Software Processes

The software process

- A structured set of activities required to develop a software system
  - Specification;
  - Design;
  - Validation;
  - Evolution.
- Software process model = abstract representation of a process.
  - a description of a process from some particular perspective.

Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development.
- Evolutionary development
  - Specification, development and validation are interleaved.
- Component-based software engineering
  - The system is assembled from existing components.
- Many variants of these models exist

Basically....

...only two types of processes

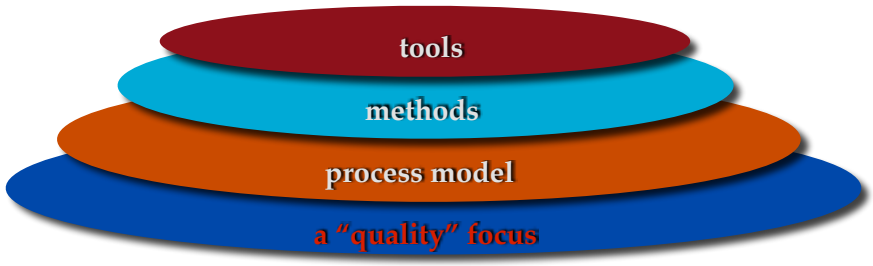
Waterfall

breaks down project based on activities

Iterative

breaks down project based on subsets of functionality

A Layered View of Software Engineering



from **R.S.Pressman, 2005**

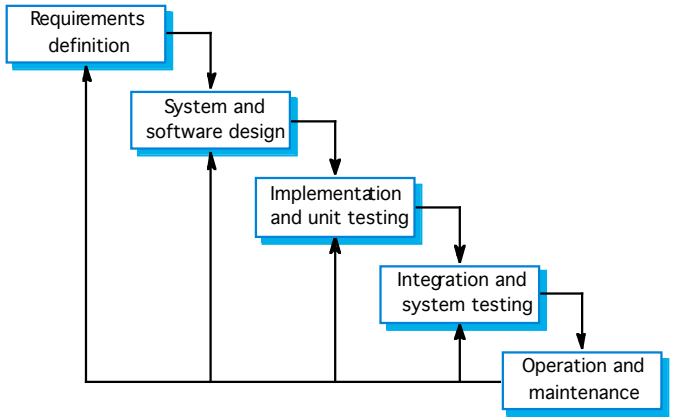
- **Quality Focus** = bedrock
- **Process** = framework and technological glue
- **Methods** = how-to (in requirement, design, construction, testing etc.)
- **Tools** = support for methods and process (CASE)

Phases of Software Engineering

- Definition Phase (**WHAT**)
  - what is processed? what performance? what interfaces? what constraints?
  - (i) information engineering; (ii) requirements eng. ; (iii) project planning
- Development Phase (**HOW**)
  - how to structure data? how to implement functions? how to choose an architecture? how to translate design into implementation?
  - (i) software design; (ii) code construction ; (iii) software testing
- Support Phase (**CHANGE**)
  - change erroneous code; change inadequate code
  - Types of change:
    1. Correction -- error in code
    2. Adaptation -- change in environment
    3. Enhancement -- change in needs

Waterfall Model

Waterfall model



from **I.Sommerville, SE8**

Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

- Main drawback: **difficulty to accommodate change**
  - after the process is underway.
  - One phase has to be complete before moving onto the next phase.

Waterfall model problems

- Inflexible partitioning of the project into distinct stages
  - makes it difficult to respond to changing customer requirements.
- Appropriate when:
  1. requirements are well-understood and
  2. changes will be fairly limited during the design process.

- Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

Basically....

...only two types of processes

Waterfall

breaks down project based on activities

Iterative

breaks down project based on subsets of functionality

Iterative Development

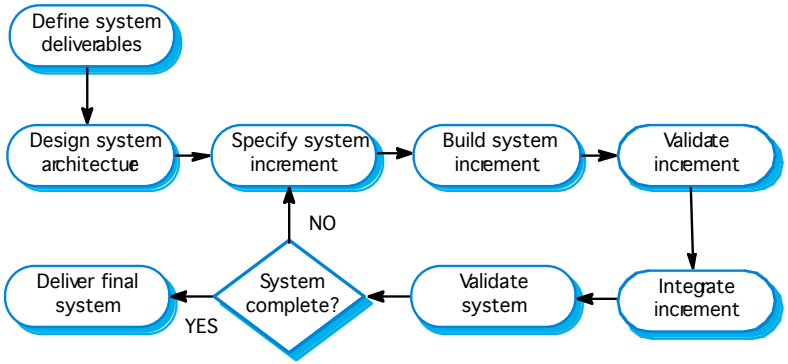
Essence of Iterations

- Get product quality, after each iteration
- No prototype! No draft!
- Real product!

Iterative Process (w. Incremental delivery)

- Development and delivery is broken down into **increments**
  - each increment delivering part of the required functionality
  - instead of delivering the system as a single delivery.
- Prioritized user requirements
  - the highest priority requirements are included in early increments.
- Requirements frozen during the development of an increment
  - yet requirements for later increments can continue to evolve.

Iterative Process (w. Incremental delivery)



from **I.Sommerville, SE8**

Time Boxing

- Set a fix time for each iteration
- When needed...

  ...**Slip features list! Don't slip date of iteration**
- Good exercise for prioritizing requirements

Advantages of Iterative Development

- Customer value can be delivered with each increment
  - system functionality is available earlier.
- Lower risk of overall project failure.
- Highest priority system services tend to receive the most testing.

Advantages of Iterative Development

- Accelerated delivery of customer services.
  - increment delivers top priority functionality.
- User engagement with the system.
  - Users have to be involved
    - the system is more likely to meet their requirements
    - the users are more committed to the system.

Problems of Iterative Development

- Management problems
  - Progress can be hard to judge and problems hard to find because there is no documentation to demonstrate what has been done.
- Contractual problems
  - The normal contract may include a specification; without a specification, different forms of contract have to be used.
- Validation problems
  - Without a specification, what is the system being tested against?
- Maintenance problems
  - Continual change tends to corrupt software structure making it more expensive to change and evolve to meet new requirements.

The Issue of Rework

- Iterative development may lead to changing/deleting code from previous iterations
- This is a waste ... in manufacturing ;-)
- Better to rework than to patch around bad designed code
- Supported by techniques
  - Refactoring
  - Regression Test
  - Continuous Integration

Predictive Planning

- Predictive Planning
  - 1. Make plans (hard to predict)
  - 2. Follow the plan (easy to predict)
- The problem with predictive planning
  - very often changes affect plans
- 2 Ways of dealing with changing requirements:
  - 1. Better requirements analysis
  - 2. Welcome change! :)

Adaptive Planning

- Prediction is an illusion!
- Software is **controllable**, but no **predictable**.
- There is no “according to the plan” in adaptive planning
  - there is planning!
  - but plan is only a baseline
- Predictive: fixed-price / fixed-scope projects
- Adaptive: Fixed price / variable scope

Agile methods

- Dissatisfaction with the overheads involved in design methods led to the creation of agile methods:
  - Focus on the code rather than a preset design;
  - Iterative approach to software development;
  - Deliver working software quickly and evolve this quickly to meet changing requirements.

Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Principles of agile methods

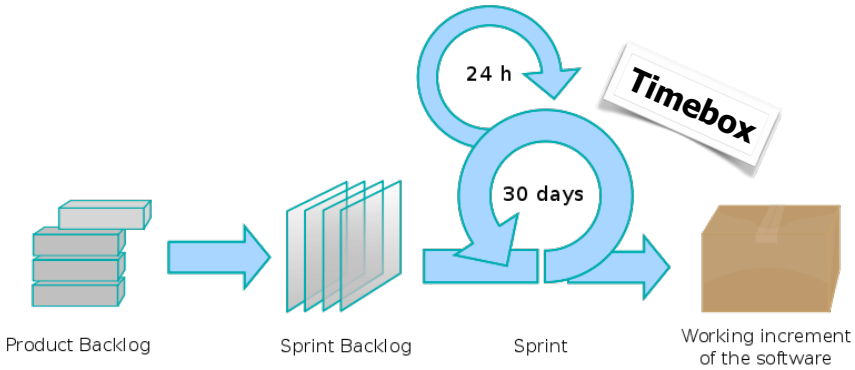
- **Customer satisfaction** by rapid, continuous delivery of useful software
- Software is **delivered frequently** (weeks rather than months)
- **Working software** is the principal measure of progress
- Even **late changes** in requirements are welcomed
- **Close (daily) cooperation** between business people and developers
- **Colocation**: Face-to-face conversation is the best form of communication
- **Trust**: build around motivated individuals, who should be trusted
- Continuous attention to **technical excellence** and good design
- **Simplicity**
- **Self-organizing teams**

Scrum

Scrum Roles

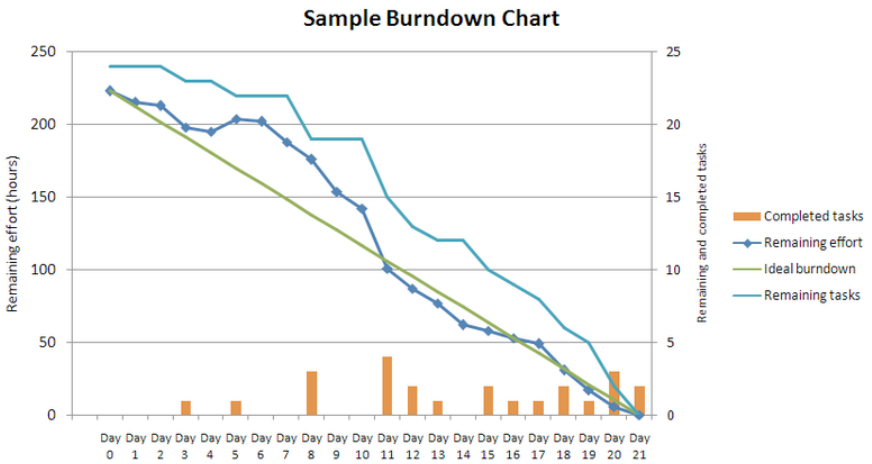
- Scrum Master
  - the Project Manager
- Product Owner
  - represents the stakeholders
- Team

Scrum



http://en.wikipedia.org/wiki/Scrum_(development)

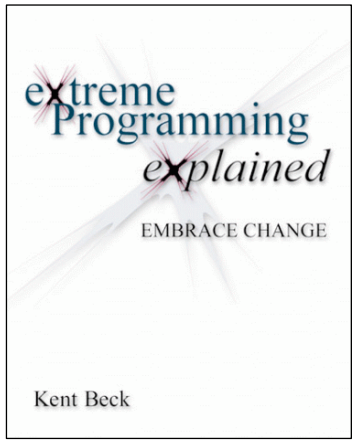
Scrum Burndown Chart



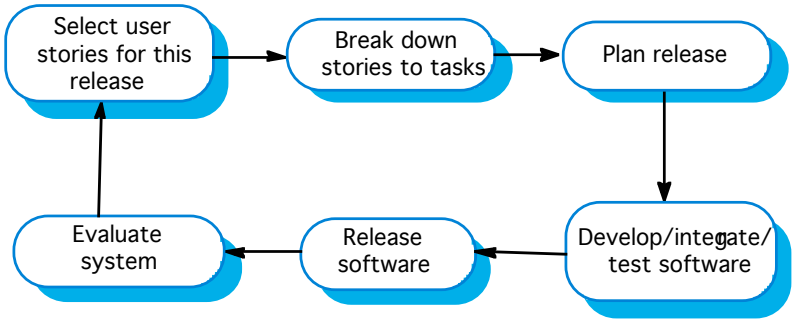
Extreme Programming

Extreme programming (XP)

- Best-known agile method.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - **New versions** each day/night
  - **Increments** are delivered to customers every 2-3 weeks;
  - **All tests** must be run for every build
    - Build is only accepted if tests run successfully.



The XP release cycle



from **I.Sommerville, SE8**

The Story within XP Story Cards...

Date: _____ Story Nr: _____	Type of Activity: New ☐ Fix ☐ Enhance ☐ Funct.Testing ☐ Priority: User _____ Technical _____ Risk: _____ Technical Estimate: _____
Story Name:	
Story Description:	
Notes:	

Task Tracing

Date	Status	ToDo	Comments

Extreme programming practices 1

Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'.
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Extreme programming practices 2

Pair Programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

When doesn't XP work?

- Business Culture
  - Big forehead specifications
  - Obsession with writing useless documents
- Extra-work to prove commitment to the company
  - instead of just going for a 40h work-week... no extra-time!
- Size-matters
  - doesn't work with 100 programmers
  - only <= 10 programmers
- Physical separation
  - even separate floor kills XP
- Environments with high viscosity
  - large build times, impossible to run all tests etc.

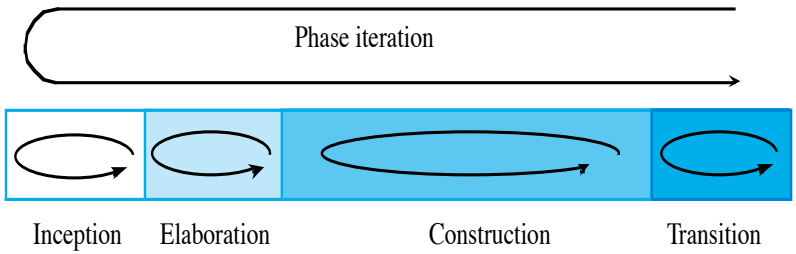
Risk Aware Processes

The Rational Unified Process

- A modern process model derived from the work on the UML and associated process.
- Normally described from 3 perspectives:
  - **Dynamic** perspective: **phases** over time;
  - **Static** perspective: **process activities**;
  - **Practice** perspective: suggests good practice.

Incremental + Iterative + Risk Aware

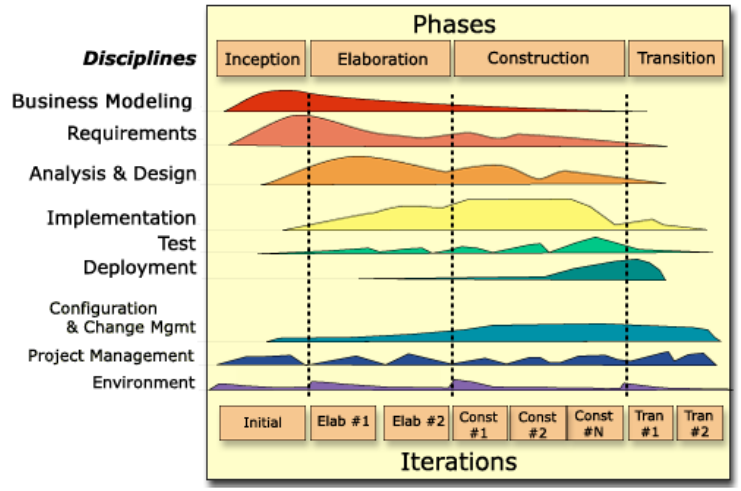
RUP phase model



RUP Dynamic Perspective: Phases

- Inception
  - Establish the business case for the system.
- Elaboration
  - Develop an understanding of the problem domain and the system architecture.
- Construction
  - System design, programming and testing.
- Transition
  - Deploy the system in its operating environment.

RUP Phases-Activities Matrix



RUP Dynamic Perspective: Activities/Workflows/Disciplines

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Test	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system
Project management	This supporting workflow manages the system development
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

RUP good practice

1. Develop software iteratively
2. Manage requirements
   - explicit management: use-cases
3. Visually model software
   - UML diagrams
4. Verify quality of software
5. Control changes to software
   - make use of CMS (Control Management Systems) for versioning, build and integration