

“Ready-to-Roll” Boxcar Development – a Flexible, Quality-Weighted Process

Russell R. Hill
WarpTime
russell@rrhill.com

Abstract

In January of 1996, Intuit’s QuickBooks team was faced with an aging code-base using a custom Mac/Win GUI toolkit, a large and rapidly growing customer base, and a rapidly growing and product-inexperienced engineering team. To increase the product’s quality and feature predictability while retaining its ship date rigidity, we created “Ready-to-Roll” Boxcar Development.

The process enabled the defining of each new feature, enhancement, or engineering re-architecture as a set of boxcars on the product train. A “coupled” boxcar was rapidly brought to a supportable level of quality, or “decoupled” for re-evaluation and re-engineering. Frontloading the highest priority boxcars increased predictability of the product train’s contents, while the process allowed for greater flexibility with respect to overall content.

“Ready-to-Roll” Boxcar Development kept the product within 2-3 weeks of being supportable and shippable. The process focused on individuals and interactions, working software, customer collaboration and responding to change. Better yet, it worked!

1. Introduction

Since 1987, I have worked in organizations on large engineering projects (200+ people) employing agile methods on older code bases. The management processes have generally been developed on-the-fly and varied according to the composition of the organization and project needs.

In the last year, I have begun to participate in the discussion of agile methods with people outside of my work environment. Most of these discussions have been focused on applying agile methods to new code bases and/or smaller teams. XP discussions have reflected the same constraints. As I have shared my experiences, I have been encouraged to share more.

“Ready-to-Roll” Boxcar Development is one of the more radical approaches that I have helped to develop, so it seemed a good place to start. In addition, while the

method predates XP, it has a philosophical connection with respect to building quality in, testing early and often, and having a code acceptance procedure. Inspired by Steve Maguire’s [Debugging the Development Process](#), we started working with Boxcar Development at Intuit in 1996. The process covers development from inception to shipping product. Most importantly, it is scalable. We used it while Engineering (Development (Dev), Quality Assurance (QA), Release Engineering (RelEng), and Documentation (Doc)) grew from 50 to over 200.

2. The core concept

In Boxcar Development, the product release is the train and each release of code is treated as a boxcar. The idea is to keep the train in “ready-to-roll” condition, by limiting the size of the boxcars and the duration of the “coupling” process, i.e. the code release and testing. If a boxcar does not achieve a supportable level of quality at the end of its coupling period, it is “decoupled”, i.e. the code was removed.

3. What problems were we trying to solve?

The list was quite long, but the biggest issues that management team wanted to address were:

- *The reality that we would be working with a team, 50% of whom would be product-inexperienced new hires. Most would have worked on the product less than half the 8-12 months it usually took to come up to speed on our large old code base with its custom development tools. We felt this increased the risk of making big mistakes.*
- *The reality that patching in a mostly pre-internet era was prohibitively expensive, given that it would mean shipping diskettes to our large customer base (1M), plus our new customers (300K/yr). Thus product quality was a critical factor in our financial success.*
- *The reality that buggy and/or confusing software results in more calls to Tech Support, which drives up supports costs and reduces customer*

satisfaction. Since customer recommendations moved 70% of our new customers to purchase the software, product quality was underscored again as a critical factor in our financial success.

- *The reality of a narrow release window, which if missed, would push the Division over a revenue cliff, losing 30-35% of its annual revenue.*
- *The reality that Marketing's long-lead product tours required at the development mid-point a guaranteed coherent feature set around which to build a strong marketing message and substantial enough to attract 35-40% upgrades.*
- *The reality that rapid growth was displacing our small team culture and its positive attributes.*

4. Why the focus on “ready-to-roll”?

By focusing on “ready-to-roll” we would be building quality into the release during the entire cycle, rather than focusing on stabilization in the closing phase of the release as we had traditionally done. While we had always tested around each new code release, the “ready-to-roll” concept meant two significant changes: we tested deeper and broader; and we prioritized fixing supportability bugs over writing new code.

By building quality in early, we believed we would reduce the risk of late release cycle surprises, which could jeopardize hitting our release window or risk features we had already guaranteed. It also appeared to provide us with more flexibility for adding features as the team grew.

By focusing on “ready-to-roll” we also believed that we might be able to better handle incorporating new team members. The process would create a tight feedback loop by breaking the work into smaller pieces and testing for and fixing deeper bugs sooner. We hoped this might shorten the learning curve. We also believed it reduced the risk of assigning new team members more significant features to work on. This increased the team's flexibility.

Finally, we saw this approach as a way to make scalable some of our small team “best practices”, especially those which put individual contributors back in the driver's seat, something that had started to disappear in our growth during the prior release.

5. Major supporting communication concepts

The Boxcar Development process would not have been successful without some supporting communication concepts that had been developed separately and, excepting one, had already been in active use.

5.1 The Feature Team and Their Meeting(s)

The Feature Team (FT) and Feature Team Meeting(s) were a part of Intuit's development process long before

we had a need to create Boxcar Development. In prior releases, the FT was limited to Development and Marketing. For Boxcar Development, the FT became a cross-functional team of 4-12 people, the smaller the better. They usually included one each from Marketing, Dev, QA, Doc, and Technical Support. The FT was responsible for understanding the Marketing/ Customer objective(s) and then developing a feature design that addressed those and any other objectives the team thought important to the process of defining the feature and its design. Prototyping and usability testing were a part of the design development process. The team held together until the feature got through usability testing.

Generally the strongest subject-area expert drove or played a significant role. Organizational affiliation was secondary. The driver plus the Doc team member (and developer when not the same as the driver) wrote the GUI and user-experience documentation and prototype that became the road map for the feature. The documentation was kept light-weight, and usually flowed out of the prototyping for usability testing. For most features, its focus was on the GUI design and feature flow. The prototype usually grew into the feature. The team decided if any other documents were needed.

5.2 DACI – Driver(s), Approver(s), Consultants and Informed

DACI was the one new supporting communication concept. As the Engineering organization grew beyond about 40-50 people, we found managers seemed to be taking on more of the driving power of the releases and becoming bottle necks to the project flow. In addition, the individual contributors (ICs) were feeling that they had less and less decision-making authority and their sense of ownership/stake in their work was decreasing. Due to this, we had already lost some and were at risk of losing more of our valuable senior ICs. We needed to find a scalable way to keep the ICs empowered, but ensure that important decisions were getting made and/or reviewed by the right people. In addition, we needed a way to spread the expertise of our senior members across the teams without them or others feeling the senior ICs and managers were responsible for driving everything under consideration.

The Marketing Team provided us with a solution: a decision-making model called DACI. Separately they had been exploring this roles-based model to better structure their communication and to clarify decision-making responsibilities. We adopted the DACI model as follows:

Driver(s): 1-2 persons responsible for driving a feature from concept to signed off.

Note: usually ICs.

Approver(s): 1-3 persons with the authority to sign off on design decisions.

Note: The Approver(s) was not necessarily a FT

member. Bigger features or the ones connected to long term product direction, likely a senior manager for an Approver. This was for two reasons: 1) it provided a cross-check on the decision-making process; and 2) some features were of strategic importance and needed senior management buy-in/sign-off.

Consultants: “Experts” to provide the FT with information or analysis for the design phase, but not needed as fulltime members of the FT.

Note: External consultants were occasionally used, but the primary purpose was to enable our senior ICs and managers to spread their knowledge and insight without having feature responsibility. Limiting responsibility in this way proved critical to Boxcar Development’s success.

Informed: a list of persons or organizations affected by the FT decisions, who therefore needed to be informed by the driver or designated member of the FT at whatever moments were appropriate in the process.

Note: Beyond the more direct functional informing, FTs would generally present their findings and features to the whole organization at the monthly Division meetings.

5.3 Project Review Meeting(s)

Weekly Project Review Meetings were also a part of our extant agile process, which we continued to use with Boxcar Development. When Engineering was small, this was just an Engineering managers’ meeting, i.e. it included Dev, QA, and Doc. By the time we began using Boxcar Development, Engineering had grown to the point where the “meeting” was in fact a series of meetings, one for each project sub-team: Core Features, Payroll Services, Banking Services...

Who attended? The senior managers for Dev, QA and Doc attended the entire series of meetings. Each sub-team’s group of managers ran the meeting. The sub-team managers attended just their content-specific meetings unless otherwise requested. Marketing folk and the Tech Support Liaison Manager periodically attended meetings on an as-needed or as-requested basis. FT drivers and senior ICs attended on an as-requested basis.

What were the meetings’ goals? The meetings provided the sub-team’s management a regular opportunity to identify and to wrestle with their project team’s important problems. It was also a way for the senior managers to periodically build a picture of where the whole release was, with respect to the release objectives. This was important because initially we did not keep a master schedule of the whole release. Given the dynamic nature of our development process, we

initially felt this was not sustainable without a lot of effort. The benefits seemed insufficient to justify the effort. However, as the project’s size and complexity grew, we shifted our stance on this, adding a dedicated project manager to the team to build and maintain a master schedule.

What was the meeting’s format? Generally the meeting began with a brief update of the week’s significant accomplishments. It then moved on to a more detailed update on problems or concerns that managers felt were significant. There was always a Q+A drill-down time from the senior managers. We spent the bulk of the meeting looking at project flow and problems. When it was helpful and important, these discussions went right down to the individual IC level. Every 3 weeks or so, we attempted to look at where we were in terms of the overall release schedule and to review the implications for the release objectives.

6.0 Our Rules for “Ready-to-Roll” Boxcar Development

From the start, we attempted to lay out a set of rules which we tweaked as we gained experience. We felt this was necessary because we were simultaneously changing the way we were working and bringing so many new people onto the team.

1. Prioritizing features:

a. All potential features and re-engineering work must be “bucketed” into *must-have*, *want*, and *nice-to-have*, and prioritized within the buckets.

b. *Must-haves* cannot exceed 50% of the available developers’ resource time.

Note: This 50% rule is rigid in direct relation to the rigidity of the ship date. The more flexible the date is, the more flexible the rule can be. It provides an early reality check on the release objectives as expressed through the features list.

2. Schedule line items and boxcars:

a. No line item or boxcar can to exceed 5 days or be less than ½ day.

b. Every feature, enhancement, or re-engineering begins with a default set of line items with an adjusted set of durations.

c. Driver(s) and/or line item owner are responsible for adjusting the line item set and their durations as soon as they can be reasonably mapped out.

Note: Usually as the design process unfolds, the work to be done becomes clear.

e. Managers are responsible for getting the changes into the schedule as soon as they are available.

e. Approval of a feature design includes approval of its schedule and resource costs.

3. Test plans:

a. Are developed while the feature is designed.

- b. Are meant to be high-level, light-weight docs 1-2 pages long. They should not include test cases. They should lay out the types and areas of testing needed for a boxcar to be validated as achieving a supportable level of quality.
 - c. Are reviewed and approved by the feature developer(s) and QA manager(s).
 - d. If time allows, test plans may be modified and expanded as the feature is developed.
4. **Testing:**
- e. Starts as soon as reasonable with private builds for QA, i.e. before a boxcar is coupled.
5. **Releasing new code to main code base:**
- a. The “coupling” period starts with the release of new code to the main code base.
 - b. A boxcar’s Dev & QA engineers’ *highest priority* is to get the boxcar to a supportable quality level.
 - c. Boxcars that might endanger the general product stability must be identified to management, so they can consider branching the code base.
Note: Generally, this would be for a major re-architecture to ease the codes’ removal.
6. **Removing newly released code:**
- a. If a boxcar fails stabilize within its specified window, it is evaluated for “decoupling”.
 - b. If the evaluation team cannot establish a high-confidence plan to stabilize the boxcar within 1 additional week, it must be decoupled.
 - c. If boxcar fails to stabilize within the extension window, it must be decoupled.
7. **How to start the process:**
- a. **New Feature:** Starts with the creation of a Feature Team (FT) with identified Driver(s) and Approver(s). The FT’s deliverables are a GUI and user-experience doc that becomes the map for the feature, a prototype for usability testing and any other doc(s) the team deems necessary. These docs need not be maintained once the feature successfully completes usability testing.
 - b. **Enhancement to an existing feature:** Most do not require a FT. The developer and an Approver are assigned. Work begins with a problem definition from Marketing or Tech Support. In most cases, usability testing is not required. A light-weight feature doc is required, but is not maintained once the design is approved.
 - c. **Engineering re-architecture:** Starts with a team from Dev and QA with identified Driver(s) and Approver(s), and a problem definition from Dev, RelEng or QA. The deliverables were left to the team to map out. They produced and maintained whatever documentation they deemed necessary to communicate the concepts and particulars up and across the Engineering organization.

8. **Non-FT Approvers:**

Are required to attend the first FT meeting where the feature’s objectives are laid out.

9. **Prototyping and usability testing:**

New features must include at least 1 round.

10. **Beta Waves:**

Should be able to ship 1 week after last loaded boxcar for the *Beta Wave* has successfully coupled.

11. **Final Boxcars:**

a. The last major boxcar could be loaded no later than the start of the release shutdown phase.

b. The last minor boxcar could be loaded no later than 4 weeks before the manufacturing date.

Note: The last week of the shut down period was targeted to get no more than 5 bug fixes, and preferably none. It was a testing safety pad.

12. **Code Reviews:**

All code releases during the last 4 weeks before the manufacturing date **must be** code reviewed.

7.0 The Project Flow

Our marketing plan required an annual release with an 11 month release cycle. The project flow was broken down into 4 major phases: release definition, project planning, development, and release shutdown. These phases often overlapped.

As we were a marketing-driven organization, the Release Definition Phase was largely a marketing-driven phase, but each group had work to do (see *Roles and Responsibilities by Phase*). It began 3-5 months prior to the start of the project and lasted 4-7 months.

The Project Planning Phase began during the last few weeks of the prior release and ran for 6-8 weeks. This was mostly an Engineering driven phase, but there was plenty of interaction with Marketing as the release definition got modified based on improved understanding of the cost and feasibility of the desired features.

The Development Phase formally started with the establishment of the first feature team, generally a couple of weeks after the prior release shipped. It ran until the start of the Release Shutdown Phase, but minor boxcars could still be released for a few more weeks.

The Release Shutdown Phase had 3 major goals: to give the developers time to “polish” the release by fixing 800-1500 minor bugs; to give QA time to do a final holistic integration/regression test; and to bring control to code changes by steadily reducing the number accepted as we approached the manufacturing date. It also allowed lagging boxcars or boxcars created as a result of *Beta* feedback to get coupled. The phase’s duration was determined by our best guess at the start of the release as to how much time we needed to achieve these objectives. It always included a 2 week window for slippage. [As we were aware that using the slippage would cost the

company an unrecoverable revenue opportunity for the year, we rarely used it, but kept it as a safety pad.] The start date was simply set back from the “drop dead” date.

Some projects required more changes than an 11 month cycle allowed. In such cases, we had the two releases going on simultaneously: Release A would go out a year before Release B. Release B had boxcar-related integration periods. These integration boxcars were the only ones that were allowed to be larger than 5 working days and got as large as 10 working days. During these periods, RelEng and Release B’s Dev team would integrate all of Release A’s changes since the code base’s last integration branch. Release B’s Dev Team and small QA team would execute the relevant tests to assure that boxcars from the earlier release still passed the tests. The Release B Team could very briefly borrow members of Release A’s QA team during the latter part of these integration periods to increase the confidence in the success of the integration boxcar’s coupling.

8.0 Roles and Responsibilities by Phase

8.1 The Release Definition Phase

Marketing Magic - Marketing created a release leadership team to work on the following release definition activities:

- Analyze customers’ wants and feedback, reviewers’ feedback, longer term product vision, past feature ideas, competitive issues, and new market opportunities.
- Conduct market research on the most promising ideas to identify target customers and to determine anticipated revenue for selected new features and feature enhancements.
- Analyze the developing feature list for potential broad marketing objectives and messages.
- Conduct market research to identify the strongest messages and the most appealing features and feature groupings.
- Work with senior management to settle on the release themes for the various target audiences.
- Prioritize the list of feature and enhancement candidates into *must-haves*, *wants*, *nice-to-haves*, *maybe-in-the-futures* and *not-worth-its*.
- Pitch their prioritized feature list to senior managers and key senior ICs. [There is a lot of give and take in these meetings.]
- Update the long term product vision document with changes, if any.

Engineering - During this phase, management and senior ICs:

- Respond to feasibility inquiries from Marketing’s release leadership team.
- Build a list of engineering and re-engineering

candidates: tools to implement, enhancements, code rewrites and updates.

- Prioritize the list into *must-haves*, *wants*, *nice-to-haves*, *maybe-in-the-futures* and *not-worth-its*.
Note: Items could be tied to Marketing’s features.
- Pitch their prioritized engineering list to Marketing and senior management. [There is give and take in these meetings as well.]

8.2 The Project Planning Phase

Managers - During the last month of the prior release Engineering managers met as frequently as time allowed to go over the feature and engineering lists to:

- Do high-level, back-of-envelope resource and time estimates for the 1st draft prioritization. Consultations (generally outside the meetings) were done with senior ICs to help “ballpark” the work.

Note: These estimates helped the prioritizing of the features. Often the team lacked the content expertise to do more than a ballpark estimate, but this step stimulated communication and thought about the problem spaces represented.

- Figure out *Beta* feedback requirements and the time customers need to be using the feature to meet those requirements.
- Figure out the release shutdown requirements and the time the different orgs needed to meet the requirements.
- Sanity check the total estimate for the *must-haves* (must not exceed 50% of available resource time), and that both the particular and number of *must-haves* whose expected delivery dates exceed the first *Beta* wave would be manageable.
- Identify Drivers, Approvers and FT members for the top *must-haves*, so FT meetings could start.
- Identify usability needs and pass the info on to the usability lab for customer identification

Once reasonable 1st-pass of the feature list was in place, the managers worked to:

- Identify the Driver(s) and Approver(s) for the rest of the *must-haves* and most of the *wants*.
- Start mapping out the rest of team requirements and composition for these features.
- Start building the Product Development schedule, populating it with the template set of deliverables used as a starting point for each feature/enhancement/re-engineering item and adding all milestone information related to the *Beta* waves and release shutdown process.

Note: The choice of scheduling tool does not matter, so long as you maintain whatever you choose.

Individual Contributors - During this period, ICs

worked on a bunch of fill-in such as to:

- Improve code or test documentation for features completed in the course of the prior release.
- Upgrade tools for the next release cycle.
- Visit customer sites to learn more about our customer base.
- Work on the Tech Support bug fix requests that were already approved for the release.

8.3 The Development Phase

Feature Team Meetings - FT Meetings were an intensely productive period central to every new feature.

- FT Meetings started the process for a feature. Approvers were kept in the decision loop as necessary through the process.
- FT Meetings clarified objectives, explored the problem space, and came up with a solution to prototype. The processes of identifying a solution and prototyping were often pretty seamless.
- The FT deliverables were scheduled and tracked. Most feature designs and prototypes were completed in the course of a few meetings and a bunch of work outside of the meetings, during a period of ½ - 2 weeks.
- The prototype was run through usability testing. In the most extended cases, the feature design went through several rounds of prototyping and usability testing, often beginning with running customers through a series of printouts representing the concepts, and eventually ending with the running of customers through computer-based mock-up of the feature's interface and flow.
- As the team closed in on a solution, its members would identify schedule line items with estimates to replace the existing feature placeholders.
- Once the feature went through usability testing, design adjustments were hammered out, schedule line items were adjusted and approval was sought.
- Once approved, the FTM phase was done.

Individual Contributors – Ics, with help from managers where necessary:

- Drove or participated in the feature design.
- Taught their managers about their features.
Note: This was structured this way under the belief that making the IC the teacher helped the IC to master the material. The manager also learned the degree of mastery and thoughtfulness the IC was applying to their part of building the feature. Managers did supplement this with the reading of feature docs and talking with other FT team members.
- Defined the boxcars and other deliverables for

the schedule.

- Established the time estimates for the line items.
- Did all of the work necessary to get a boxcar successfully built and coupled.

Managers - For Boxcar Development, managers:

- Worked with their ICs to translate features into schedule line items and boxcars, and to help new hires build reasonable estimates.
- Updated the schedule for their ICs deliverables.
- Helped/coached their ICs in identifying and working through problems that came up.
- Represented their ICs work at the weekly Project Review Meeting.
- Participated in FTs, if needed, as members or consultants with content expertise, or as coaches to new hires who were learning the process.

Senior Managers - Senior Managers used the weekly Product Review Meetings as well as scheduled and ad-hoc 1-on-1 meetings to:

- Build and keep the big picture, i.e. maintain an understanding of where the whole release was and where it was going.
- Keep the flow moving through the feature priority list with a watchful eye on the *must-haves*.
- Help the managers and senior ICs recognize potential and real problems before they disrupted the release flow.
- Help keep the team focused on “what’s important” and “how much is enough”.
- Help the team to make the tough decisions.
- Help shoulder some responsibility.

Technical Support (TS) - Throughout the first ½ of this phase, TS Liaisons worked with the FAQ (Frequently Asked Questions) team and the TS database to:

- Identify and prioritize top 100-150 fix/enhancement requests (either from customer call analysis or customer support needs for reducing cost of time-consuming issues). These were reviewed by marketing for a message/theme for getting customers to purchase an upgrade.

8.4 Release Shutdown Phase

“Ready-to-Roll” Boxcar Development did not have an impact on the roles and responsibilities of this phase. These remained unchanged.

Boxcar Development did affect the duration of this phase. Prior to using this process, the Release Shutdown Phase grew from release to release as more legacy code/features needed to be regression tested. Once we implemented Boxcar Development, we found we were able to shrink the duration of this phase, eventually by about 40%. In addition, we were able to add new minor features later than in previous releases.

9.0 Problems during implementation

As with every new process, there were anticipated and unanticipated problems to handle.

Feelings of loss of control. As anticipated, both Marketing and the Engineering managers initially felt they had ceded control of the Product Development process. The feeling was real and based in fact. Marketing felt their loss of control in that: 1) they were not used to having to bucket and prioritize the features without knowing where the precise cut off was; and 2) the process ultimately gave re-engineering tasks more weight than they had historically had. Engineering managers felt that they had ceded control both across organizations and downward to their ICs in that: 1) the Boxcar method did return to Marketing some of the workflow control that Engineering had gained as it grew; and 2) the Boxcar method had returned decision-making authority to the ICs through their becoming Drivers and Approvers, through the reduction of management participation in FT meetings, and through the process assertion that the ICs needed to take primary responsibility for identifying and defining the work to be done.

The big fear in losing control is the cost of failing, personal and corporate. As a group, we mostly believed that “control” in any precise sense was an illusion when concerning large-scale software development, especially when working on problems no one had previously attempted to address, and particularly with a team of people working on a large, undocumented and unfamiliar code base. As problems cropped up, we focused on solutions, not blame. We also reminded ourselves that our older approach had been taking us further from our ideals of an empowered, productive, fun and flexible work environment. Fortunately, the outcome was better for the whole organization. Once people learned how to work with the process, people found in the end its flexibility gave them more control where they wanted it

Getting Marketing to limit *must-haves* to between 35-50% of the available developers’ resources time. Every release Marketing needed to identify the broad marketing themes that they would use to sell the product to its target audiences: new markets that the product did not previously address, existing customers to get them to upgrade their software, and new customers in existing markets. Each audience had a list of features that was thought necessary to get them to buy. With the limited time and resources to do customer research on audience and feature combos, they were not always able to divine with precision, which of the pieces of the bundle of features were most compelling. In addition, market research can simply differ from reality.

In the past, Marketing gave the bundled list to Engineering and only had to identify the least important members of the list, the *nice-to-haves* and the least

important *wants*. Now we were asking them to put their Research and intuition on the line and bucket and prioritize the other items into *must-haves* and *wants*. Furthermore, we were asking them to limit the *must-haves* to 35-50% of the development capacity.

As anticipated, this was and remained a challenge for each new Marketing release leadership team. Some were more successful than others. In exchange, however, Marketing got release content flexibility. By not requiring the entire feature and resource plan to be laid out at the beginning of the release, Marketing found Engineering could easily reshuffle the priority of the remaining work to address new opportunities, their latest research, and/or competitive situations as they arose.

Difficulty adhering to our guideline for sustained communication. We believed that sustained communication happens best when the number of people that an IC needs to regularly interact with is kept small (3-4). The combination of not mapping out the whole release resource plan in advance and the addition of a significant number of new hires, who needed training, led us to anticipate this as becoming a major weakness in the whole method.

Working through the process, we discovered that this wasn’t the big issue we had anticipated. The clarification of the Consultant and Informed roles in the DACI model helped to identify and limit the relationships requiring sustained communication. The Boxcar method, of developing features to a supportable level before moving on, turned out to reduce the need for maintaining communication after a boxcar was coupled, thus freeing all parties to let go of existing communication demands and thereby be free to make new ones. Over the course of a release, the ICs needed to interact with a larger number of team members than would be normal in a small team environment, yet the number of relationships that they had to sustain at any time was kept within the guidelines.

The management team still made an effort to keep the IC “pairings” of Dev and QA together as much as possible over the course of a release. Overall, we did do more moving around and thus gained functional flexibility that we could retain as we grew. Despite this change, ICs felt the method reduced their communication burden and thus allowed them to be more focused and productive. The main exception to this was for the senior ICs who were called in as Consultants.

Senior ICs needed more time needed for consulting. We really underestimated this. There were 3 probable reasons for this:

- We had identified and clarified a previously hidden role and therefore people felt freer to avail themselves of the experience and wisdom of the senior members.
- We had more new hires and we were giving them more significant responsibilities than we had

previously – IC Consultants were part of their support system.

- Boxcars that failed to couple often required Sr. IC involvement for evaluation for next steps.

As we proceeded, we found the benefits of Senior ICs spending more time consulting were outweighing the costs, so we scaled back our expectations for an individual's feature output.

Difficulty managing the combined schedule.

Managing a flexible schedule for a large, complex project involving scores of people (eventually over 200) is a challenge. There is no way around this. Worse yet, the challenge scales. That said, it was manageable.

Initially we made no scheduling tool requirements. It was the senior managers' responsibility to keep a handle on the big picture. For a couple of releases we had been able to do so without a unified schedule. About 3/4 of the managers used spreadsheets and 1/4 used Microsoft Project. Each method had its strengths and weaknesses.

By the third release, the complexity of the project had grown to the point where senior management felt they could not longer maintain the big picture in this ad hoc fashion. We needed to settle on a unified process that allowed us to evaluate and compare the stories of the Project Review Meetings consistently within and across the sub-teams. There was a camp for MS Project and different camps for the different spreadsheet models in use. Senior Management decided to use MS Project to gain a unified scheduling process. We added a dedicated Project Manager to maintain the schedule.

Integration testing ownership for later boxcars was murky. Who should be responsible for the integration testing of boxcars that were coupled later in the release, but were dependent on functionality provided by boxcars that were coupled earlier?

Initially, we made it the responsibility of the later boxcar testing owners to capture whatever information they needed on paper from the testing owner of the earlier boxcar. This proved way too time consuming. The ICs doubted the effectiveness of this strategy as well.

We made a decision to limit the amount of time spent planning and instead opted for just listing the affected components/features and types of issues. The set of testing owners and their managers would negotiate the division of labor for the integration testing and schedule line items were created for each testing owner. This made the testing more ad hoc. If there was an area of special concern, the QA managers could assign a senior QA engineer to discern "reality" with respect to the state of the feature. This improved in subsequent releases, but remained a challenge. Any problems missed at the time, however, got caught by the time of the final regression. Thus while it remained conceptually untidy, it did work.

Content flexibility had costs in Beta planning.

Boxcar Development allowed us to juggle the content of

the *Beta* waves quite easily. This was a boon due to the unpredictable output of a new team. We had not anticipated, however, its implications for the *Beta* team who identified and prepped *Beta* customers, or for those doing *Beta* support. They did not have processes in place to handle the changes in content, in timing and in number of waves. Eventually we worked out more flexible processes with them as well.

10.0 Benefits

10.1 Major Benefits

Quality was sustained. In the face of the rapid growth of the Engineering organization, the product size and complexity, and the customer base, this was a substantial benefit. Over time, our per customer support costs were dropping, increasing our profitability.

The quality of the product's design and the architecture of code base were improving. Our senior developers were spending more time working on architectural and design issues. They were used more widely upfront as design consultants as well as during the "coupling failures". The ability to have Engineering *must-haves* allowed them to finally move the code base from its late-80's roots to mid-90's technology. QA and Technical Support were getting involved earlier, i.e. during the FT design phase. Their feedback from the customer and testability perspectives aided in these improvements.

Individual contributors felt they had regained the empowerment of a small team environment. ICs were drivers again, and in some cases approvers as well. Across the organization, ICs felt the process renewed their sense of ownership of their work. Senior ICs were actually doing the architectural designing that they knew the code base needed.

Managers' jobs became more manageable. By focusing on management issues and getting less involved in the product design space, the management bottlenecks that had occurred in the prior two releases mostly disappeared. Some who became managers to gain authority in the design space, returned to senior IC roles, which generally better suited their personal goals anyway.

Release predictability was improved. There was less nail biting in the latter part of the release. The upfront prioritization gave us comfort that the most important development work was in the release and was stable early. We did not have to wait until *Beta* or the later regression testing to know if we would really make our manufacturing date. We didn't experience late instability surprises, despite our rapid growth.

The method was scalable! The Engineering organization grew from about 50 to over 200 over the next 4 years. The process still worked despite the increasing complexity of the work, the implementation of

off-cycle releases, and the requirement to integrate with a wide variety of interconnected services, including 3rd party solutions.

10.2 Other Important Benefits

Flexibility. It came in many forms:

- Team composition was flexible and easier to scale to the work requirements. DACI helped.
- Good hiring is by nature unpredictable. Boxcar Development was effective at plugging new hires into building new features, to the benefit of all.
- Allowed us to add features later in the cycle than historically, both due to the methodology and to its shorter regression test period.
- The deeper testing of Boxcar Development increased our confidence in our ability to release hidden partial features that could be completed and exposed with off-cycle patches. This ability became critical as the demand increased for off-cycle web service releases, particularly ones that involved working with third parties.

Improved communication. The Boxcar Development method improved communication in many ways already discussed.

One of the method's most dramatic and useful features was the ability to pull a boxcar off the train. The desire to have a smooth coupling made for better communication between team members. When important problems arose, managers and senior engineers got involved sooner in evaluating the problem and consulting on solutions. The end result was better features, better code and less lost time in attempting to stabilize code that really needed to be yanked and rewritten.

Shorter training cycles and better integration of new hires. The management team took a number of steps to cut our unacceptably long training period by 50-66%. Boxcar Development helped these efforts in 3 ways: 1) increased the availability and contact with senior ICs as consultants; 2) made new hires bigger stake holders from the outset, which increased their own incentives to ramp up faster; and 3) over the long haul, increased the re-engineering efforts to bring the code base into the 90's so that there was less for new hires to learn. In addition having better defined processes allowed the new hires to more quickly grasp expectations when they were handed feature responsibilities. This in turn allowed us to give more new feature responsibilities to new hires and thus to derive more immediate benefit from our growth.

Clarified expectations with respect to prototyping, documentation and wider communication. The Boxcar Development rules were our first attempt to clarify expectations with respect to documentation and "working software". Prior to this approach we had left these decisions to the ICs and managers involved. Our

organization's rapid growth meant we were incorporating a lot of people with disparate experience and opinions regarding both. These differences of experience and opinion were a source of tension within the organization, and created challenges to effectively incorporating new hires. After working with the clarifications and the process, the tensions and the challenges diminished.

Engineering architecture documents started to exist. All new core architectural additions and all major architectural re-engineering efforts created documents that were kept limited in scope and detail. That they existed at all was new. Their goal was to inform other engineers of what they needed to know, not to sell to mgt or marketing. White board discussions drove the designs. A team might take over a whiteboard for days or weeks so that they didn't have to translate design discussions into design documents to preserve their thoughts while they were working them out.

Beta releases improved in quality and scope. With *Betas*, Boxcar Development showed that it was truly a "ready-to-roll" process. *Betas* needed to be free of any data-damaging bugs or any bugs that interfered with existing functionality, because our *Beta* customers could not afford to maintain two sets of books, and we needed them to use the software daily in their businesses.

Boxcar Development created higher quality *Beta* releases while reducing our *Beta* regression test period from 2 weeks in prior releases to 1 day. We saw this in our own statics as well as heard it in direct feedback from *Beta* customers who started saying that the *Beta*'s were "as solid as the final product."

The Boxcar method also allowed us to get *Beta* software to our customers sooner in the release cycle, thus allowing for the longer beta cycle requirements of our payroll services. It also allowed us to put out more *Beta* waves with significant new features while maintaining our "near production quality" requirements. This was done with very little disruption to the project flow.

Improved understanding of release goals. Boxcar Development required more upfront planning on the part of Marketing with respect to marketing message and prioritization of features. This was seen and experienced as a good thing.

11.0 A story

Seven months into our first release cycle using Boxcar Development, it proved its value. A senior IC, who had been with Intuit about a year, was attempting to significantly extend some old code's functionality in order to build a new feature. He had originally advocated a re-architecture of the older functionality, but the design team had favored the extension. So, he extended the code per the team's design. The feature's boxcars were to be amongst the last loaded for the first *Beta* wave.

The boxcar failed to couple due to the deeper integration testing that process required. Each attempted fix brought different problems. A group of senior ICs were brought together to consult. The boxcar was decoupled and the *Beta* wave was shipped. The feature was targeted for the next *Beta* wave 3 weeks later. *The* design was modified in a second attempt at extending the old code. This also failed.

The immediate and intense focus that this feature got from the senior ICs and managers enabled them to much more readily accept that the old code really did need to be re-engineered. The decision to bypass the old code and create a new architecture based on the earlier design proposal was quickly approved. The new feature used the new architecture and made it into the 2nd *Beta* wave.

From the first coupling failure to the successful coupling on the new architecture's set of boxcars took just over 2 weeks. Admittedly the developer's heroic efforts made this possible. Prior to Boxcar Development, however, the problem would not have been seen as early, nor would so far reaching an outcome have been concluded. The old architecture was able to be scheduled to be replaced in the rest of the product during the next release cycle. The speed with which we were able to make such major and disruptive decisions, made the team believers in the Boxcar Development process.

12. To use or not to use...?

Ultimately, the purpose of sharing "Ready-to-Roll" Boxcar Development is so that others can gain from our experience and consider adopting or adapting part or all of the methods within their organizations. So far, I have experienced Boxcar Development used within two organizations: Intuit's Business Products Division; and a startup web-based payroll service. Both I think used it with good success. That said, it is not a panacea. There are conditions under which I can imagine it not being successful. Hopefully the questions below will help you in your considerations.

12.1 When to consider not using:

If there is a "cost" to Boxcar Development it comes from the time and effort spent to build quality in at the beginning. This cost may be more theoretical than real, as we seemed to make up for it with shorter *Beta* prep and Release Shutdown periods, and gained the ability to release features later in the cycle. In theory, however, where the ship date is rigid, this cost expresses itself in the reduction of the number of new features that will be attempted in the release cycle. Conversely, where the feature set requirements are rigid, this cost expresses itself in the reduction of your ability to firmly fix your ship date. Thus the usage question should depend on the

relative weight of product quality to new features and ship date rigidity.

In light of the possible gap between theory and reality, there are only two conditions under which I would strongly recommend **not** considering the use of the Boxcar Development method:

- *Assuming a rigid ship date*, if the total time estimates for your list of *must-have* features exceeds 50% of your development capacity, i.e. available resource time.
- If your management team is inexperienced with and/or resistant to fluid, flexible communication and decision-making methods. Your management team's ability to learn and manage a flexible, just-in-time decision process is critical to the method's success. Many decisions do get made upfront, but Boxcar Development allows them to easily be made differently at the point just before real resources are committed.

12.2 When to consider using:

Presumably, you have already read the benefits, so here are some questions to consider.

How important is your product quality:

- Is it mission critical?
- Are avoidable Tech Support costs eating up your profitability?
- Does your product's quality give you a measurable competitive advantage?
- If your ship date is inflexible, is quality more important than new feature content?
- If your feature requirements are inflexible, is quality more important than your ship date?
- If you are working on an old code base with a large customer base with rapid adoption practices, are mistakes very expensive?
- If you are developing a web-service that you frequently update with new features, is it very important to protect your customers from software "released before its time", i.e. buggy or poorly designed updates?

Is your rapidly growing organization struggling:

- To retain its small team feel and best practices?
- With training cycles that approach or exceed your development cycle?

Is your large organization struggling with:

- Management bottlenecks and disenfranchised individual contributors?
- Making its targets for date, quality or content?

Is your organization seeking more flexible project management methods?

If you answered yes to any of the above, then Boxcar Development **is** worth your consideration.