Note 1: Agile Development

Yan Shi
Lecture Notes for SE 3330
UW-Platteville
Issues With Traditional Processes

- How long do you need to finish this?
  - Estimation based on no data
  - unanswerable question

- Analysis → Design → Implementation
  - upfront analysis and design takes too much time
  - not enough time for implementation

- “I want this additional feature added!”
  - no time for it
  - ruins the “perfect” design
What is Agility?

- Agile development: respond quickly to changing needs
  - Business models won’t wait 6 months for implementation
  - Market conditions change rapidly, must respond quickly
Agile Manifesto (2001)

- Official Website: [http://agilemanifesto.org/](http://agilemanifesto.org/)
- **Individuals and interactions** over **processes and tools**
- **Working software** over **comprehensive documentation**
- **Customer collaboration** over **contract negotiation**
- **Responding to change** over **following a plan**
12 Principles of Agile Development

1. **Highest priority**: *satisfy customer* through early, continuous delivery of valuable software

2. **Welcome changing requirements**
   – even late in project!

3. **Deliver working software frequently**
   2-8 wks, 2 is preferable

4. **Customer and developers**: *work together daily*
5. **Build projects around motivated individuals:**
   - give them environment & support;
   - trust them to get job done

6. **Emphasis on face-to-face communication**

7. **Working software:** primary measure of progress

8. **Sustainable development**
   establish a constant pace that can be maintained indefinitely
9. Technical excellence and good design:
   - enhances agility

10. Simplicity is essential:
    - maximize the amount of work not done

11. Self-organizing teams produce the best architectures/requirements/design

12. Team reflects on how to become more effective
    - Done at regular intervals, adjust as needed.
Kent Beck

- Currently work for Facebook
- Known for:
  - Extreme Programming
  - Design patterns
  - Junit
  - ...

[Image of Kent Beck]

Extreme Programming (XP)

- XP is a major agile development approach.
- The seminal work by Kent Beck was published in 1999.

- Framework activities:
  - Planning
  - Managing
  - Designing
  - Coding
  - Testing

- The rules of XP:
  [http://www.extremeprogramming.org/rules.html](http://www.extremeprogramming.org/rules.html)
XP Planning

- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.
XP Planning: Stories

http://www.youtube.com/watch?v=LGeDZmrWwsw&feature=player_embedded

• What is a user story?

• What should be included in a story card?

• Who are involved during developing user stories?
Create a set of stories, each story
- is one sentence
- describes required features and functionality
  - Essentially scenarios, though a little less refined
  - Focus on usage that provides a benefit to the user
  - “As a ... (user type), I need... (features/functionality) so that ... (goal)”
- is written by the customer
- is placed on an index card

Note:
- no if, no loop, single train of interaction
- can start with epic stories and then break them into small, ready stories.
- new stories can be written any time!
- acceptance criteria at the back of the story card!
Exercise: Restaurant Management System

- A local restaurant would like to use a software system to automate their daily work as much as possible.

- Write as many story cards as you can.
Acceptance Criteria

- 3C’s of a user story: card, conversation, **confirmation**
  - Customer writes acceptance criteria
  - developers writes acceptance tests

- Acceptance criteria
  - are written in simple language.
  - are used to confirm when a story is completed and working as intended.
  - Define boundaries and parameters of a user story
  - state intent, but not a solution.
  - should be testable.

- Questions to the product owner ➔ acceptance criteria
  - “As a conference attendee, I want to be able to register online, so I can register quickly and cut down on paperwork.”
Acceptance Criteria

- Functional Criteria
  - identify specific user tasks
  - “A user is able to access a list of available reports.”

- Non-Functional Criteria
  - identify specific non-functional conditions the implementation must meet.
  - “Edit buttons and Workflow buttons comply with the Site Button Design Standard.”

- Performance Criteria
  - include if specific performance is critical.
  - “Query response time should be less than 2 seconds.”
(for a HR system) As an administrator, I want to be able to create user accounts, so that I can grant users access to the system.

- An admin can create user accounts.
- An admin can create a user account by entering the following information about a user: name, email, phone number, user type(Power/Basic/None), account status(active/inactive), reports to(a list of active users)
- An admin cannot assign a new user to report to an “inactive” user
- An admin cannot assign a new user to report to a user if it creates a cyclical relationship
- The system notifies the admin that it sent an email to the new user’s email address, with initial password and login instructions.
- The new user receives an email with initial password and login instructions within 5 minutes after the account is created.

http://www.seguetech.com/what-characteristics-make-good-agile-acceptance-criteria/
Exercise

- write acceptance criteria at the back of two of your restaurant story cards.
XP Planning: Prioritization

http://www.youtube.com/watch?v=QfZo9cxnQgY&feature=player_embedded

• How to prioritize?
XP Planning: Release Planning

Stories are then used to organize the project:

- Each story is assigned a **value** (priority) by the customer.

- Each story is then assigned a **cost** (in terms of development weeks) by the XP teams
  - Ideal programming week: nothing else to do, no dependencies, no extra work, but do include tests
  - If the cost is more than 3 weeks, split the story
  - If the cost is very small, merge the story with another one

- Customer & XP team then group stories into **releases**:
  - highest value or highest risk stories usually come first
  - story driven (by scope) or date driven (by date)
The Planning Game

Based on the story cards you have:

- **Players**: Business and Development
- **Goal**: put the greatest possible value of stories into production over the life of the game.
- **Pieces**: User stories
- **Moves**:
  - **Write Story**: Business assign a value to a story (1-5).
  - **Estimate Story**: Development assign a cost to the story as X hours.
    - Business split/merge the story based on the estimation if necessary.
  - **Make Commitment**: Business and Development work together to plan releases
XP Planning: Changes

- What if the release is overbooked?
- What if some stories from the previous release were snow plowed?
- What if Business changes a value of a story?
- What if Business writes a new story?

→ Re-negotiate during iteration planning
XP Planning: Iteration Planning

- Take place at the beginning of each release.
- What we already have:
  - a set of user stories for this release (old and new)
  - failed acceptance tests to be fixed
- What we should do:
  - break down the stories and fixes into tasks.
  - developers sign up for tasks and then estimate how long it will take
  - use project velocity to check if the release is over booked or not.
    - It cannot exceed the project velocity from the previous release.
  - Adjust to whatever changes as needed
Project Velocity

- A measure of how much work is getting done on your project

- Simply add up the estimated cost of the stories finished during the iteration.
  - don’t divide by iteration length or #of people!

- Questions:
  - How to estimate velocity for the first iteration?
  - What if project teams change size?
  - What if iteration length changes?
  - What if velocity fluctuates?
  - Should velocity be accumulated across all agile teams or projects?
  - Does maximum velocity mean maximum productivity?
the **KISS** principle: Keep It Super Simple!

- Light documentation

**XP encourages:**

- the use of **CRC** (class-responsibility-collaborator) cards (we use class diagram instead in the course project)
- **spike** solution
- refactoring
CRC Card Review

- Class Responsibility Collaborator cards
- Types of Responsibilities
  - To do something
  - To provide information
- Use scenarios and role plays to figure out responsibilities and collaborators
- Example: [http://agilemodeling.com/artifacts/crcModel.htm](http://agilemodeling.com/artifacts/crcModel.htm)

<table>
<thead>
<tr>
<th>Class</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient</td>
<td>Name</td>
<td>Email</td>
</tr>
<tr>
<td></td>
<td>MedicalHistory</td>
<td>Appointment</td>
</tr>
</tbody>
</table>
### Spikes

- A spike is an operational prototype to
  - validate the original estimation
  - lower the risk of true implementation

- You can use a spike to
  - learn new techniques needed for the project
  - clarify requirements or UI expectations with customer
  - make better technical decisions
Refactoring

- Refactoring is the process of improving the internal structure without altering the external behavior of the code.

- Steps: Identify code smells ➔ refactor ➔ test!

- Smells to Refactoring Cheat Sheet

- Principles of refactoring:
  - Do not refactor and add functionality at the same time
  - Make sure you have good tests before you refactor and test after refactoring
  - Take small steps
XP Implementation: Test Driven Development

http://www.youtube.com/watch?v=uGaNkTahrlw&feature=player_embedded

• What is TDD?
• What are the benefits of TDD?
XP Implementation: Test Driven Development (Cont.)

- In each release, before coding, develop a unit test for each class/function
- Write just enough code to pass the unit test
- Refactor your code

Why?
- KISS: The developer is better focusing on what MUST be implemented to pass the unit test.
- The code is unit tested when completed.
- Provide documentation for different team members.
Wake’s Traffic Light Metaphor

The test doesn’t compile!

The test passes!  The test compiles, but the code fails.
Normal Cycle

1. **Start.** *(Green light!)*
2. **Write a test.**
3. Try to run the test.
   - *It fails to compile, because the code is not written yet.* *(Yellow light!)*
4. **Write a stub for the new routine.**
5. Recompile the test and the stub
   - *It now compiles*
6. Try to run the test.
   - *It fails, because the stub doesn't do anything yet.* *(Red light!)*
7. **Write the body of the stubbed routine.**
8. Recompile the test
9. Try to run the test.
   - *It passes.* *(Green light again!)*
   - Or it still needs work *(Red light!)* and the production code needs more work.
   - Eventually it will work and go to *(Green light)*
10. **Start the cycle again for the next routine.**
Wake’s Traffic Light Metaphor

- The test doesn’t compile!
  - classes and methods need to be implemented
  - check in the code and continue to next
  - work on the code
- The test passes!
  - The test compiles, but the code fails.
As a registrar staff, I need to know whether a student is minor or not

- A student younger than 18 year old is reported as minor
- A student 18 year or older is reported as adult.
Abnormal Conditions

- From **Green** to **Green**: The test passed right away.

- From **Green** to **Red**: The test failed right away.

- From **Yellow** to **Yellow**: still cannot compile.

- From **Yellow** to **Green**: The test didn't compile without the stub, but adding the stub let the test pass.

- From **Red** to **Yellow**: test failed ➞ cannot compile!

- From **Red** to **Red**: test still failed after adding new code...
Abnormal Conditions

- **From Green to Green**: The test passed right away.
  - Either the test is bad, or you've already implemented the feature being tested.
  - You might consider modifying the implementation just to see the test fail.

- **From Green to Red**: The test failed right away.
  - This is OK if it's a new test for an existing routine. ➔ rework the routine!

- **From Yellow to Yellow**: syntax error creating the stub.

- **From Yellow to Green**: The test didn't compile without the stub, but adding the stub let the test pass.
  - Suspicious: if a do-nothing stub makes the test work, is the test valid?

- **From Red to Yellow**: syntax error implementing the routine. This happens a lot!

- **From Red to Red**: New code didn't work.
  - This happens to everyone - just fix the code. **But** - if it happens a lot, it's telling you to move to smaller tests (so you'll add smaller bits of new code as well).
As a system engineer, I need a simple calculator, so that I can get the sum of a sequence of numbers.

Acceptance criteria:
- The calculator can take 0, 1 or 2 numbers.
- The calculator returns the sum of input numbers.
- For an empty input, the calculator returns 0.

Use TDD to implement this story. Use the traffic metaphor to document your steps.
XP Implementation: Pair Programming

http://www.youtube.com/watch?v=ET3Q6zNK3Io&feature=player_embedded

• What are the different roles in pair programming?
• What are the benefits?
• Why do we need to change partners?
XP Implementation: Pair Programming (Cont.)

- Two roles:
  - driver: write the code
  - copilot: watch, correct, suggest

- Other programmers are going to be intimately familiar with the way you code:
  - being comfortable is important!

- Pair with multiple partners
  - spread the ideas and techniques within the group
Benefit of Pair Programming:

- Real-time code reviews
  - typos, cognitive dropouts, and invalid assumptions
- Avoid distractions
- Managing of two
- Knowledge and information migration
XP Testing

- Unit test: TDD. done with implementation
- Integration test: on a daily/hourly basis
- Acceptance test: AKA customer test
  - derived from user stories implemented in a release
  - should be automated so they can be run often.
XP Management

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
  - What did I do yesterday?
  - What will I do today?
  - Any roadblocks
- The Project Velocity is measured.
- Move people around.
- Fix XP when it breaks.
  - Have retrospective meetings to talk about what is working and what is not and devise ways to improve XP.
Other Agile Development Processes

- Adaptive Software Development (ASD)
  - focus on human collaboration and team self-organization
- Dynamic Systems Development Method (DSDM)
  - incremental prototyping in a controlled project environment
- Scrum
  - emphasize the use of a set of “software process patterns”
- Crystal
  - put a premium on “maneuverability”
- Feature Driven Development (FDD)
  - can be applied to moderate or larger sized projects
- Agile Modeling (AM)
- Kanban
  - Visualize the workflow and focus on continuous improvement
Summary

- Agile manifesto
- 12 principles of agile software
- Extreme programming (XP)
  - user stories
  - release planning and iteration planning
  - design: CRC, spikes, refactoring
  - test driven development
  - pair programming