Optimizing Major Release Frequency Using Agile Practices

Mara Yale
The MathWorks, Inc.
Mara.Yale@mathworks.com

Abstract

My experience report focuses on the Image Processing Toolbox story, how we transitioned to agile practices to address the problem of needing more frequent major releases of our shrink-wrapped product, and how we have continually adapted our processes in the context of an evolving business environment. Over the last three major releases of the product, we have evolved from being primarily scope-driven to finely balancing scope and schedule over the entire span of the release. The most recent major release was planned to be diversified in scope and was delivered right on its planned schedule, fifteen months after the previous major release.

1. Background

The Image Processing Toolbox serves scientists and engineers who need to apply image processing techniques to solve research, development, and analysis problems. In The MathWorks product line, many of the add-on products, like the Image Processing Toolbox, specialize in a particular domain. The developers of these products typically have a background in the relevant discipline and not in software engineering. This is true for the developers on the Image Processing Toolbox team, and we have all had to learn software engineering, project planning, and project management on this job over a period of time.

2. Introduction

In early versions of toolbox development, decisions were largely based on what we wanted to add to the product with limited clarity on when we could deliver the planned new features. These decisions about what to build were well informed by customer feedback, but inherently risky because of our lack of skill in managing scope and schedule together.

2.1. Image Processing Toolbox release history

The following summary highlights the scope and schedule challenges we’ve encountered over the last several major releases of the toolbox.

- Version 2.0 changed scope significantly part way through its development from a GUI release to a release focused on improving the performance of the toolbox. Altogether, this took 3 years.
- Version 2.1/2.2 together took a year and added a lot of features. In hindsight the work delivered by this pair could have been called version 3.0.
- Version 3.0 was released about 4 years after version 2.0. Version 3.0 was meant to be the "algorithm release." Actually, original plans included a GUI too and many other features that we have yet to build. The team suffered from considerable turnover during this release, further delaying the release.
- Version 4.0 was supposed to be the "GUI release" in response to version 3 which was the "algorithm release." Hurdles included a 6-month long fuzzy front end, and technology difficulties which threatened to limit the value of the GUI features. This was our first release using agile approaches, which allowed us to confidently change the scope and schedule together to make the release come in 2 years after the 3.0 release.
- Version 5.0 was diversified from the beginning to mitigate against some of the challenges we encountered in Version 4.0. We also used a 5-quarter release framework which provided coarser grained stopping points for evaluation, and adjustment in our release plans. We delivered 15 months after version 4.0 right on our planned schedule of 12-18 months.
2.2. Motivation for process improvement

We were motivated to improve our processes for release management during versions 4.0 and 5.0 in response to these primary problems:

1. We had a pattern of major releases taking too long.

2. We were not skilled at choosing or knowing what an appropriate amount of work would be given our team resources.

3. We had a pattern of worrying about release dates late within a release which was disruptive and did not feel rational and systematic to many team members.

4. We had a pattern of optimistically focusing on a single release goal (algorithms in version 3.0, GUI in version 4.0) which can be inherently risky.

As the following sections demonstrate, we made real incremental improvement in our release management strategies in versions 4.0 and 5.0 by staying focused on solutions that would directly address the problems listed above.

3. Version 3.0 and 4.0 processes

3.1. Version 3.0 processes

We didn’t have many tools or processes for tracking and monitoring our progress during version 3.0. For part of the release, the team lead did keep a spreadsheet of work remaining, but it was overly complicated with fudge factors for vacation time and to compensate for overly optimistic estimates, and as a result it was not consistently maintained and reviewed.

When version 3.0 was released, we received clear feedback from senior management to reduce the time between major releases.

3.2. Version 4.0 processes

Several team members read a variety of Extreme Programming books early in version 4.0 development and we began to experiment with story cards, iterations, interim releases, and standup meetings. Some of these early experiments were not so successful like the “interim release” we introduced which confused team members and seemed artificial because we didn’t actually release. Also, some Extreme Programming principles seemed difficult and artificial to apply to mass market shrink-wrapped software development.

Our first interim release ended in late January 2002. We held a bash and a retrospective. Then we regrouped and planned to have two more interim releases. Plans changed.

In May 2002, almost a year-and-a-half into version 4.0 development, we hit serious performance problems with the GUI tools we had built in Java. We focused our energy on addressing the performance issues for several iterations with limited success. At the same time we also began talking with marketing about release criteria, scope and timing.

In August 2002, we made a key decision to essentially rescue the release by diversifying it. We added some algorithm features, algorithm performance enhancements, and file format features, and together with the basic image viewer, this was enough for us to call it version 4.0. We chose an amount of additional work that we projected would take four more months. Version 4.0 was released nearly two years after version 3.0, a significant improvement in the length of the release.

Throughout versions 3.0 and 4.0 we had tracked tasks and feature work using a spreadsheet. During version 4.0, we also calculated our velocity in terms of Perfect Engineering Days (PEDs) per iteration. It was these estimates of velocity which enabled us to appropriately scope the extra phase that we added to version 4.0. See Figure 1 for a plot showing our progress through this extra phase of version 4.0. This was the first time that we used a plot, prominently displayed in the team hallway, to help communicate to everyone our progress relative to our goals. Note that the slope changes abruptly two iterations before the feature freeze. This was due to deliberate triage decisions we made as the schedule was fixed at this point, so we had to defer some features.
4. Version 5.0 processes

We set a goal at the beginning of version 5.0 of releasing 12-18 months after the release of version 4.0. We also aimed to mitigate against the technology hurdles we faced in version 4.0 by planning some balance of algorithm, performance, format, and GUI work from the beginning. Building on the success of our multi-month phase that we planned and delivered near the end of version 4.0, we framed version 5.0 with a 5-quarter release framework that included a quarter for elaboration, three quarters for construction, and a quarter for the end game. We largely kept to this schedule. We learned to improve our ability to project what we could do on a quarterly basis. We recognized early in the first construction phase that we were carrying along more features than we could do but we continued to do the most important ones first and eventually at the end of the second construction quarter we had a “get real” discussion where in an offsite meeting we reconciled our historical velocity with our ambitions. We pruned our planned feature list to an amount that would fit in the remaining construction quarter. Version 5.0 was released 15 months after version 4.0, making it our shortest major release cycle ever, and exactly within our planned 12-18 month release goal.

4.1 Transition to Version 5.0

In an effort to avoid the long fuzzy front end that plagued us in version 4.0, I led a planning phase for version 5.0. I was eager to try an explicit "Elaboration" phase. We started this in mid-April, 2003, a few weeks before version 4.0 was actually released.

Sometime near the end of our elaboration phase, I posed the question, “What if we aim to release in less than a year and a half?”

We knew from our version 4.0 end game that the end game really takes a full 3 months. We knew from our experiments that elaboration can easily take two months. I proposed a 5-quarter release framework – one quarter for elaboration, three construction quarters, and a quarter for the end game. I presented it to the team as an experiment. Everyone agreed to try it.

We started version 5.0 with the following assumptions:

1. It makes sense from a business point of view to release every 12-18 months.
2. A diversified release is less risky and will please more customers.
3. The quickest route to a high quality release suggests we figure out what's most important to build next and finish that before taking on more projects.

4.2 Quarter 1: Elaboration

During elaboration, we thoroughly reviewed all sources of input for what we might want to put into the toolbox in the future. We started by casting a wide net. We have enough development work to keep us all busy for several more releases of the toolbox. The more challenging part was to decide what to build for this release, to figure out what was most important to users in the near term. We did a number of activities to narrow the scope for version 5.0. And, at the same time, we developed a set of documents that we can go back to when we finish version 5.0 and want to begin to plan version 6.0.

We also developed some themes during elaboration; themes that we think can span multiple releases. In each release, we will aim to add to the toolbox in these diverse areas: algorithms, performance, interactive tools, data import and export, documentation and demos.

We included everyone who wanted to express their opinion about what should be in the toolbox. Our customer facing colleagues in Marketing, Application Engineering, Technical Support, Training, and Consulting were all invited to help us brainstorm and prioritize features.
4.3 Quarter 2: Construction 1

During our first construction quarter (C1), we began development work in earnest and part of that work was to refine our estimates for some of the tasks that seemed fuzzy or ill-defined. As a result our progress, as measured by PEDs remaining, appeared stalled. We completed work and added new estimates to the plan in almost equal amounts. We weren't tracking our progress on C1 tasks separately until we tried to explain our metrics to the VP of Development who was puzzled by the flat curve in overall version 5.0 progress as shown by the bold line in Figure 2 over the first 3 months of the plot. We were able to go back and use our version controlled revisions of our task spreadsheet to figure out the C1-specific progress which is labeled in Figure 2. You can see the curve drops consistently each iteration, though our velocity was still insufficient for the amount of work we had committed to for the quarter as indicated by the large drop in the final iteration. This represents throwing features over into the next quarter through triage decisions.

We assembled a list of all work completed during C1. This list helped us feel good about our progress and helped us choose a more appropriate amount of features for C2.

4.4 Construction 1 to Construction 2 Transition

This transition was surprisingly smooth. We literally added one extra hour-long meeting in addition to our regular iteration planning meeting. During this extra meeting, we considered all the tasks that remained on the spreadsheet and we chose the ones that made the most sense to complete during Construction 2 considering the resources available. We made some effort to defer tasks from the overall release, but we largely deferred these tough choices.

4.5 Quarter 3: Construction 2

With our new metric of tracking our progress on C2-specific tasks, we more actively managed the scope of work during C2 than we had during C1. You can see in the C2 curve on Figure 2 that we recognized after only two iterations that we had too much to do, and decided at that point to defer some work out of the C2 phase. We again made triage decisions during the final iteration of C2.

We again made a list of all tasks that we completed this quarter. These lists and our progress chart were prominently displayed on a bulletin board in the team hallway.

Late in C2, we realized that the transition from C2 to C3 would need more time than the transition from C1 to C2. We no longer had the luxury of deferring the tough choices. We decided to add a phase, a 2 week "Get Real Bridge."

4.6 Get real bridge - C2 to C3 transition

Up to this point during version 5.0, each quarter we had successfully chosen a realistic subset of features to complete and we have largely delivered on these plans. We had been much less successful at "letting go" of features that we hoped would still fit. So as we entered this transition, the list of possible features included roughly 3 times as much as would fit based on our past progress during version 5.0.

From experience on previous releases, we had learned that convergence to a release must be forced and it requires a concerted group effort.

We needed to shift from "adaptive" to "predictive" planning (see Martin Fowler’s commentary on "Predictive and Adaptive Planning" [1]). I proposed that this transition from C2 to C3 required a shift to focus on our strength in being able to accurately pick the most important stuff to do next and therefore implicitly forced us to let go of everything else for this release. In a sense, the sooner we could do that the sooner we expected to be able to begin again with version 6.0.

Each of us spoke of new features that we had added or planned to add to version 5.0. This process of...
sharing mini-advertisements helped the whole team get a sense of the breadth of the product that we were on track to deliver.

We held an offsite meeting for most of the day on Wednesday, January 14, 2004 to kick off the "Get Real Bridge" so that we could all take a step back to assess where we were and where we wanted to go with the version 5.0 release. We evaluated a straw man proposal, which limited the scope to features we had already committed significant resource towards. The meeting was highly successful. Everyone was well prepared and the level of discussion was high. We reached consensus that the release scope and schedule outlined in the straw man proposal made a credible "Version 5.0." We captured and shared detailed notes from this meeting which was particularly useful for interested parties who were unable to attend the offsite meeting.

Since consensus emerged, not much additional work was needed during the "Get Real Bridge." The remaining activities included three one-hour meetings:

- Bug review - January 20
- Technical meeting - January 21, to discuss open issues on modular interactive tools
- Iteration planning meeting- January 23
  - Review development work completed during "Get Real Bridge."
  - Plan iteration 1 of C3.
  - Draft annex to version 5 plan - The annex included items that we really hoped we’d have time to get into version 5.0.

At the offsite meeting, we discussed rational reasons for schedule changes. We wanted to be clear about our decisions regarding schedule and under what conditions we’d consider adapting the schedule again in this release and in future releases. These were the main points:

1. We picked the current schedule. No one imposed it on us. In some sense it is arbitrary.

2. The schedule had already changed in minor ways. We added a "Get Real Bridge" and we added an iteration to C2 to make it end after the holidays. These changes reflect things we learned during version 5.0 about how to structurally manage a release within such a framework.

3. R14 (the general release upon which we depended) had slipped and we anticipated it could slip again. This was outside of our control.

4. Some as yet undiscovered bug in MATLAB (the product upon which the toolbox depends) could break or block our code. If we didn’t discover this until it was too late to get such a fix into the general release, we’d be sadly tied to the next general release which would delay our release by several months. This was our "worst fear."

5. Beta feedback from customers could lead us to decide we can't ship. We would need to explain the decision so senior management could understand and in turn explain as needed. Our team is responsible for such decisions but must be able to give a rationale that makes sense to those above.

6. We could learn something during development that convinces us that something is really worth doing now and it's okay for the schedule to slip. An example of this in the past was when we learned how key it would be to add multi-dimensional image processing support to version 3.0. This made a big difference in the marketing messaging for version 3.0.

4.7 Quarter 4: Construction 3

The third construction quarter (C3) proceeded much like the first two. You can see from Figure 2 that after 3 iterations, we had another mini get-real moment where the curve drops considerably. Ideally, we would have completed feature work at the end of C3. But, as you can see from Figure 2, some feature work spilled over into the end game.

4.8 Quarter 5: End game

We ran a successful bash inside The MathWorks to invite anyone interested to try out version 5.0. This was also a way of qualifying the beta release before shipping it to customers. We then ran a very successful beta program. Our quality engineers did thorough qualification on all platforms we support, and helped ensure that our release was packaged appropriately for web delivery. We fixed as many bugs as we could before the final code freeze.

5. Conclusions

Over a series of releases, we have adaptively improved our ability to manage both scope and schedule. Some key strategies that we have discovered work for us include:
1. Consider overall release schedule from the beginning.

2. Use 3-week iterations to track and monitor team velocity. Reestimate tasks as needed based on new information each iteration.

3. Use 3-month phases to reflect on release goals and make more major adjustments as needed.

4. Share data with the team through graphs and reminders at each iteration meeting of where we are in the overall context of the release. This makes the decision points more obvious, and helps set the conditions for consensus to emerge naturally.

Some of the learning challenges that we have faced in adopting agile techniques on our team are:

1. How to get each team member to sign up for an appropriate amount of work each iteration. This is really something that everyone needs to learn through practice. We discovered that a couple of questions at iteration meetings can be useful here: “Does anyone have too much to do this iteration?” “Does anyone need more to do this iteration?”

2. How to forecast what we can do in a quarter and in a release. Once team members get reasonably good at estimating what they can do in an iteration, the estimates are fine-grained enough to enable extrapolation to a quarter. It’s still a challenge to get it exactly right for a whole release, and that’s okay because we want to keep the flexibility to adapt the release based on lessons learned along the way. That’s a big part of what agile approaches mean to us.

3. How to focus a release on a small number of meaningful themes. We want to find a balance between too much focus on a single theme which can be risky and too much diversification which can make the overall release message less crisp.

Our team continues to develop new agile skills with each major release. We have evolved towards using more agile practices over several releases and are now recognized as a resource within the company for our capacity to work in an agile way as more teams are being encouraged to consider agile approaches.

By managing a release on different time scales (3-week iterations, 3-month phases, 5-quarter release framework), it’s simple to make decisions in a rational, open way that leads to high quality, well balanced releases being delivered on time and meeting the needs of our varied customer population.