

# Software Engineering II

TUM

## Examples of Methodologies: Royce and XP

**Bernd Bruegge**

Technische Universität München  
Institut für Informatik  
Lehrstuhl für Angewandte Softwaretechnik  
<http://www.bruegge.in.tum.de>  
11 July 2006

# Miscellaneous

- Student Feedback
- Tomorrow's exercise
- Next week: Guest lecture by Frank Mang
  - How to write winning proposals, 18. July 2006
  - No exercise
- Date for Final is now "final":
  - No lecture and exercise on Tuesdays
  - HS 2 Wednesday 26 July 2006, 18:00-19:30.

# SE II Exercises: Student Feedback

- Our team liked the SCRUM-Exercise very much, the icebreaker a while ago as well.
- The invited talks were also very nice in general, maybe it's better to have invited talks from more than one companies.

# SE II Exercises: Student Feedback ctd

- Mir persönlich haben die Übungen in diesem Semester sehr viel Spaß gemacht und ich beurteile die große Praxisnähe, das "Learning-by-doing" und den Einbezug von Gastdozenten von Accenture sehr positiv.
- So hat man eine gute Vorstellung von den besprochenen Abschnitten des Projektmanagement bekommen, auch wenn man sich manches Mal gerne etwas mehr Zeit für die Bearbeitung der Aufgaben im Vorfeld der Übung gewünscht hätte und sich nicht immer sicher war, wie diese zu bearbeiten sind.
- Hier könnte man vielleicht in der vorhergehenden Vorlesung noch stärker auf die Aufgabe Bezug nehmen. Alles in allem möchte ich aber meinen positiven Eindruck von dem Konzept unterstreichen.

# Tomorrow's Exercise: Testing in XP

- Bring your laptop installed with Eclipse 3.2
  - Home for Eclipse
    - <http://www.eclipse.org/>
  - Download URLs
    - <http://download3.eclipse.org/eclipse/downloads/>
  - Make sure JUnit Plugin is available
- Exercise will focus on
  - XP's mantra: Test first
  - Regression testing
  - Generation of stubs for methods
  - Generation of stubs for APIs.

# 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?

# Today's Lecture:

## Walk through 2 Methodologies

- Royce Methodology
- XP (Extreme Programming)
  
- Literature:
  - W. Royce, Software Project Management: A Unified Framework. Addison-Wesley, 1998
  - K. Beck, Extreme Programming Explained, Addison-Wesley, 1999

# Royce's Methodology ("Mantras")

- **Demonstration-based approach**
  - Identify performance issues
  - Assess intermediate artifacts
- **Architecture-first approach**
  - Focus on critical use cases
  - Make architecture decisions before committing resources.
  - Address architecture and plan together
- **Iterative life-cycle process**
  - Each iteration should focus on a specific risk
  - Requirements, architecture and plan should be moved in a balanced manner
- **Component-based development**
  - Minimize human generated code
  - Use commercial components.

# Royce's Methodology ("Mantras")

- **Change management environment**
  - Automate processes to deal with changes introduced by iterations
- **Round-trip engineering**
  - Couple models with source code
- **Objective quality control**
  - Use metrics to assess progress
- **Languages**
  - Use visual languages to support modeling and documentation.

# How much Planning? (Royce)

The project plan is developed iteratively like the software

- The plan is refined as the stakeholders increase their knowledge in the application and solution domain
- Planning errors are treated like software defects
  - Early fixing means less impact on project success

WBS is organized around the activities of the Unified Process

- First level elements in the WBS represent workflows (management, requirements, design,...)
- Second level elements represent phases (inception, elaboration, construction, and transition)
- Third level elements are artifacts produced during the phases

Estimation:

- Compute the initial estimate with a model
- Refine it with the project manager, developers, and testers

After each iteration, the plan is revised and estimates are updated.

# How much Reuse? (Royce)

- **Buy versus build** decisions are treated as risks to be addressed early in the life cycle (e.g., in the first iteration of the elaboration phase)
  - When components are reused in more than one project, the return on investment can be further increased
- **Key principle:** Minimize the amount of human-generated source code
  - Reuse commercial components
  - Use code generation tools
  - Use high-level visual and programming languages
- Reuse is treated as a return on investment decision which decreases development time
  - Mature components and tools also reduce time to repair defects
  - Immature components and tools increase quality problems drastically off-setting any economic advantage.

# How much Modeling? (Royce)

Modeling artifacts based on the activities of the Unified Process

- Management Set:
  - Artifacts associated with planning and monitoring activities
  - Ad hoc notations to capture the “contracts” among project participants and other stakeholders
  - Problem statement, SPMP, SCMP and status descriptions
- Requirements set
  - Visionary scenarios, prototypes for user interfaces, requirements analysis model
- Design set
  - Software architecture and interface specifications
- Implementation set
  - Source code, components, executables
- Deployment set
  - Deliverables negotiated between project manager and client
  - Executable, user manual and administrator manual
- Test artifacts are part of each of the above sets.

# How much Process? (Royce)

- Royce's methodology includes guidelines for tailoring the life cycle process based on the technical and managerial complexity of the project. Factors that influence the tailoring:
- **Scale** (Most important factor in determining the process)
  - **Small Projects (1-10 participants)**
    - Require less management, performance depends on skills of participant and on tools
    - Focus on artifacts, few milestones, no formal processes
  - **Larger Projects (more than 10 participants)**
    - Management skills of team leaders is primary bottleneck
    - Well-defined milestones, focus on change management
- **Stakeholder cohesion**
  - Cooperating stakeholders: flexible plan, informal agreements
  - Contenting stakeholders: formal agreements, well-defined processes
- **Process flexibility**: Rigor of contract => rigor of process definition
- **Process maturity**: Mature Organizations are easier to manage
- **Architectural risk**: Demonstrate feasibility of architecture before full-scale commitment
- **Domain experience**: Domain expertise shortens the earlier phases of the life cycle.

# How much Control? (Royce)

3 management metrics and 4 quality metrics:

- Management metrics:
  - **Work**: How many tasks have been completed compared to the plan?
  - **Costs**: How many resources have been consumed compared to the budget?
  - **Team dynamics**: How many participants leave the project prematurely and how many new participants are added?
- Quality metrics:
  - **Change**: How many change requests are issued over time?
  - **Breakage**: How much source code is reworked per change?
  - **Rework**: How much effort is needed to implement a change?
  - **Mean time between failures**: How many defects are discovered per hours of testing?

# Summary of Royce's Methodology

Issues	Methods
<b>Planning</b>	Evolutionary WBS, Initial model-based estimation of cost and schedule (COCOMO II), Iteration planning, including all stakeholders
<b>Modeling</b>	Critical use cases and driving requirements first, Architecture first, UML, Round-trip engineering
<b>Reuse</b>	Buy vs. build decisions during elaboration, Focus on mature components
<b>Process</b>	Scale, Stakeholder cohesion, Process flexibility, Process maturity, Architectural risk, Domain experience
<b>Control</b>	Management indicators (work, cost, team dynamics) Quality indicators (change traffic, breakage, rework, MTBF)

# Outline of the Lecture

- ✓ Royce Methodology
- XP

# XP Methodology (“Mantras”)

- **Rapid feedback**
  - Confronting issues early results in more time for resolving issues. This applies both to client feedback and feedback from testing
- **Simplicity**
  - The design should focus on the current requirements.
  - Simple designs are easier to understand and change than complex ones
- **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)

- **Embracing change**
  - Change is inevitable and frequent in XP projects
  - Change is normal and not an exception that needs to be avoided
- **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
  - Stories are high-level use cases that encompass a set of coherent features
  - Developers then decompose each 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.

# Team Organization

- Production code is written in pairs
- 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 planning in XP?

- **Ideal week**
  - 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.

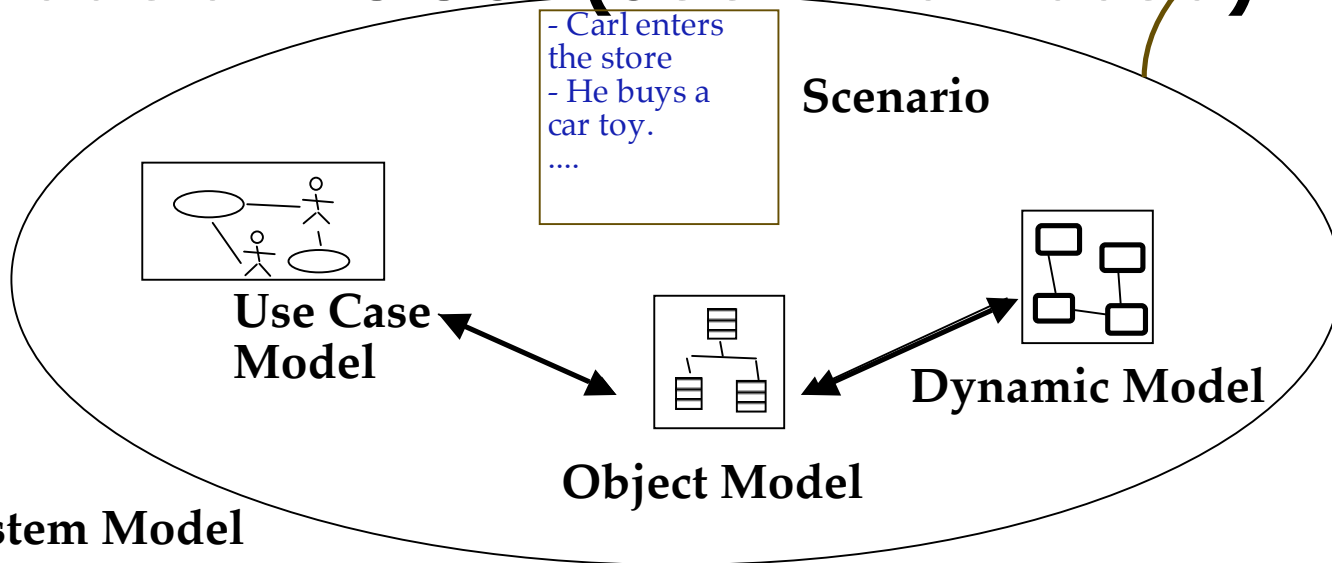
# How much reuse in XP?

- **Simple design**
  - Developers are encouraged to select the most simple solution that addresses the user story being currently implemented
- **No design reuse**
  - 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 specification models (for analysis/design)
  - 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.

# Models in OOSE (Scenario-Based)



Analysis,  
System Design  
Object Design

are Model  
Transformations

System Model

**Reverse Engineering:**  
The system model is  
reconstructed from existing  
source code (legacy systems)

**Forward Engineering:**  
Source code is generated from  
the system model (Ideal: „0 %  
manual coding“, Component-  
Based Software Engineering)

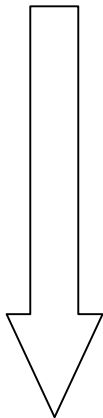
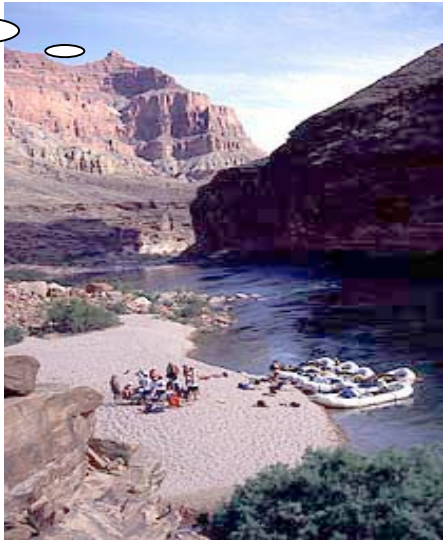
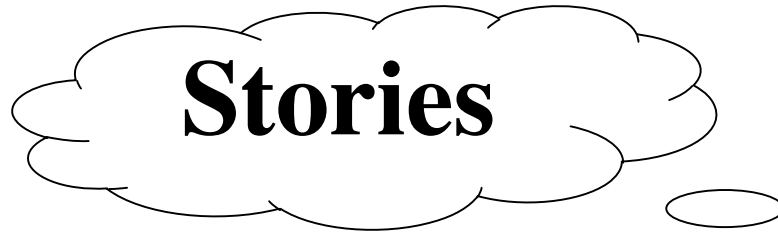
```
class...
class...
class...
```

Source Code

Textual scenarios  
generate external models

**Refactoring:**  
The source code is  
transformed according to  
refactoring rules (program  
transformation)

# Models in XP (Story-Based)



```
class...  
class...  
class...
```

Source Code

Stories  
generate source code

**Refactoring:**  
The source code is transformed according to refactoring rules (program transformation)

# How much process in XP?

- Iterative life cycle model with 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 are implemented. They are written by the developer
  - When defects are discovered, another unit test is created to reproduce the defect
  - Always see if you can refactor before modifying or extending the source code.

# Write the test for getName() before writing getName()

```
*PersonTest.java X Person.java

package organization;

import junit.framework.TestCase;

public class PersonTest extends TestCase {

    public void testNewPerson() {
        Person p = new Person("Max");
        assertNotNull(p);
    }

    public void testGetName() {
        Person p = new Person("Max");
        assertEquals("Max", p.getName());
    }
}
```

# How much control?

- 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

<p><b>Planning</b></p>	<p>Collocate the project with the client , Write user stories with the client, Plan frequent small releases (1-2 months) Create schedule with release planning, Kick off an iteration with iteration planning, create programmer</p>
<p><b>Modeling</b></p>	<p><del>pairs, allow rotation of pairs</del> 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</p>
<p><b>Process</b></p>	<p>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</p>
<p><b>Control</b></p>	<p>Code is owned collectively. Adjust schedule, Rotate pairs, Daily status stand-up meeting, Run acceptance tests often and publish the results</p>

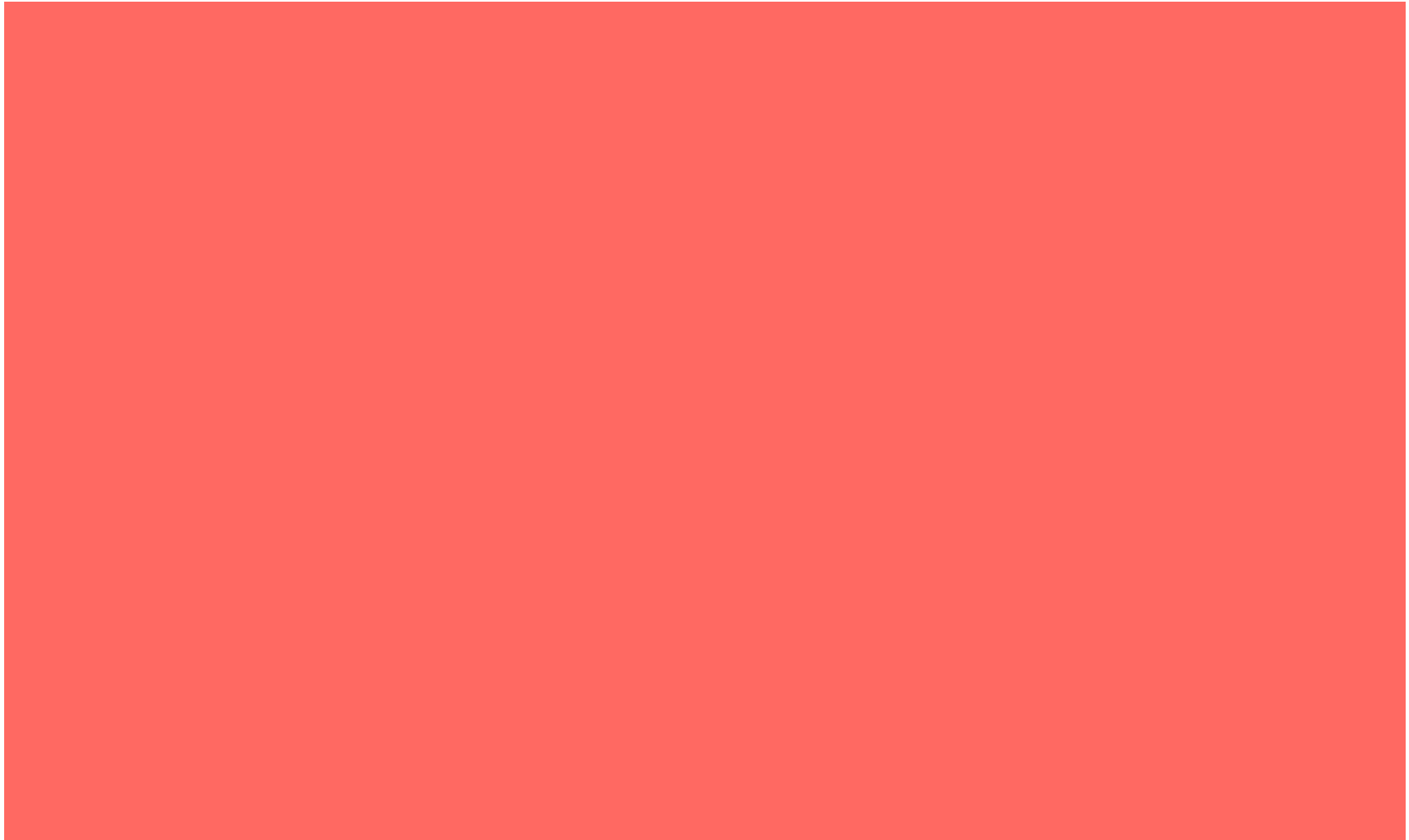
# Summary Methodologies

- Royce (Heavy-weight)
  - Focus on Defined Process
  - Architecture Driven Design
  - Demonstration-based
  - Based on unified Process
- XP (Light-weight)
  - Focus on Technical Development
  - Constant Component Testing
  - Refactoring of selected parts of the system
  - Pair Programming

# References

- Literature:
  - W. Royce, Software Project Management: A Unified Framework. Addison-Wesley, 1998
  - K. Beck, Extreme Programming Explained, Addison-Wesley, 1999

# Backup and Additional Slides



# Scrum

- Focus: Day-to-day management of a software project
- Entity-oriented software lifecycle
- Light-weight Process based on sprints and checklists
  - Product Backlog
  - Sprint Backlog

# Managing a Software Project with Scrum

- **Lists**
  - Project Feature List ("Product Backlog"): Todos for whole project
  - Iteration Feature List ("Sprint Backlog"): Todos for iteration
- **Activities (Management)**
  - Establish the Project Backlog: List of features formulated together by developers, manager and customer
  - Establish the Sprint Backlog: List of items to be implemented in the current iteration
- **Meetings**
  - Kickoff Meeting: Project start, establishing scrum teams
  - Sprint Planning Meeting: Start of an iteration ("sprint"), list of prioritized features to be implemented
  - Daily Scrum: Daily informal status meeting, 15 min
  - Sprint Review Meeting: Demonstration of features in running system, shown to management and customer.

# Characteristics of a Scrum Project

- No formal planning phase
  - No PERT Charts, cost estimates, roles nor assignments
  - No formal agendas
- Basis of work is done by teams
  - Instead of waiting for instructions, team takes initiative
  - Teams are put in a time-box and told to create a product
  - The teams focus on what can be done and how the problem can be solved with the available resources
- Teams are self-organizing
  - Roles, tasks and WBS established by the teams
  - Cost estimates are also done by the teams
- Product functionality is delivered incrementally
  - Increment is regularly built and discussed (“daily build”)
- Scrum Room
  - Single place for informal daily meetings.

# Scrum Kickoff Meeting

- Identify the Product Backlog
  - Product Backlog is never expected to be finalized, even if the project is finished
  - Product Backlog content comes from anywhere: Users, Customers, Sales, Marketing, Customer Service and Engineering
- Identify the product owner
  - Only the product owner can prioritize the Product Backlog
- Set up Scrum Teams
  - These are cross-functional team that perform all the development
    - Take up as much from the Product Backlog as they think they can turn into an increment of product functionality within a 30 day iteration or a sprint.

# Product Backlog

- Here the list of to-dos for the whole project

# Sprint 1: Sprint Planning Meeting

## Actions:

- Talk to the customer: Understand the task at hand.
  - What problem do we need to solve?
- Check the sprint backlog (“list of issues”)
  - Sort the issues by highest-priority
- Training:
  - What skills do we have?
  - What skills do we need to learn? (Read Ant, Read XML, Modify Configuration Makefile)
- Identify product functionality for the sprint
  - What needs to be done to build it?
  - Identify actions for the team members
  - Create the sprint backlog

# Sprint 1: Daily Scrum Meeting

- Check the sprint backlog (“list of open issues”)
- Sort the issues by highest-priority
- Are there new issues? Add them to the sprint backlog
- Identify action items for the team members