We've learned that by using methodologies such as Usage-Centered Design, Extreme Programming, and Scrum, along with general principles of agility, we're able to leverage close customer collaboration, early progress feedback, and designs that allow us to omit features. This allows us to soften scope that was once rigid. Where we used to feel happy to survive the delivery of a project, now we're able to enjoy successful delivery with our customers.

9. Acknowledgements

Thanks goes to my supportive team at Tomax Corporation. Special thanks to Alistair Cockburn for encouragement and advice; to Brian Marick for wonderful supportive feedback; to the Salt Lake Agile Discussion Group for their continuous stream of good ideas and discussion; and, finally to Stacy and Grace for their every day encouragement, love and support.

10. References

[1] Beck, K and Cunningham, W., A Laboratory for Teaching Object-Oriented Thinking, OOPSLA Conference Proceedings, October 1989.

[2] Beck, K., Extreme Programming Explained, Addison-Wesley, November 1999.

[3] Beck, K. and Fowler, M., Planning Extreme Programming, pp 33-34, Addison-Wesley, October 2000.

[4] Brooks, F., The Mythical Man-Month, Addison-Wesley, August 1995.

[5] Cockburn, A., Agile Software Development, Addison-Wesley, December 2001

[6] Constantine, L, and Lockwood, L., Software For Use, Addison-Wesley, April 1999

[7] Constantine L., Process Agility and Software Usability, Information Age, August/September 2002, http://www.foruse.com/articles/agiledesign.pdf

[8] Cooper, A., The Inmates are Running the Asylum, Sams, April 1999.

[9] Jefferies, R., Results of Frequent Release, XPUniverse, July, 2002, ftp.xprogramming.com/ftp/ReleaseResults.xls

[10] Schwaber K. and Beedle, M., Agile Software Development with Scrum, Prentice Hall, October 2000