

Chapter 16, Methodologies **Examples: XP and Scrum**



Outline of the Lecture

- Examples of Methodologies
 - Extreme Programming
 - Scrum

XP (Extreme Programming)

- XP is an agile software methodology
 - Higher priority on *adaptability* (“empirical process control model”) than on *predictability* (“defined process control model”)
 - Change in the requirements is normal during software development
 - Software developer must be able react to changing requirements at any point during the project
 - XP prescribes a set of day-to-day practices for managers and developers to address this problem.

History of XP

- Original cast
 - Kent Beck, Ron Jeffries, Ward Cunningham (also created Wiki)
- Application of XP in the Chrysler Comprehensive Compensation project (C3 Project) in 1995
- Lots of initial excitement but later a lot of problems:
 - Daimler actually shut down the C3 Project in 2000 and even banned XP for some time
 - (See Additional References).

XP Day-to-Day Practices

1. Rapid feedback

- Confronting issues early results in more time for resolving issues. This applies both to client feedback and feedback from testing

2. Simplicity

- The design should focus on the current requirements
- Simple designs are easier to understand and change than complex ones

3. Incremental change

- One change at the time instead of many concurrent changes
- One change at the time should be integrated with the current baseline.

XP Mantras (continued)

4. Embracing change

- Change is inevitable and frequent in XP projects
- Change is normal and not an exception that needs to be avoided

5. Quality work

- Focus on rapid projects where progress is demonstrated frequently
- Each change should be implemented carefully and completely.

How much planning in XP?

- Planning is driven by requirements and their relative priorities
 - Requirements are elicited by writing stories with the client (called **user stories**)
- User stories are high-level scenarios or use cases that encompass a set of coherent features
 - Developers decompose each user story in terms of development tasks that are needed to realize the features required by the story
 - Developers estimate the duration of each task in terms of days
 - If a task is planned for more than a couple of weeks, it is further decomposed into smaller tasks.

How much planning in XP?

- **Ideal weeks**

- Number of weeks estimated by a developer to implement the story if all work time was dedicated for this single purpose

- **Fudge Factor**

- Factor to reflect overhead activities (meetings, holidays, sick days...)
 - Also takes into account uncertainties associated with planning

- **Project velocity**

- Inverse of ideal weeks
 - i.e., how many ideal weeks can be accomplished in fixed time.

How much planning in XP?

- **Stacks**
 - The user stories are organized into stacks of related functionality
- **Prioritization of stacks**
 - The client prioritizes the stacks so that essential requirements can be addressed early and optional requirements last
- **Release Plan**
 - Specifies which story will be implemented for which release and when it will be deployed to the end user
- **Schedule**
 - Releases are scheduled frequently (e.g., every 1–2 months) to ensure rapid feedback from the end users.

Team Organization in XP

- Production code is written in pairs (pair programming)
- Individual developers may write prototypes for experiments or proof of concepts, but not production code
- Moreover, pairs are rotated often to enable a better distribution of knowledge throughout the project.

How much reuse in XP?

- **Simple design**
 - Developers are encouraged to select the most simple solution that addresses the user story being currently implemented
- **No design reusability**
 - The software architecture can be refined and discovered one story at the time, as the prototype evolves towards the complete system
- **Focus on Refactoring**
 - Design patterns might be introduced as a result of refactoring, when changes are actually implemented
 - Reuse discovery only during implementation.

How much modeling in XP?

- **No explicit analysis/design models**
 - Minimize the amount of documentation
 - Fewer deliverables reduce the duplication of issues
- **Models are only communicated among participants**
 - The client is the “walking specification”
- **Source Code is the only external model**
 - The system design is made visible in the source code by using descriptive naming schemes
- Refactoring is used to improve the source code
 - Coding standards are used to help developers communicate using only the source code.

How much process in XP?

- Iterative life cycle model with 5 activities: planning, design, coding, testing and integration
 - Planning occurs at the beginning of each iteration
 - Design, coding, and testing are done incrementally
 - Source code is continuously integrated into the main branch, one contribution at the time
 - Unit tests for all integrated units; regression testing
- **Constraints on these activities**
 - **Test first.** Unit tests are written before units. They are written by the developer
 - When defects are discovered, a unit test is created to reproduce the defect
 - **Refactor** before extending the source code.

How much control in XP?

- **Reduced number of formal meetings**
 - Daily stand up meeting for status communication
 - No discussions to keep the meeting short
- **No inspections and no peer reviews**
 - Pair programming is used instead
 - Production code is written in pairs, review during coding.
- **Self-organizing system with a leader:**
 - The Leader communicates the vision of the system
 - The leader does not plan, schedule or budget
 - The leader establishes an environment based on collaboration, shared information, and mutual trust
 - The leader ensures that a product is shipped.

Summary of the XP Methodology

Planning	Collocate the project with the client, write user stories with the client, frequent small releases (1-2 months), create schedule with release planning, kick off an iteration with iteration planning, create programmer pairs, allow rotation of pairs
Modeling	Select the simplest design that addresses the current story; Use a system metaphor to model difficult concepts; Use CRC cards for the initial object identification; Write code that adheres to standards; Refactor whenever possible
Process	Code unit test first, do not release before all unit tests pass, write a unit test for each uncovered bug, integrate one pair at the time
Control	Code is owned collectively. Adjust schedule, Rotate pairs, Daily status stand-up meeting, Run acceptance tests often and publish the results.

Scrum

- What is Scrum?
- History of Scrum
- Agile Alliance
- Agile Project Management
- Functionality of Scrum
- Components of Scrum
 - Scrum Roles
 - The Process
 - Scrum Artifacts
- Scaling Scrum
- Evolution of Scrum
- Conclusion

Introduction

- Classical software development methodologies have some disadvantages:
 - Huge effort during the planning phase
 - Poor requirements conversion in a rapid changing environment
 - Treatment of staff as a factor of production
- Agile Software Development Methodologies
 - Minimize risk → short iterations
 - Real-time communication (preferable face-to-face) → very little written documentation
 - www.agilealliance.org

Scrum

- Definition (Rugby): A Scrum is a way to restart the game after an interruption,
 - The forwards of each side come together in a tight formation and struggle to gain possession of the ball when it is tossed in among them
- Definition (Software Development): Scrum is an agile, lightweight process
 - To manage and control software and product development with rapidly changing requirements
 - Based on improved communication and maximizing cooperation.

History of Scrum

- 1995:
 - Jeff Sutherland and Ken Schwaber analyse common software development processes
 - Conclusion: not suitable for empirical, unpredictable and non-repeatable processes
 - Proposal of Scrum
 - Enhancement of Scrum by Mike Beedle
 - Combination of Scrum with Extreme Programming
- 1996: Introduction of Scrum at OOPSLA
- 2001: Publication “Agile Software Development with Scrum” by Ken Schwaber & Mike Beedle
- Founders are also members in the Agile Alliance.

Why Scrum ?

Traditional methods are
like relay races



Agile methods are like
rugby



Image sources: http://en.wikipedia.org/wiki/File:Relay_race_baton_pass.jpg
http://upload.wikimedia.org/wikipedia/commons/b/bf/Rugby_ST.F-ST.T_27022007-19.JPG
For reuse see http://commons.wikimedia.org/wiki/Commons:Reusing_content_outside_Wikimedia

Practicing a Scrum



Real Scrums



Sources: <http://www.youtube.com/watch?v=Dt9MuKZGOiA>
<http://www.youtube.com/watch?v=qOhoJSmHado>

Testudo: Battle Formation used by the Romans



<http://www.youtube.com/watch?v=mhWXElA-4aM&feature=Playlist&p=FB93AB56308E6107&index=0&playnext=1>

Manifesto for Agile Software Development

- <http://www.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.

Methodology Issues

- Methodologies provide guidance, general principles and strategies for selecting methods and tools in a given project environment
- Key questions for which methodologies provide guidance:
 - How much involvement of the customer?
 - How much planning?
 - How much reuse?
 - How much modeling before coding?
 - How much process?
 - How much control and monitoring?

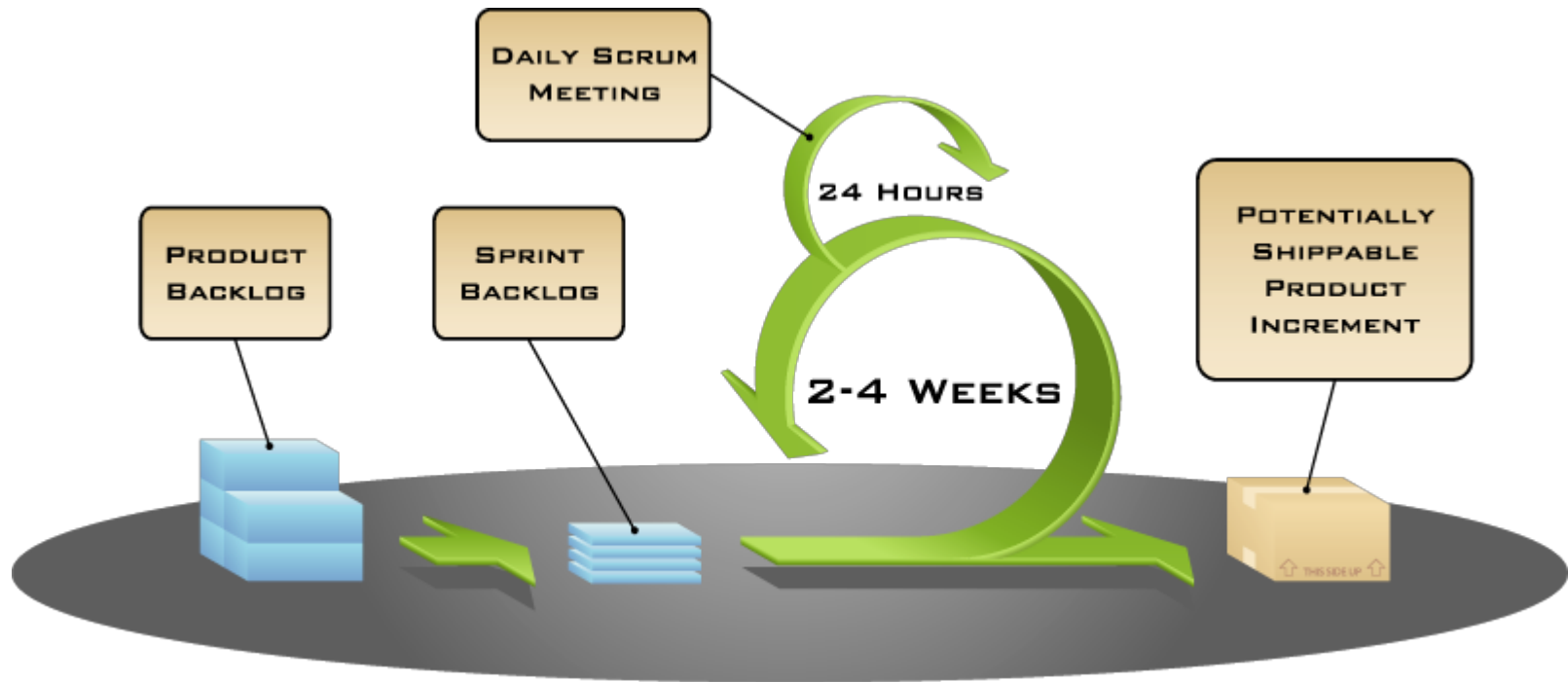
Scrum as Methodology

- Involvement of the customer
 - Onsite customer
- Planning
 - Checklists and incremental daily plans
- Reuse
 - Checklists from previous projects
- Modeling
 - Models may or may not be used
- Process
 - Iterative, incremental process
- Control and Monitoring
 - Daily meetings.

Components of Scrum

- Scrum Roles
 - Scrum Master, Scrum Team, Product Owner
- Process
 - Sprint Planning Meeting
 - Kickoff Meeting
 - Sprint (~~ Iteration in a Unified Process)
 - Daily Scrum Meeting
 - Sprint Review Meeting
- Scrum Artifacts
 - Product Backlog, Sprint Backlog
 - Burndown Charts

Overview of Scrum



COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

Source: <http://www.mountaingoatsoftware.com/scrum-figures>

Scrum Master

- Represents management to the project
- Typically filled by a project manager or team leader
- Responsible for enacting scrum values and practices
- Main job is to remove impediments.

The Scrum Team

- Typically 5-6 people
- Cross-functional (QA, Programmers, UI Designers, etc.)
- Members should be full-time
- Team is self-organizing
- Membership can change only between sprints

Product Owner

- Knows what needs to be build and in what sequence this should be done
- Typically a product manager

Scrum Process Activities

- Project-Kickoff Meeting
- Sprint Planning Meeting
- Sprint
- Daily Scrum Meeting
- Sprint Review Meeting

Project-Kickoff Meeting

- A collaborative meeting in the beginning of the project
 - Participants: Product Owner, Scrum Master
 - Takes 8 hours and consists of 2 parts (“before lunch and after lunch”)
- Goal: Create the Product Backlog

Sprint Planning Meeting

- A collaborative meeting in the beginning of each Sprint
 - Participants: Product Owner, Scrum Master and Scrum Team
- Takes 8 hours and consists of 2 parts (“before lunch and after lunch”)
- Goal: Create the Sprint Backlog

Sprint

- A month-long iteration, during which is incremented a product functionality
- No outside influence can interference with the Scrum team during the Sprint
- Each day in a Sprint begins with the Daily Scrum Meeting

Daily Scrum Meeting

- Is a short (15 minutes long) meeting, which is held every day before the Team starts working
- Participants:
 - Scrum Master (which is the chairman), Scrum Team
- Every Team member should answer on 3 questions

Questions for each Scrum Team Member

1. Status:

What did I do since the last Scrum meeting?

2. Issues:

What is stopping me getting on with the work?

3. Action items:

What am I doing until the next Scrum meeting?

Summary

- XP and Scrum are agile software development methodologies with focus on
 - Empirical process control model
 - Changing requirements are the norm
 - Controlling conflicting interests and needs
- Very simple processes with clearly defined rules
- Self-organizing teams, where each team member carries a lot of responsibility
- No extensive documentation
 - Possibility for “undisciplined hacking”.