

Shinkhansen Proposal: Adaptive Development Methodology

Author: Chris Fry
Last Updated: 9/13/2006 2:30:13 PM
Feedback: Salesforce R&D Team

Proposal

1.1 Overview

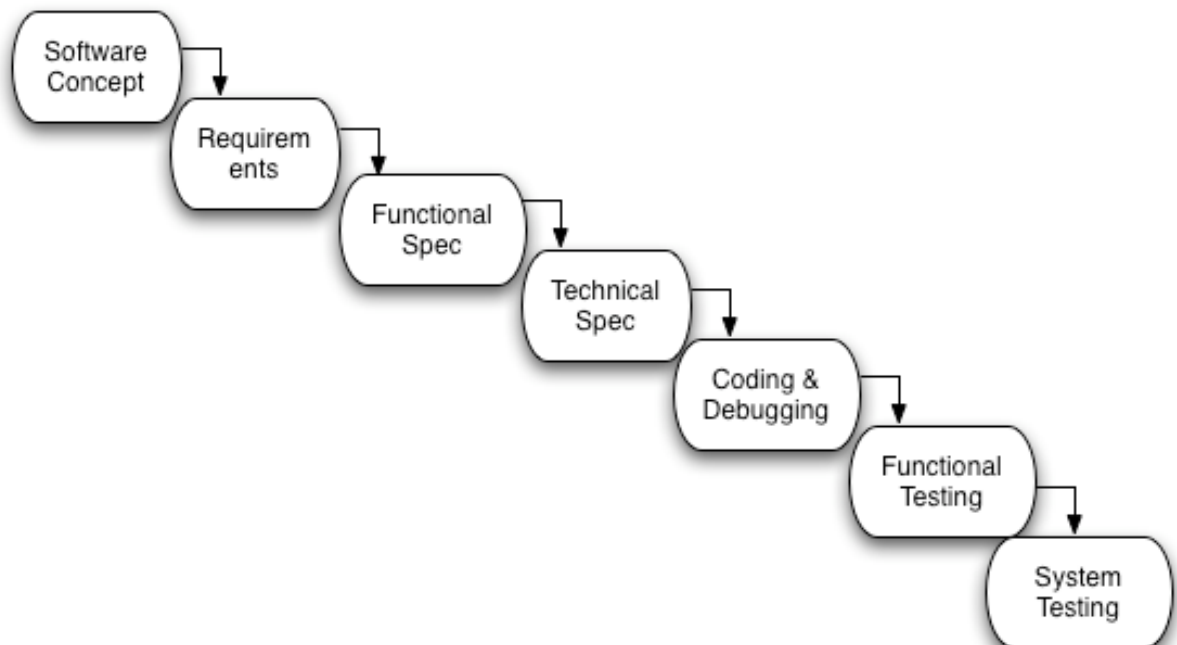
This document describes an iterated prototyping approach to developing software. It borrows from adaptive development techniques (Scrum, XP) but is intended to be a practical / easy modification to the current software development life cycle at Salesforce.com. It is not intended to be a wholesale adoption of XP or Scrum but a tailoring of the best of these methodologies to Salesforce. The goal is to introduce a small change to the development process that will reduce risk, increase predictability, allow for more accurate information flow, and result in more efficient software development.

This document proposes changes to our development methodology that should result in higher quality code, greater information flow, earlier feedback on features, better designed features and more predictable development cycles. Just as we are dividing up long releases into multiple shorter releases to get code into customer's hands quicker, this methodology divides a single release into multiple iterations, which creates predictable milestones for design and quality feedback.

This document does not deal with capacity planning however the architecture team will only be able to accept a certain number of disruptive changes per release.

1.2 Assumptions

This section outlines the assumptions that govern our current life cycle process. Our current model is a basic waterfall approach with variability across teams on executing on features.



The waterfall model is document driven and is designed to require up-front planning. It works best on well-defined maintenance of an existing product.

We are currently suffering from problems in certain areas:

- Our early estimates are not accurate resulting in missed feature complete dates and compressed testing schedules.
- There is little visibility early in the project on features that are missing milestones.
- We often don't complete one waterfall stage in the current model before moving on to the next (specs never close)
- Late feedback on features at the end of our release cycle.

The proposed model plans to deal with this by dividing features up into n iterations, estimating each iteration, and performing design / code / test at each iteration. We will schedule time up front for overall design and time at the end for UI automated testing and system testing.

Items **not** adopted from XP/Scrum

- Pair programming
- Total code ownership
- Feature stories

1.3 Detail

This section covers certain details of our current model that are best practices and should be maintained across process changes.

1.3.1 CURRENT BEST PRACTICES

This section describes adaptive techniques already in use at Salesforce.com some of the descriptions borrow heavily from [\[3\]](#)

CODE REVIEWS

Each change is reviewed by a dev reviewer for code quality. Code reviews are our basic mechanism for validating the design and implementation of code changes. It helps maintain a level of consistency in design and implementation practices across the code base. A review is focused on a changes design, implementation, usefulness in fixing a stated problem, and fit within its module. A reviewer should be someone with domain expertise in the problem area.

CONTINUOUS INTEGRATION (Integrate Often)

Developers should be integrating and releasing code into the code repository every few hours, when ever possible. Continuous integration often avoids diverging or fragmented development efforts, where developers are not communicating with each other about what can be re-used, or what could be shared. Everyone needs to work with the latest version. Unit tests run at 100% pass throughout the project.

Always work in the context of the latest version of the system. Continuous integration avoids or detects compatibility problems early. Integration is a "pay me now or pay me more later" activity. If you integrate through out the project in small amounts you will not find your self trying to integrate the system for weeks at the project's end while the deadline slips by.

CONTINUOUS BUILD AND TEST (Immediate Feedback)

Each change triggers a build, changes are batched and tested on an hourly and nightly basis. These builds are available throughout the organization giving total visibility into the code base and its functionality to designers, PMs, QA and Dev.

Unit Tests

Unit tests are one of the corner stones of development. Unit tests are released into the code repository along with the code they test. Code without tests may not be released. If a unit test is discovered to be missing it must be created at that time.

Unit tests enable collective code ownership. When you create unit tests you guard your functionality from being accidentally harmed. Requiring all code to pass all unit tests before it can be released ensures all functionality always works.

Regressions

When a bug is found tests are created to guard against it coming back. A bug in production requires a functional test be written to guard against it.

Functional Tests

Iteration goals are translated into functional acceptance tests. Functional tests are black box system tests. Each functional test represents some expected result from the system. Functional tests are also used as regression tests prior to a production release. An iteration goal is not complete until it has passed its functional tests. This means that new functional tests must be created each iteration or the development team will report zero progress. Quality assurance (QA) is an essential part of the process. Functional tests are automated so they can be run often. The functional test score is published to the team. The team must schedule time to fix failed tests at each iteration.

DEPLOY ANY TIME

Unit Tests and Functional Tests are always passing allowing us to deploy at any time. Code is checked in with tests allowing complete visibility into functionality.

DESIGN OWNERSHIP

Designers own the successful design of their features.

TRANSPARENCY

Code is available, feature status is communicated, design documents are centralized and available. Feature owners are able to observe the feature in its current state as soon as it is available.

1.4 New Process

This section describes the proposed process change. The proposed model is iterated prototyping combined with scrum style project management. We will define 30 day iterations that result in functioning code at the end of each iteration. Iterated prototyping will give us a regular iteration schedule across the product, the scrum techniques give us a way to plan work, calculate velocity and evaluate progress in stages.

Figure a) shows a simplified view of our current process.

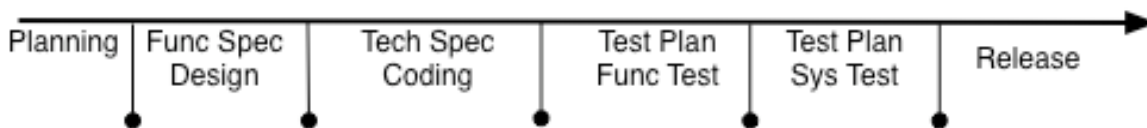


Figure a) A simplified view of the current software life cycle

Figure b) shows the modified proposal.

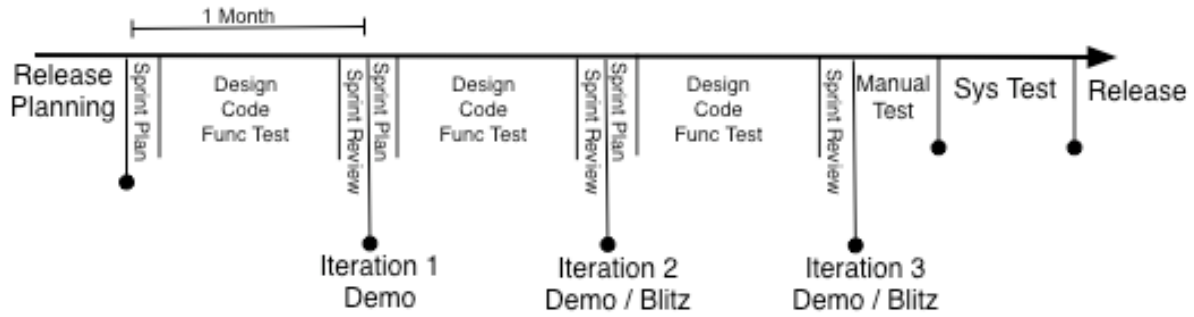


Figure b) A simplified view of the software life cycle

Figure c) shows the suggested release timeline:

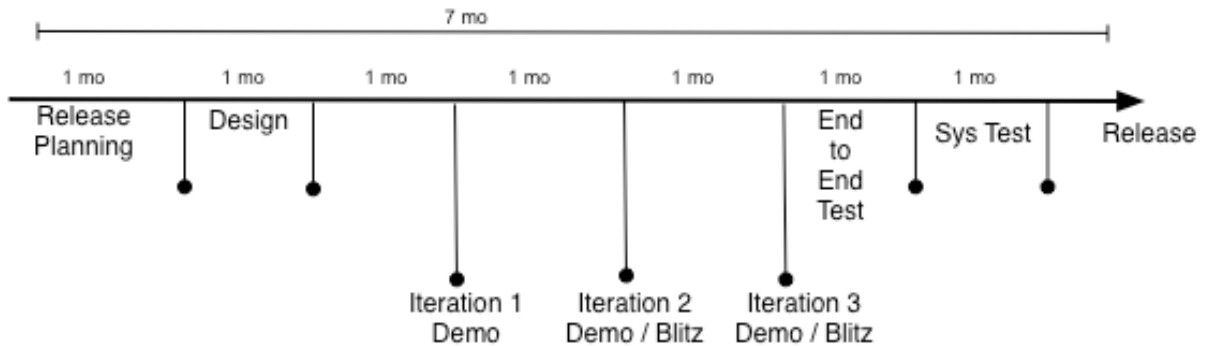


Figure c) Proposed release timeline

Figure d) shows a way to implement usability testing and design (adapted from [5])

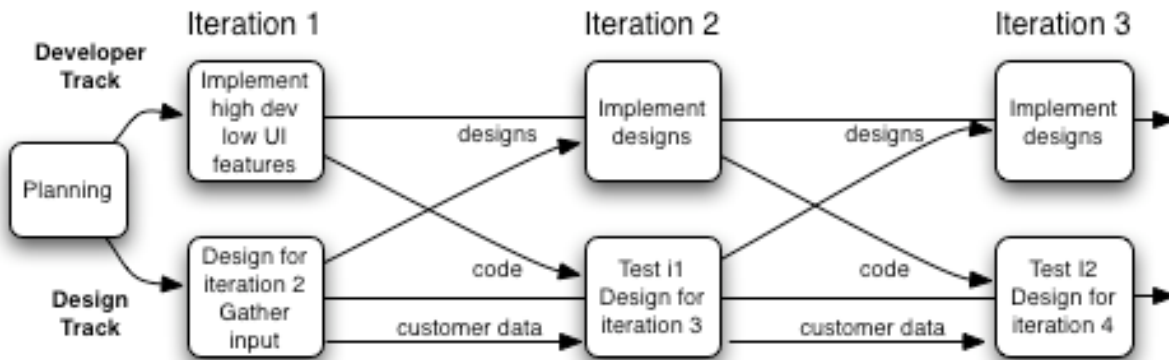


Figure d) Parallel design tracks

Release Planning

Initial features are targeted for research, this list should be broad and results in a candidate list. This list is winnowed and prioritized by product management / development and qa. The end result is a final candidate list for rapid design. N features are estimated and designed in the initial design phase resulting in the final features that enter the development phase.

Initial Design

An initial design team is formed (1 product manager, 1 senior developer, 1 optional UI designer) that iterates daily and creates:

- A working prototype (if necessary)
- A candidate functional spec
- A technical work breakdown and estimates
- A list of requirements

Miller [5] suggests creating a parallel design track that continues throughout the iterations and focuses usability engineers on a smaller set of features. Figure d, illustrates this model. It provides a way to get design feedback injected into iterations on a regular schedule and not at the end of the process. In the first iteration development focuses on development centric tasks and builds testing infrastructure. During this time the design team works on designs for the next iteration. Usability tests are presented on a regular schedule to the team and inform the next phase of development.

We would also like to create design teams that work through designs decoupled from a specific release that work through design issues and pull items that need more clarity off the product backlog.

Development

In this proposal we will divide each release into N iterations. The duration of 4 weeks has been selected, however the iteration length can be designed to fit the length of the release schedule. Each iteration has the defined goal of creating running, deployable code. Each iteration consists of a design phase, coding phase and testing phase. The first iteration of the code design should cover major cross system touchpoints. The code produced at the end of each iteration will be deployed and reviewed for design issues & quality issues. Tests are checked in while the code is being written. This gives greater visibility earlier in the cycle for features that are not meeting the iteration goals. One technique to making the first iteration productive is to schedule code infrastructure tasks for the first iteration to allow design to continue (see [5], section 8.1).

Iteration Planning

An iteration planning meeting is called at the beginning of each iteration to produce the plan of tasks for that iteration. Part of the plan should include fixing failed functional tests. Developers sign up for iteration tasks and estimate how long their tasks will take to complete. The person who is doing the task is responsible for the estimate. The project velocity from the last iteration should be used to scope the velocity of the next iteration. If the iteration is under scoped more tasks can be added. At the end of the iteration the total effort for tasks is reported, summed, and used to measure the team velocity.

After the iteration is planned no changes are allowed to the iteration. You cannot change the resources or the priorities of the team.

Each iteration should have a demo session for key stake holders. This creates natural, internal pressure to complete iteration goals. It also provides a point for *cross team* visibility into feature design. QA test planning, automation and test execution all happens within the iteration. QA and dev work as one unit. QA is working closely with development and executing on feature testing during the iteration. Tests are automated and kept at 100% passing to guard against regression.

Characteristics of scrum

This section describes the characteristics of scrum style project management. We can adopt some / all / none of these. The backlog and planning sessions are key to a successful application of this technique.

- A living backlog of prioritized work to be done (this should be rank ordered);
- Completion of a largely fixed set of backlog items in a series of short iterations or iterations;
- A brief daily meeting or scrum, at which progress is explained, upcoming work is described and impediments are raised (we should pick either feature team meetings or daily meetings)
- A brief planning session in which the backlog items for the iteration will be defined.
- A brief heartbeat retrospective, at which all team members reflect about the past iteration.

Application of iterated prototyping and scrum to Salesforce.com

Each release is broken up into N smaller iterations. Features are designed to be completed in N iterations. Work is planned so work items line up across the product for all feature areas. At each iteration the design for that iteration is completed, the feature is coded and the tests are written and checked in for that iteration. This model has the

following desirable features: it gives visibility into features early and provides early feedback on features that are missing milestones. The tasks that are being estimated are smaller and more predictable, leading to more accurate estimates. Feedback on estimates comes within an iteration of the estimates being given. Each iteration results in running code that can be evaluated for design, usability & quality. The iteration results in shipping quality code, not prototype code.

The basic plans are updated at each iteration (func spec / tech spec / test plan). Automated tests are designed, implemented and checked into the automated test suite. Regressions are prevented by keeping the tests at a 100% pass rate. This reduces the overall test debt and keeps work from piling up until the end of the project. Development receives feedback earlier and bugs are filed to track issues. We need to schedule time for developers to fix bugs and allow for regular code maintenance.

The basic outline is as follows:

1. Features are conceived and assigned to PM / dev / QA
2. General requirements are established
3. Overall architecture and scope of the feature are estimated
4. The goals for the iteration are agreed to by PM/dev/QA
5. The iteration is planned
6. PM does an initial design for the iteration and updates the func spec
7. Dev does mini goals for the iteration and estimates the time to achieve them and writes a tech spec for the iteration. Each one of these work items is tracked, with time to complete. If the item is not completed it is added to the backlog.
8. Tests are checked in with the code
9. The technical spec, functional spec, and test plan are updated
10. Software is presented to stakeholders (Blitz can be scheduled here), UI feedback is collected and fed back into the process by the designer and QA engineer
- 10.a. Iteration review (total velocity per iteration is tallied)
11. Goto step 4 until N iterations are complete
12. Complete UI automated testing, bug fixing
13. System Testing
14. Deploy

Branching

This development model is specifically designed to work with a continuous integration, build and test model of software development. This implies that code should and will be checked into a single shared branch. Keeping automated tests passing at all times and keeping code coverage at targeted goals prevents regressions.

Feature branches can be used to defray risk for certain large projects, but they are not required or desired for this process.

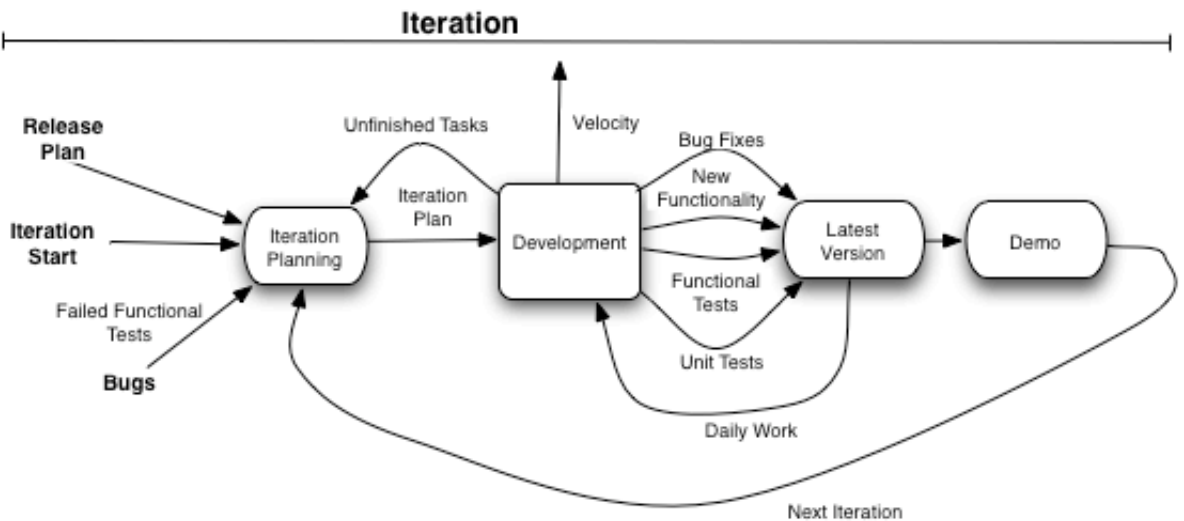
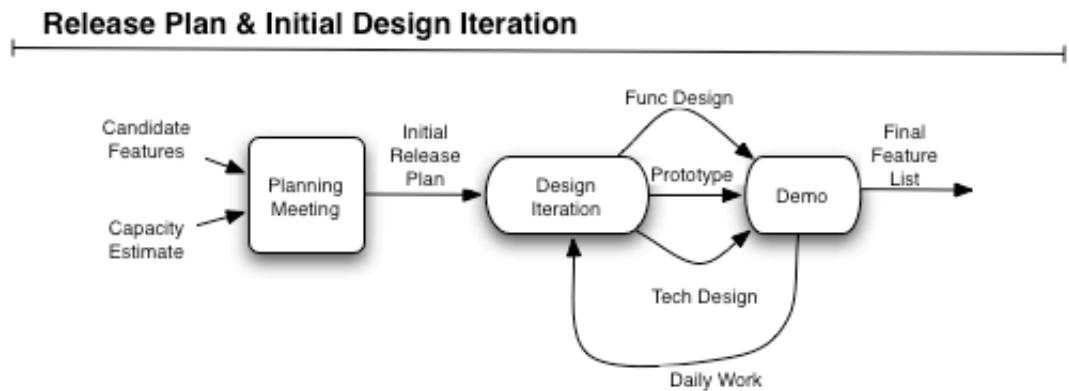
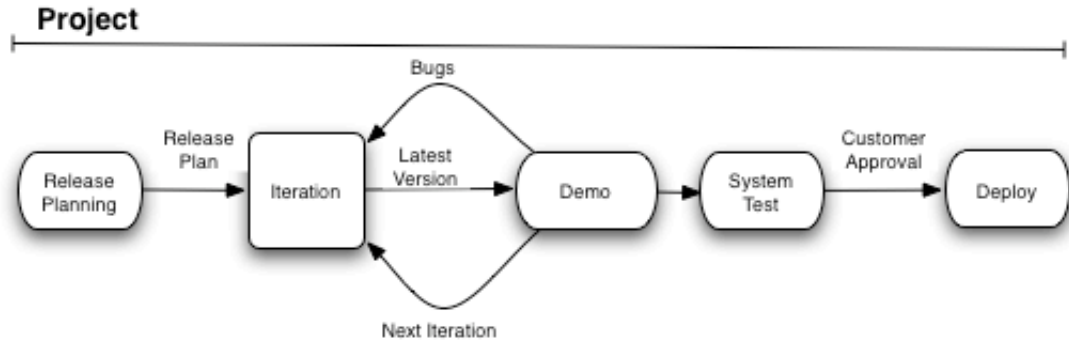
Data Model

We will have a data base per release with a forward rolling data model. The development DBA and the software engineer manage breaking data model changes. Each release will have its own upgrade script. The architecture team has the right to schedule features for a later release based on overall impact. No one release should perturb the data model too much (too many architectural changes). PM must work with the architecture team early in the release planning process, with the possible outcome of pushing features to later releases based on early estimates of risk, overall impact and capacity.

Downtime

The 7 month cycle will create some slack for developers during month 1 and parts of 2 and 7. This time can be used for production bug fixing, but should also be left open for developer driven refactoring and infrastructure projects. The changes could be on the next release line and must be done before the iterations on new features start in month 3.

1.4.1 Flow Chart



1.4.2 Tools/Reporting/Visibility

Project Velocity is the number of tasks completed per iteration. The backlog (tasks targeted for an iteration but not completed) gives visibility into features that are not on track. The explicit goal of having running, production quality code gives a tangible way to observe the progress of a feature.

New Metrics:

- Iteration tasks completed (we could also use total duration of tasks completed)
- % of iteration tasks complete
- Stretch goals/tasks complete

1.5 Obstacles and Issues

- [Large Features] Not every feature can be broken into smaller segments. The goal is to divide delivery of every feature into 3 predictable increments. Large architectural features may not be amenable to this strategy. **Mitigation:** Accept only a few large features per release, use the Velocity tracking measure proposed in this document, front load testing. Another way to mitigate risk would be only to accept pieces of a large change in steps.
- [Small Requests] Smaller marketing driven features arrive late and must be addressed. **Mitigation:** This model provides a way to schedule small requests into iterations.
- [Cross release roadmap] We need a high level multi-release plan and goals for the year.
- [Adoption] Overall adoption and rollout to the organization. Developer adoption of functional testing and unit testing as a priority.
- [Tracking] This model only provides visibility if iteration goals are tracked and reported on. If they are not reported on we will not gain visibility. **Mitigation:** make iteration goal tracking and reporting part of the TPM standard report.
- [Resource Scheduling] Other priorities drag resources away from iteration goals
- [Feature Creep] Overall development cycle does not complete a feature (3 iterations do not result in a complete feature)
- [Feature Creep] There's a risk with a 1 month sprint that is really part of a 7 month release, that sprints are allowed to bleed into each other (the features aren't complete and shippable at the end of sprint).
- [Cutting Features] If we have visibility into a feature that is missing it's goals, will we have the organizational commitment to cut that feature
- [Marquee Features] Should we have marquis features that would result in an added iteration if they are not complete by the third iteration (must have marketing features for the release that would cause the schedule to push)
- [Resource Scheduling] How do we schedule overbooked resources in this model (for example designers often stage work to work on multiple features, if they are required to provide feedback on each feature at a specific date they may be unable to satisfy the demands on their time)
- [Resource Scheduling] With shorter iterations, it may be difficult to swap a resources on a project, or to pull them off temporarily. This could be a positive benefit as it would force managers to dedicate resources to a project, but could be a drawback too if unexpected issues arise that require pulling someone off.
- [Production Quality Code] Production quality code must be checked in at all time, no prototype level code.
- [Automation] UI and Functional automation must be in place to support this model
- [Automation/Regressions] UI can regress during development
- [Manage Expectations] PMs are going to be seeing 1/3 of the work at the beginning; they need to understand that the feature is not completed.
- [Usability Testing] Where and when do we do it?
- [UI Testing] It seems we may have higher net testing time of the UI due to the retesting that may be needed of the UI each iteration. This depends on how much UI testing is done in the iterations.
- [Design Discipline] With the iterative approach, PMs are not pressured to think through the entire functional design early in the release cycle. While some would argue that this is a good thing, there are also potential downsides. If the PM is not thinking through the entire design early, we could be discovering functional gaps/issues later in the release.

1.6 FAQ

Who is the scrum master? The scrum master is someone with leadership qualities that we feel can handle the project manager capabilities of the role (this is probably a dev manager / tpm / lead)

How many scrums will we have? We want to scale the organization, while we don't want to limit our flexibility the max size for a scrum team is 8-10, and the minimum size is probably 1-3. No individual contributor should be required to attend more than one scrum and no scrum master should directly lead more than one scrum meeting. Who leads the scrum of scrums and who is the overall product owner? Scrum of scrums for each business unit, scrum master for the scrum of scrums is the release manager. Who is the overall product owner (don't know for this one)

If it's not on the list it doesn't exist (no new work during the scrum, don't devolve into chaos).

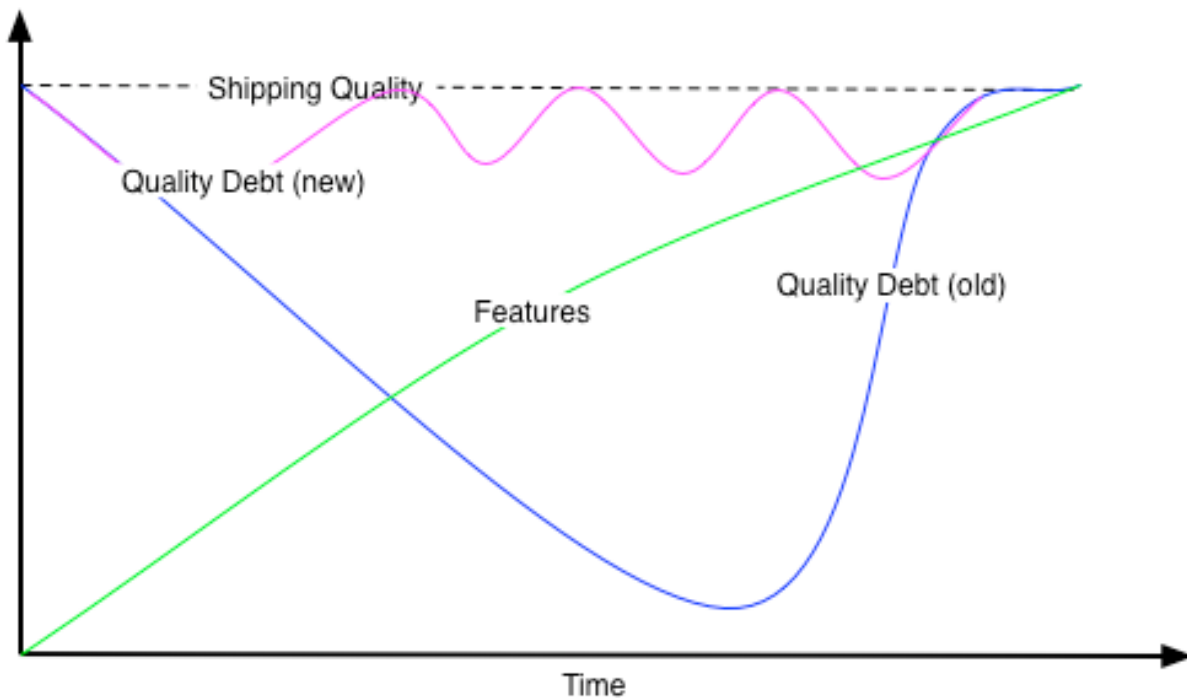
1.7 Summary

This development methodology is designed to give us visibility, predictability and higher quality with better estimation, tracking and testing. It outlines an iterative based approach that should be practical to adopt.

1.8 Rollout

- Piloted in 144 ML project
- Pilot in 146
- GA in 148

This model predicts that our quality debt will be addressed in each iteration, rather than being built up towards the end of the release. The following diagram shows this relationship.



1.9 References

- [1] Rapid Development, Steve McConnell, 1996
- [2] Scrum development [URL](#)
- [3] eXtreme Programming [XP](#)
- [4] [Eclipse Culture](#)
- [5] [Lynn Miller's Report on UE and Agile methods](#)
- [6] Refactoring, Martin Fowler, 2000

[7] Agile Software Development, Robert Martin, 2002

[8] Agile Development with Scrum, Schwaber and Beedle, 2001