Requirements
System Design (Architecture, High-level Design)
Module Design (Program Design, Detailed Design)
Implementation of Units (classes, procedures, functions)
Unit testing
Module Testing (Integration testing of units)
System Testing (Integration testing of modules)
Acceptance Test (Release testing)

Validate Requirements, Verify Specification
Verify System Design
Verify Module Design
Verify Implementation

Project Management, Software Quality Assurance (SQA), Supporting Tools, Education

Maintenance
Project vs. Process

**Project**

A project is a temporary endeavor undertaken to create a unique product or service.

Project Management Institute
Project vs. Process

Process

- Ordered set of activities
- May contain sub-processes
- Goal of each activity
- Each activity has entry/exit criteria and input/output
- Constraints

Processes are reoccurring
A software life-cycle model

(Software Engineering Process Model)

representation or simplified version of a concept, phenomenon, relationship, structure, system, or an aspect of the real world.

LiU EXPANDING REALITY
Model of a life-cycle (a Process Model)

Carol the customer

Diana the developer

Time
A familiar model?

- **Requirements**
- **System Design** (Architecture, High-level Design)
- **Module Design** (Program Design, Detailed Design)
- **Implementation of Units** (classes, procedures, functions)
- **Verification**
  - Requirements, Verify Specification
  - System Design
  - Module Design
  - Implementation
  - Module Testing (Integration testing of units)
  - System Testing (Integration testing of modules)
  - Acceptance Test (Release testing)
  - Maintenance

*Time*
The waterfall model

- One of the first life-cycle models (Royce, 1970)
- The waterfall development model originates in the manufacturing and construction industries
- Very common, very criticized
The Waterfall model

Requirements
System Design
Module Design
Implementation
Unit Testing
Module Testing
System Testing
Acceptance Test
Maintenance

Time

Finish each phase before continue to next.

Milestone and deliverable at each step. (Artifacts such as Design document, Req. Specification. etc.).
The Waterfall Model

- Why is the waterfall model so criticized?
- Which are the problems?
- Can it be useful sometimes?
The Waterfall Model - some arguments

Cons

- Software requirements change, hard to sign-off on a SRS.
- Early commitment. Changes at the end, large impact.
- Feedback is needed to understand a phase. E.g. implementation is needed to understand some design.
- Difficult to estimate time and cost for the phases.
- Handling risks is not an explicit part of the model. Pushes the risks forward.
- Software "is not" developed in such a way. It evolves when problems are more understood. Little room for problem solving.
The Waterfall model - some arguments

Pros

- Simple, manageable and easy to understand
- Fits to common project management practices (milestones, deliverables etc.)
- Can be suitable for short projects (some weeks)
- Can be suitable for "stable" projects, where requirements do not change
- Focus on documents, saves knowledge which can be reused by other people.
- Can be suitable for fixed-price contracts
Can we improve the model?

Danger! E.g. a performance problem can result in a major requirements change. Very expensive rollback…
Do it twice?

First round, a prototype
Second round, do it right.

The original paper is actually misunderstood!
(Royce, 1970) includes
- Iteration of phases
- "Do it twice" prototype

Input to the phases in the second round
Is overlapping phases a solution?

When do we "sign-off", e.g. when do we have all requirements?

What if a major design flaw is discovered at the testing phase?

Time

- requirements
- design
- implementation
- test

Release!
What should be built?

The hardest single part of building a software system is deciding precisely what to build.

(Frederick P. Brooks)

How? By delivering several releases?
Iterative Development

When should the releases take place?

**Time-boxing** - The time period is fixed for each iteration.

What should be included in the release?

**Prioritized functionality** - Do the most important parts first.

---

Customer Feedback

Time

**Iteration 1**

**R1**

**Iteration 2**

**R2**

**Iteration 3**

Customer Feedback

Final Release!
Dependent project parameters (revisited)

Calendar time and resources are fixed

Select the most important functions

Select quality. E.g. how general should we be?

Features

Calendar Time

Resources

Project

Liu EXPANDING REALITY
Prioritization of requirements

Customer Value

High

Low

Sweet Spot

Avoid

Development Effort

High

Low
Iterative Development - Cons

Is iterative development the silver bullet?

- Problem with current business contracts, especially fixed-price contracts.
- With short iterations it can be hard to map customer requirements to iterations.
- Overhead added

Customer Feedback:

- Is iterative development the silver bullet?

Diagram:

- Iteration 1
- Iteration 2
- Iteration 3

Time:

Final Release!
Iterative Development - Pros

Pros

• **Misunderstandings and inconsistency** are made clear early (e.g. between requirement, design, and implementation)
• Encourage to use feedback -> elicit the *real* requirements
• Forced to focus on the **most critical issues**
• Continuous testing offers **project assessment**
• **Workload** is spread out over time (especially test)
• The team can get "lesson learned" and continuously **improve the process**
• Stakeholders get concrete **evidence of progress**
We are using an iterative process!

Define a plan with 1..N iterations. We do not have to care about plans...

Now, let's hack!

Is this a good iterative process? Of course not. We need some structure!

Methodologies and defined Processes
Processes, Models, Methodologies...

Process Models

Waterfall model
V- model
Spiral model
Prototype model

"what" at a high level of abstraction

Which is the "best" approach?

Methodologies and defined Processes

Extreme Programming (XP)
Rational Unified Process (RUP)

agile

Scrum

"what" and to a certain level "how"
Fashion industry

Our concept is new
Earlier concepts are old
Either you are in or out!
Do you dare to be out?

... but the basic needs remain
Agile Approaches - Agile Alliance

Lightweight approaches to satisfy the customers with "early and continuous delivery of valuable software"

**Manifesto for Agile Software Development**

*Individuals and interactions* over processes and tools
*Working software* over comprehensive documentation
*Customer collaboration* over contract negotiation
*Responding to change* over following a plan

(http://agilemanifesto.org, 2001)
According to the 2011 CHAOS Manifesto from the Standish Group, Agile projects are 3 times more often successful than non-agile projects. Has been criticized for selection method.
Scrum

Approach public in 1995 at OOPSLA

"Scrum" strategy used in rugby for getting an out-of-play ball back into play.
Scrum in a nutshell

Small, cross-functional teams

Product split into small, roughly estimated, stories

Iterations - sprints

Continuous improvement and deployment
The Sprint

Sprint end date and deliverable do not change
The Team

Sprint end date and deliverable do not change
The Scrum Master

Scrum Master

The Team

Sprint end date and deliverable do not change
The Product Owner

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

The Team

Sprint end date and deliverable do not change
The Product Backlog

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

Product Owner

The Team

A prioritized list of what is required, features, stories

1

2

3

4

5

6

7

Product Backlog

Sprint end date and deliverable do not change

Every 24 Hours

1-4 Week Sprint

LiU EXPANDING REALITY
The Sprint Planning Meeting

**Inputs from Executives, Stakeholders, Customers, Users, Team**

- **Scrum Master**
- **Product Owner**
- **The Team**

**Product Backlog**

A prioritized list of what is required, features, stories

Team selects starting at top as much as it can commit to deliver by end of sprint

**Sprint Planning Meeting**

- Sprint end date and deliverable do not change
The Sprint Backlog

- **Inputs from Executives, Stakeholders, Customers, Users, Team**
- **Scrum Master**
- **Product Owner**
- **The Team**

**Product Backlog**

1. A prioritized list of what is required, features, stories
2. 
3. 
4. 
5. 
6. 
7. 

**Sprint Planning Meeting**

- **Task Breakout**
- **Sprint Backlog**
- **Sprint end date and deliverable do not change**

**Every 24 Hours**
Sample Taskboard
The Daily Scrum Meeting

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

The Team

Product Owner

Product Backlog

1. A prioritized list of what is required, features, stories

2. Team selects starting at top as much as it can commit to deliver by end of sprint

3. Sprint Planning Meeting

4. Task Breakout

5. Sprint Backlog

6. Sprint end date and deliverable do not change

7. Daily Scrum Meeting

1-4 Week Sprint

Every 24 Hours

LiU EXPANDING REALITY
The Burn Down Charts

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

The Team

Burn down charts

Daily Scrum Meeting

Every 24 Hours

1-4 Week Sprint

Sprint end date and deliverable do not change

Task Breakout

Sprint Planning Meeting

Product Backlog

A prioritized list of what is required, features, stories

Team selects starting at top as much as it can commit to deliver by end of sprint

Product Owner

Product Backlog
The burn down chart

- Only track hours remaining, not hours worked
- X – days (in Sprint)
- Y – hours remaining in estimated time
- Remove meeting time, vacation etc. from total available hours
- Update only when PBIs are DONE
- When not done – Undone PBIs
Electronic Taskboard
The Sprint Review Meeting

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

The Team

Product Owner

Burn down charts

Daily Scrum Meeting

Sprint review meeting

Every 24 Hours

1-4 Week Sprint

Sprint Backlog

Sprint end date and deliverable do not change

Done?

Finished work

Team selects starting at top as much as it can commit to deliver by end of sprint

A prioritized list of what is required, features, stories

Sprint Planning Meeting

Task Breakout

Product Backlog
The Definition of Done!

- When are we done?
- “No more remaining work”
- Includes testing, documentation etc.
- Possible to ship after each sprint
- Everybody – understand what done means

Tools to support done
- Version handling (SCM)
- Automated build
- Automated tests (Continuous integration)
The Sprint Retrospective

Inputs from Executives, Stakeholders, Customers, Users, Team

Scrum Master

Product Owner

The Team

Burn down charts

Daily Scrum Meeting

Sprint review meeting

Finished work

1-4 Week Sprint

Every 24 Hours

A prioritized list of what is required, features, stories

Team selects starting at top as much as it can commit to deliver by end of sprint

Sprint Planning Meeting

Sprint Backlog

Task Breakout

Sprint end date and deliverable do not change

Product Backlog
The two pillars of the Toyota production system are just-in-time production and automation with a human touch, or autonomation. The tool used to operate the system is kanban.
Kanban is a Japanese word that means “visual card,” “signboard,” or “billboard.”

Toyota originally used Kanban cards to limit the amount of inventory tied up in “work in progress” on a manufacturing floor.

Kanban is a lean approach to agile software development.
How does Kanban Work?

• **Visualize the workflow**
  - Split the work into pieces, write each item on a card and put on the wall.
  - Use named columns to illustrate where each item is in the workflow.

• **Limit WIP** (work in progress) – assign explicit limits to how many items may be in progress at each workflow state.

• **Measure the lead time** (average time to complete one item, sometimes called “cycle time”), optimize the process to make lead time as small and predictable as possible.
A simple Kanban Board

Source: http://www.crisp.se/gratis-material-och-guider/kanban
Work In Progress

Work In Progress, WIP, limits are designed to:
- reduce multitasking
- maximise throughput
- enhance teamwork

Reducing multitasking is beneficial for two primary reasons
Reducing Multitasking

20% time is lost to context switching per ‘task’, so fewer tasks means less time lost

(from Gerald Weinberg, Quality Software Management: Systems Thinking)
Reducing Multitasking

Performing tasks sequentially yields results sooner.

multi-tasking A, B and C (on the top), delivers A much later, and even C slightly later, than sequentially (on the bottom).
Typical Measurements

- **Cycle time** – Measured from when you started working on it
- **Lead time** – Measured from when the customer ordered
- **Quality** – Time spent fixing bugs per iteration
- **WIP** – Average number of “stories” in progress
- **Throughput** – Number of “stories” completed per iteration (when using fixed iterations)
Benefits of Kanban

• Eliminate over-production, the #1 waste
• Produce only what is ordered, when ordered, & quantity ordered
• Increase flexibility to meet customer demand
• Competitive advantage by sequencing shipments to customers (what they want, when they want it, in the order they want it!)
eXtreme Programming

XP

Copyright © 2003 United Feature Syndicate, Inc.
Extreme Programming

• Formulated in 1999 by Kent Beck
• XP is “a light-weight methodology for small to medium-sized teams developing software in the face of vague or rapidly changing requirements.”
XP - Some Practices

**Pair Programming**
- Programming as a collaborative conversation
- Focus on task
- Clarify ideas
- Rotate frequently

**Stories**
- "requirements", but not mandatory
- A token for a piece of system capability to be implemented
- Name + short story
- On index cards (paper)

**Continuous Integration**
- Integrate and test often
- Automated build system
- Automated regression tests (e.g. JUnit)

**Refactoring**
- Improve the design of existing code without changing its functionality
- Tool support, e.g. Eclipse

**Test-First Programming**
- Create tests before code
- Focus on interface and "what is needed"
- Gets tests for free
XP Values

• Communication
  • On-site customer, user stories, pair programming, daily standup meetings, etc.

• Simplicity
  • "Do the simplest thing that could possibly work" (DTSTTTCPW) principle

• Feedback
  • Unit tests tell programmers status of the system
  • Programmers produce new releases every 2-3 weeks for customers to review

• Courage
  • Communicate and accept feedback, throw code away, refactor the architecture of a system
The Rational Unified Process

- The Rational Unified Process (RUP) is an iterative software development process framework
- Defined in 1997 by Grady Booch, Ivar Jacobson and James Rumbaugh
- Recognized to be particularly applicable to large projects with large teams
RUP Phases

- Inception
  - Shared understanding of the system with the customer

- Elaboration
  - Architecture to build the system

- Construction
  - Developing the system

- Transition
  - Customer takes ownership of system
RUP - Disciplines and Phases

Core Technical Disciplines

- Business Modeling
- Requirements
- Analysis and Design
- Implementation
- Test
- Deployment

Core Supporting Disciplines

- Change & Config. Mgm.
- Project Mgm.
- Environment.