Agile 101

- Clear, customer visible stories
- Test-Driven Development
- Continuous Integration
- Short Iterations
- Pair Programming
- Extensive Customer Involvement
Customer-Visible Stories

- The focus on customer-visible stories keeps development concrete, and fights over-architecture.

- It reminds us why we’re doing the work in the first place.

- It forces us to justify our design choices, grounding them in the real customer need.
Test-Driven Development

- You write the tests before you write the code
- Separates requirements from implementation
- Produces an executable specification, and documentation that stays in sync with the code
- No code without a test
- 101% test coverage (well, at least at the start.)
Continuous Integration

- Everyone runs the test suite before submitting.
- The continuous build checks out the trunk, builds it, and runs all tests.
- Problems are caught early, when they’re easy to fix.
- The entire application is kept in a deployable state from the first week.
Short Iterations

- One week iterations make for easier course corrections, and shorten the feedback cycle.

- Requirements change as the customer has a chance to validate the design through play testing.

- Complete test coverage and alignment between customer and developer make course corrections painless instead of arduous, cheap instead of expensive.
Pair Programming

- Probably the most controversial part of XP

- How can it be faster for two developers to work on the same problem? Surely it’s faster if they work on two separate problems in parallel...

- Yes, we actually work this way, with two developers working on a single machine.
We do this naturally

- Developers will instinctively ‘pair program’ when one introduces another to a new code base.

- Developers will instinctively work together to solve hard design problems.

  *But...*

- Developers don’t always want to be so closely scrutinized.
How Pairing Works

- Pairing accelerates knowledge transfer
- Pairing makes deep problems shallow
  - Your 80/20 rule overlaps favorably with your partner’s 80/20 rule
- Pairing keeps you focused
- Pairing keeps you honest
How do you write code?

- Ask yourself what percentage of your development time you spend...
  - ...stuck on some difficult problem
  - ...stuck on some trivial problem
  - ...trying to choose between two implementation choices
  - ...reading email or news or blog posts
Customer Involvement

- The customer owns the priorities, the developer owns the cost estimates

- A feature isn’t done until it’s deployed to the demo server and approved by the customer

*The results:*

- Features are really done when they’re marked done

- The customer and developer are aligned in reaching their common goal
Tracking Progress

- Estimate Complexity, not Duration
- Constant Feedback improves accuracy
- Applies to the iteration level
- Equally applies to project-level estimation and resource planning
The Planning Game and the Point System

- Stories estimated at weekly Planning Game
- Stories are broken down into units of 1, 2 or 3 points
- Points measure complexity, not duration
- Larger stories are broken down into smaller stories that are 3 points’ worth or smaller
- The customer (usually PM) prioritizes the work, informed by the complexity estimates
Task Estimating

- People are better at estimating complexity than duration
- Tasks have fractal complexity: Small tasks are more predictable than large ones
- Exposing the cost of features and giving control to the customer creates alignment between the developer and the customer
The Point System

• One point: I know exactly how to do this, and can do it in half a day.

• Two points: I know exactly how to do this, but it will be some work.

• Three points: “Somehow we will implement this feature.”

• Three points almost always turns into more, and is a big red flag that the story needs to be broken down into smaller stories.
Velocity Tracking

- A focused team gets about as much done each week as it did the week before.

- The number of points completed in one week is an excellent predictor of the number of points completed in the next.

- Predictable results build trust between developer and customer.
Agile Works, But does it Scale?

- “Agile is great for small projects, you can’t make it work in a big company.”
- “Agile is great for exploratory projects, where you don’t know the requirements up-front, but not for projects with big up-front requirements.”
- “Agile works for greenfield projects, but not for big legacy applications.”
Three Examples

- Large Legacy Application
- Parallel Development via the Repository Pattern
- Greenfield Project
Legacy Application

- Large existing web application
- Functional/characterization tests
- Refactoring for testability
- Replacing a hard-to-test MVC model with a test-friendly one (WebWork)
- Test-driving new code
Parallel Development

- Large web application
- Distributed persistence layer wasn’t ready at the time UI development needed to begin
- Working to a mock repository enabled UI-layer development to proceed in parallel
- Shortened time-to-market by months on a 6-month release cycle
Greenfield Application

- Large new internal web application suite
- Select technologies for performance and testability (WebWork, Hibernate)
- Test-drive all code
Test running times for one subproject

- 375 common tests (19 seconds)
- 611 unit tests (5 seconds)
- 300 HTML tests (378 seconds)
- 56 integration tests (32 seconds)
- 12 scenario tests (22 seconds)
- 1428 total tests (456 seconds, ≈ 7½ minutes)
Code Quality and TDD

- In one application built on the platform, one subsystem was not test-driven, and not developed using agile practices.

- One non-Agile, non-TDD subsystem comprised <10% of the code, but was the source of ~75% of the bugs.

- The non-Agile subsystem missed its release dates, and pushed the release date of the whole application, by 4 weeks. (From a 12 week planned release cycle.)
Agile v. Up-Front Design

Plans are nothing; planning is everything.

–Dwight D. Eisenhower
Agile v. Up-Front Design

- Google culture is oriented toward design documents and PRDs
- Agile approaches favor ‘just enough’ direction
- Designs and requirements change over the course of a project
- Agile approaches guide that change to a productive outcome
PRDs

- PRDs capture business requirements

- Requirements tend to change over time, but that change isn’t typically captured in the PRD

- Direct involvement of the PM in the customer role keeps the real business requirements in focus

- Direct involvement of QA lets QA verify the most current requirements
Specifications

- The test is an executable specification
  - It stays in sync with the code
  - It proves that the code does what the test describes
  - Tests provide a catalog of correct usage
  - Well-written tests elucidate intent
Specifications

- The test is not an executable specification
- Tests don’t do a good job capturing an overview of the high-level requirements
- Tests don’t readily expose relative importance of requirements
- Tests provide a catalog of an arbitrary set of usage examples
- Poorly written tests don’t elucidate intent
Design Documents

- Provide a mechanism for peer review
- Provide a mechanism for communicating the design
- Provide a mechanism for thinking through the design before committing anything to code
- Development tends to reveal deficiencies in the design, which changes as a result
- Can be updated once the implementation is done

Copyright © 2006 Google, Inc.
Code Reviews

- Code Reviews provide a second pair of eyes on every line of code that gets checked in.
- Every change list must be reviewed before it can be submitted.
- Pairing acts as a Continuous Code Review.
- Change lists are copied to the project dev list for additional ad hoc review by interested parties.
Estimating Schedule Up Front

- Useful for resource planning
- Useful for planning external dependencies
- Marketing
- Partner timing (e.g. for integrations)
- Hard to get right
Managing to Schedule

- Estimate, then re-estimate
  - in two weeks
  - as new information becomes available
- Decompose stories until the scope becomes evident
The Release Plan

- The release plan contains the backlog
- The iteration plan pulls stories from it
The Release Plan

- Development Points v. Release Points
  - Development points track iteration velocity
  - Release points track completion of stories from the release plan
  - Development velocity tends to track release velocity
  - Disconnects between these velocities give critical feedback early
How do you scale safely without agile?

- Take a thousand brilliant engineers, put them in a room, and let them figure it out.

- This works great, to a point. Further than you might think, but there are limits.

- We do Agile at Google in response to a real need, to provide real results, not because someone thought it would be a cool thing to do.
Why is this so important?

• As code ages, the cost of change goes up

• As Google grows, the percentage of code in maintenance mode grows with it

• We need to keep code flexible, so we can change it to suit new market conditions quickly

• It’s not important if all you’re trying to do is get new products out. (Smart programmers can develop very good applications without tests, but those applications will be much harder to maintain.)
Layering provides Developer Scalability

- Test-Driven Designs encourage isolation of concerns
- Isolated concerns allow one large team to turn into many small teams
- Test-Driven (and test-covered) internal APIs
The Google Agile Development Process

- Varies by team, and adapts to local conditions
- Different projects adopt different subsets of the techniques, with varying levels of rigor
- Lets QA spend more time on edge cases instead of just verifying basic behavior
The Google Agile Development Process

- Continuous Build
- Green smoke tests required to check in
  - Where practical, run all tests before check-in
- No (new) code without a test
- Pair program and/or code review before check-in
The Google Agile Development Process

- Estimate schedule and resources at the PRD level. Re-estimate as details become clear.
- Decompose stories until they are all smaller than 3 points
- Track developer velocity in short iterations
- Track project velocity based on feature points
Continuous feedback is good!

Google is actively recruiting agilists

You can ask me more: imf@google.com