CSE 2311

Software Development Project

Click to edit Master text styles
Second level
Third level
Fifth level

Wednesday, January 9, 2013

YORK UNIVERSITÉ UNIVERSITY

NSERC Undergraduate Student Research Awards (USRA) 2013

- Information session
  - Friday, January 11, 1:30–2:30
  - Lassonde 3033 (Comp Science & Eng building)
- Duration and value of awards
  - 16 weeks on a full-time basis (Apr 29 - Aug 16, 2013)
  - $4,500 from NSERC + at least $1,125 from the supervisor
- Application deadline
  - Friday, January 25, 2013, 3:00PM
  - See www.cse.yorku.ca/undergrad/usra/ for procedure

YORK UNIVERSITÉ UNIVERSITY

NSERC Undergraduate Student Research Awards (USRA) 2013

- Eligibility
  - Canadian citizens or permanent residents
  - Full-time students in natural sciences and engineering
  - Completed 18 credits of their degree program by December 31, 2012
  - A cumulative GPA of at least “B”
- Program coordinator
  - Prof. Burton Ma, burton@cse.yorku.ca
- For more info
  - See www.cse.yorku.ca/undergrad/usra/

YORK UNIVERSITÉ UNIVERSITY

Reading

- Author: Ian Sommerville
- Title: Software Engineering
- Chapters 1-4

Software Engineering

- Software Engineering is the science and art of building significant software systems that are:
  1. on time
  2. on budget
  3. with acceptable performance
  4. with correct operation

Software Engineering

- The economies of all developed nations are dependent on software.
- More and more systems are software controlled.
- Software engineering is concerned with theories, methods and tools for professional software development.
- Software engineering expenditure represents a significant fraction of the GNP of developed countries.

Software Costs

- Software costs often dominate system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop.
- Software engineering is concerned with cost-effective software development.

Software Products

- Generic products
  - Stand-alone systems which are produced by a development organization and sold on the open market to any customer.
- Customized products
  - Systems which are commissioned by a specific customer and developed specially by some contractor.

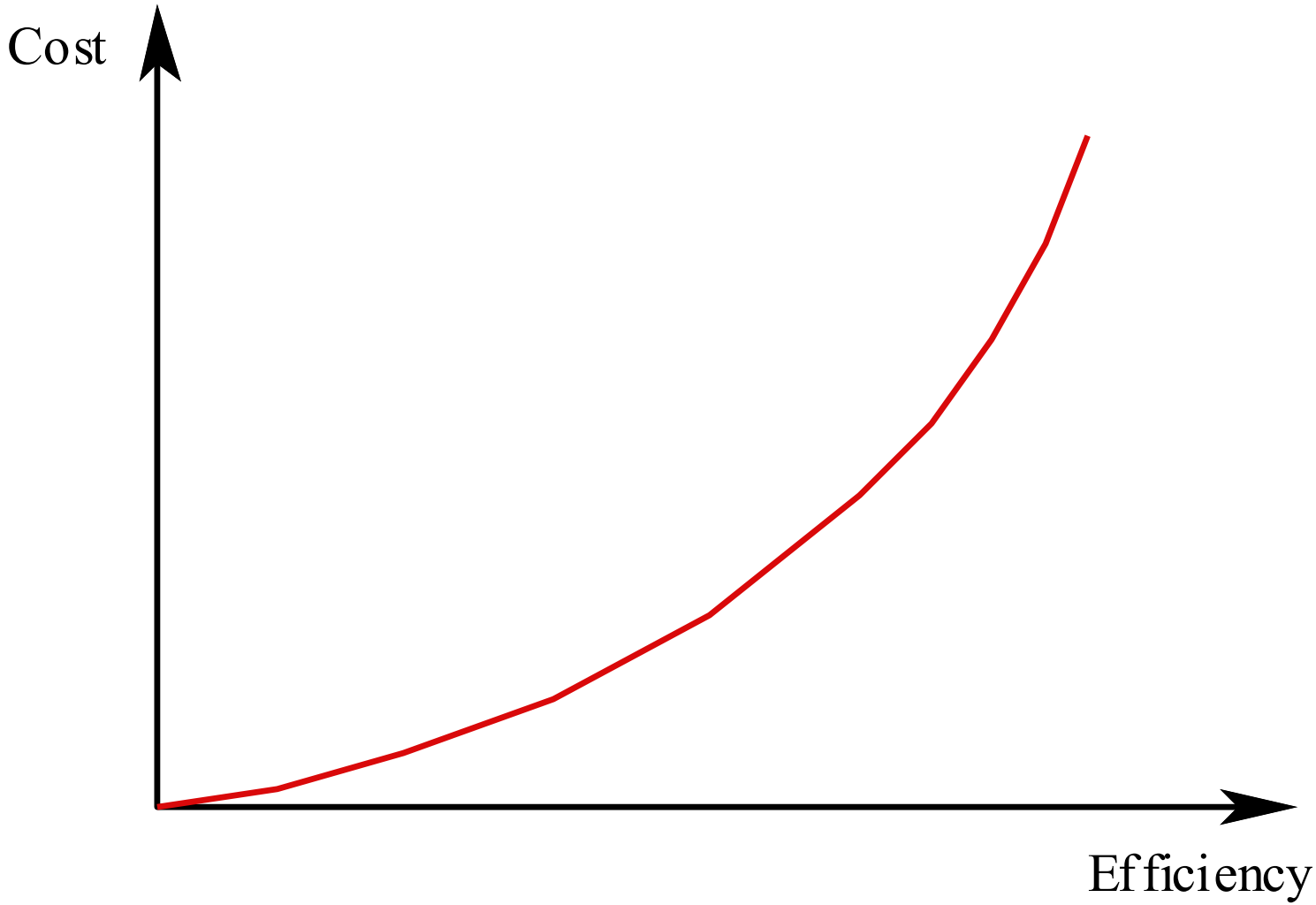
Software Product Attributes

- Maintainability
- Dependability
- Efficiency
- Usability

Importance of Product Characteristics

- The relative importance of these characteristics depends on the product and the environment in which it is to be used.
- In some cases, some attributes may dominate
  - In safety-critical real-time systems, key attributes may be dependability and efficiency.
- Costs tend to rise exponentially if very high levels of any one attribute are required.

Efficiency Costs



The Software Process

- Structured set of activities required to develop a software system
  - Specification
  - Design
  - Validation
  - Evolution
- Activities vary depending on the organization and the type of system being developed.
- Must be explicitly modeled if it is to be managed.

Engineering Process Model

- **Specify**: Set out the requirements and constraints on the system.
- **Design**: Produce a model of the system.
- **Manufacture**: Build the system.
- **Test**: Check the system meets the required specifications.
- **Install**: Deliver the system to the customer and ensure it is operational.
- **Maintain**: Repair faults in the system as they are discovered.

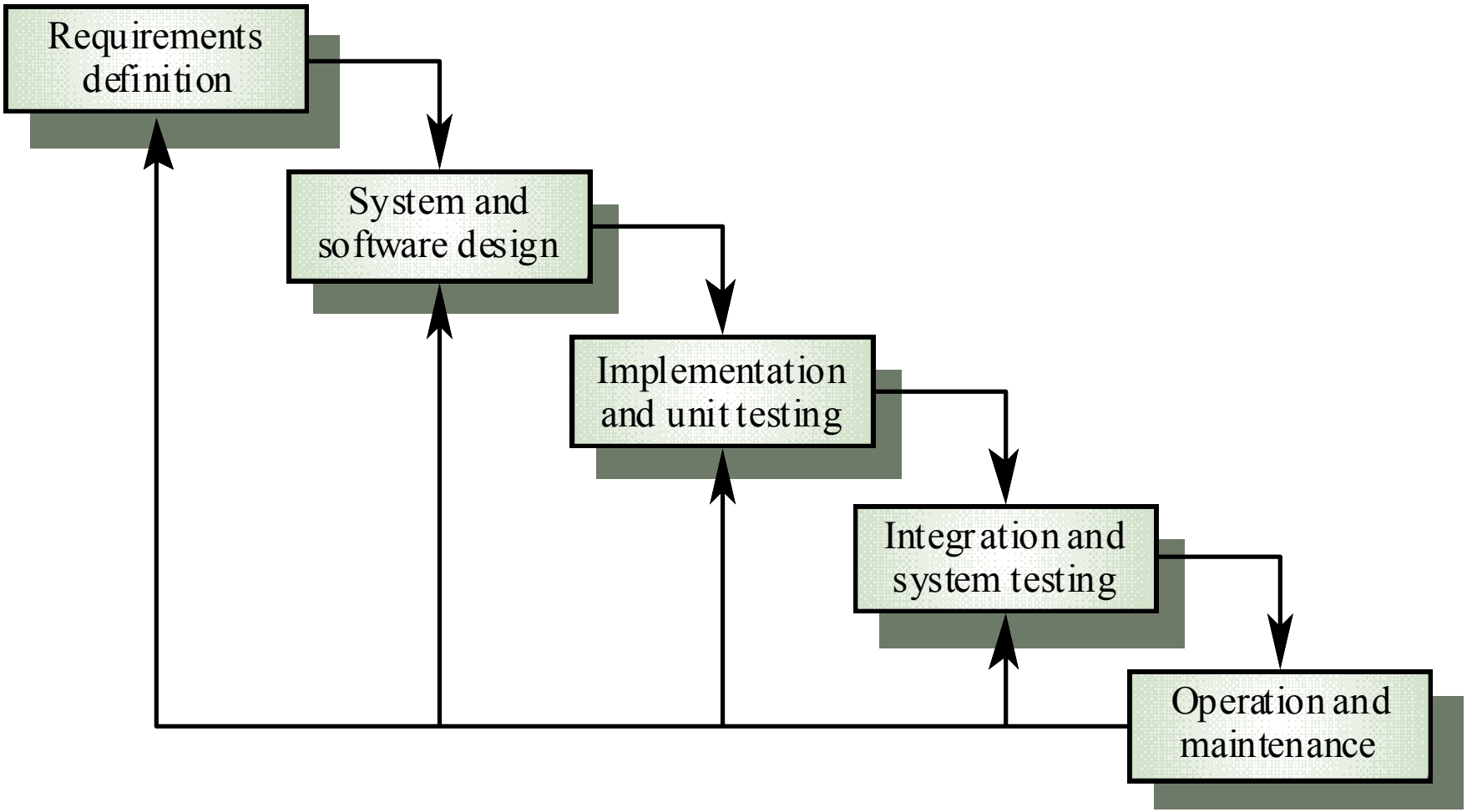
Software Engineering is Different

- Normally, specifications are incomplete.
- Very blurred distinction between specification, design and manufacture.
- No physical realization of the system for testing.
- Software does not wear out - maintenance does not mean component replacement.

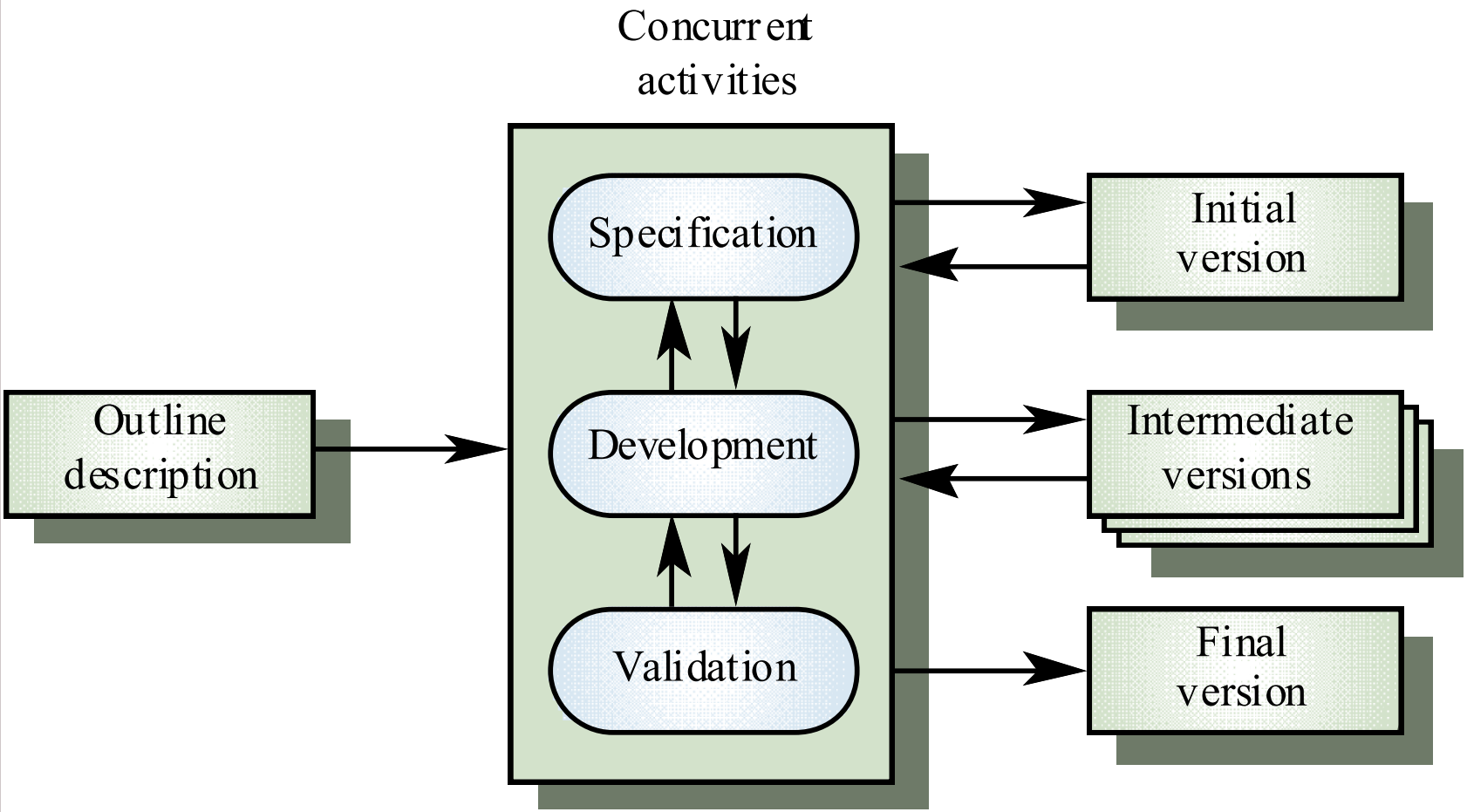
Generic Software Process Models

- **Waterfall**
  - Separate and distinct phases of specification and development
- **Evolutionary**
  - Specification and development are interleaved
- **Formal Transformation**
  - A mathematical system model is formally transformed to an implementation
- **Reuse-based**
  - The system is assembled from existing components

Waterfall Process Model



Evolutionary Process Model



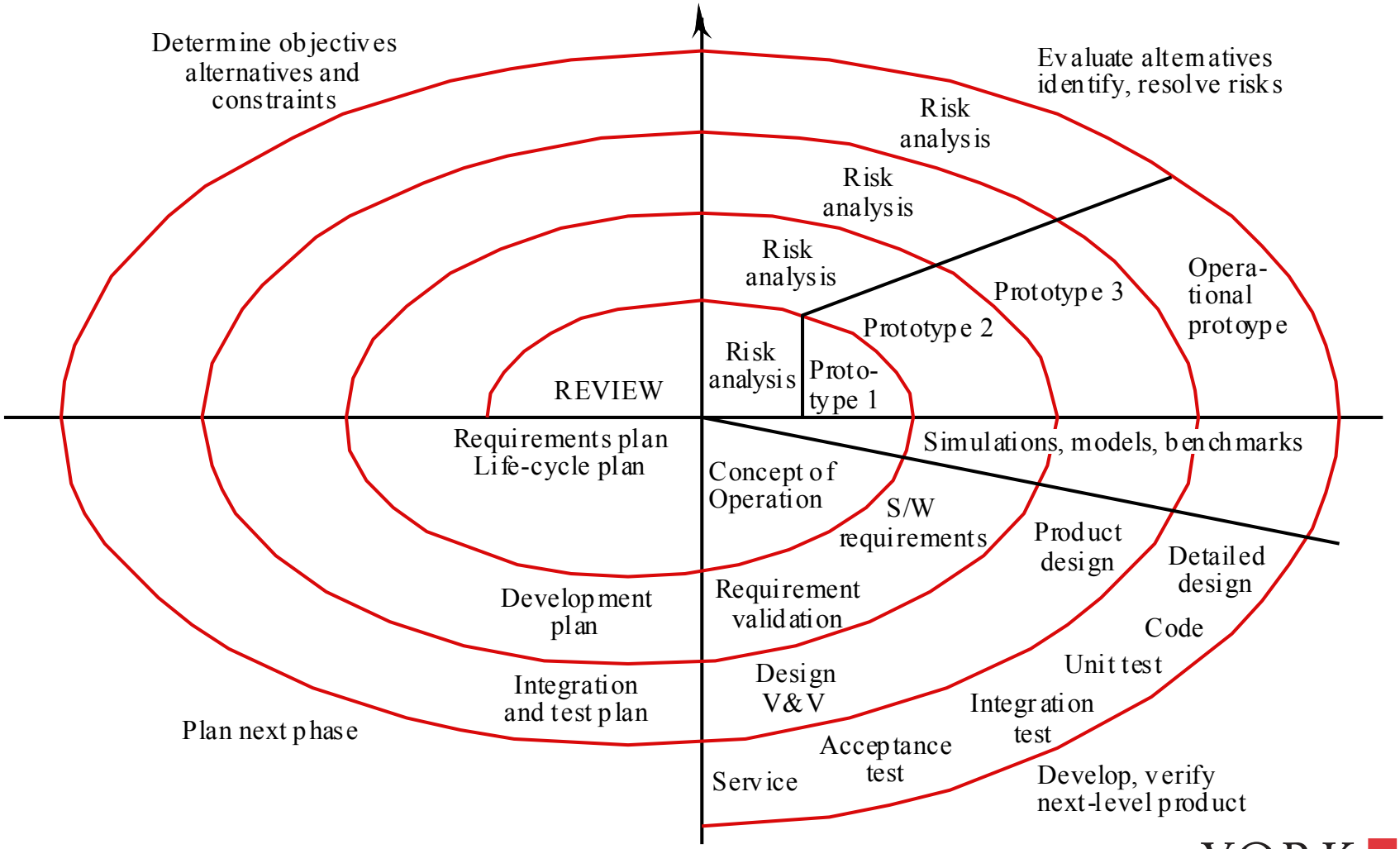
Process Model Problems

- Waterfall
  - High risk for new systems because of specification and design problems.
  - Low risk for well-understood developments using familiar technology.
- Prototyping
  - Low risk for new applications because specification and program stay in step.
  - High risk because of lack of process visibility.
- Transformational
  - High risk because of need for advanced technology and staff skills

Hybrid Process Models

- Large systems are usually made up of several sub-systems.
- The same process model need not be used for all subsystems.
- Prototyping for high-risk specifications.
- Waterfall model for well-understood developments.

Spiral Process Model



Spiral Model Advantages

- Focuses attention on reuse options.
- Focuses attention on early error elimination.
- Puts quality objectives up front.
- Integrates development and maintenance.
- Provides a framework for hardware/software development.

Spiral Model Problems

- Contractual development often specifies process model and deliverables in advance.
- Requires risk assessment expertise.

Process Visibility

- Software systems are intangible so managers need documents to assess progress.
- Waterfall model is still the most widely used model.

Waterfall Model Documents

Activity	Output documents
Requirements analysis	Feasibility study, Outline requirements
Requirements definition	Requirements document
System specification	Functional specification, Acceptance test plan Draft user manual
Architectural design	Architectural specification, System test plan
Interface design	Interface specification, Integration test plan
Detailed design	Design specification, Unit test plan
Coding	Program code
Unit testing	Unit test report
Module testing	Module test report
Integration testing	Integration test report, Final user manual
System testing	System test report
Acceptance testing	Final system plus documentation

Process Model Visibility

Process model	Process visibility
Waterfall model	Good visibility, each activity produces some deliverable
Evolutionary development	Poor visibility, uneconomic to produce documents during rapid iteration
Formal transformations	Good visibility, documents must be produced from each phase for the process to continue
Reuse-oriented development	Moderate visibility, it may be artificial to produce documents describing reuse and reusable components.
Spiral model	Good visibility, each segment and each ring of the spiral should produce some document.

Agile methods

- Dissatisfaction with the overheads involved in design methods led to the creation of agile methods. These methods:
  - Focus on the code rather than the design;
  - Are based on an iterative approach to software development;
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- Agile methods are probably best suited to small/ medium-sized business systems or PC products.

Principles of agile methods

Principle	Description
Customer involvement	The customer should be closely involved throughout the development process. Their role is provide and prioritise new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognised and exploited. The team should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and design the system so that it can accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

Problems with agile methods

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

Extreme Programming

- An agile development methodology
- Created by Kent Beck in the mid-90s
- A set of 12 key practices taken to their "extremes"

The XP slides are based on a slide set by Daniel Baranowski

What else is agile?

- Adaptive Software Development
- Crystal Methodologies
- Dynamic Systems Development Method
- Feature-Driven Development
- SCRUM
- And others

The 12 Practices

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-Hour Workweek
- On-site Customer
- Coding Standards

1 - The Planning Game

- Planning for the upcoming iteration
- Uses stories provided by the customer
- Technical persons determine schedules, estimates, costs, etc
- A result of collaboration between the customer and the developers

The Planning Game – Advantages

- Reduction in time wasted on useless features
- Greater customer appreciation of the cost of a feature
- Less guesswork in planning

The Planning Game – Disadvantages

- Customer availability
- Is planning this often necessary?

2- Small Releases

- Small in terms of functionality
- Less functionality means releases happen more frequently
- Support the planning game

Small Releases – Advantages

- Frequent feedback
- Tracking
- Reduce chance of overall project slippage

Small Releases – Disadvantages

- Not easy for all projects
- Not needed for all projects
- Versioning issues

3 – Metaphor

- The oral architecture of the system
- A common set of terminology

Metaphor – Advantages

- Encourages a common set of terms for the system
- Reduction of buzz words and jargon
- A quick and easy way to explain the system

Metaphor – Disadvantages

- Often the metaphor is the system
- Another opportunity for miscommunication
- The system is often not well understood as a metaphor

4 – Simple Design

- K.I.S.S.
- Do as little as needed, nothing more

Simple Design – Advantages

- Time is not wasted adding superfluous functionality
- Easier to understand what is going on
- Refactoring and collective ownership is made possible
- Helps keeps programmers on track

Simple Design – Disadvantages

- What is “simple?”
- Simple isn’t always best

5 – Testing

- Unit testing
- Test-first design
- All automated

Testing – Advantages

- Unit testing promote testing completeness
- Test-first gives developers a goal
- Automation gives a suite of regression test

Testing – Disadvantages

- Automated unit testing isn't for everything
- Reliance on unit testing isn't a good idea
- A test result is only as good as the test itself

6 – Refactoring

- Changing how the system does something but not what is done
- Improves the quality of the system in some way

Refactoring – Advantages

- Prompts developers to proactively improve the product as a whole
- Increases developer knowledge of the system

Refactoring – Disadvantages

- Not everyone is capable of refactoring
- Refactoring may not always be appropriate
- Would upfront design eliminate refactoring?

7 – Pair Programming

- Two Developers, One monitor, One Keyboard
- One “drives” and the other thinks
- Switch roles as needed

Pair Programming – Advantages

- Two heads are better than one
- Focus
- Two people are more likely to answer the following questions:
  - Is this whole approach going to work?
  - What are some test cases that may not work yet?
  - Is there a way to simplify this?

Pair Programming – Disadvantages

- http://www.cenqua.com/pairon/
- Many tasks really don't require two programmers
- A hard sell to the customers
- Not for everyone

8 – Collective Ownership

- The idea that all developers own all of the code
- Enables refactoring

Collective Ownership – Advantages

- Helps mitigate the loss of a team member leaving
- Promotes developers to take responsibility for the system as a whole rather than parts of the system

Collective Ownership - Disadvantages

- Loss of accountability
- Limitation to how much of a large system that an individual can practically “own”

9 – Continuous Integration

- New features and changes are worked into the system immediately
- Code is not worked on without being integrated for more than a day

Continuous Integration - Advantages

- Reduces to lengthy process
- Enables the Small Releases practice

Continuous Integration – Disadvantages

- The one day limit is not always practical
- Reduces the importance of a well-thought-out architecture

10 – 40-Hour Week

- The work week should be limited to 40 hours
- Regular overtime is a symptom of a problem and not a long term solution

40-Hour Week – Advantage

- Most developers lose effectiveness past 40-Hours
- Value is placed on the developers well-being
- Management is forced to find real solutions

40-Hour Week - Disadvantages

- 40-Hours is a magic number
- Some may like to work more than 40-Hours

11 – On-Site Customer

- Just like the title says!
- Acts to “steer” the project
- Gives quick and continuous feedback to the development team

On-Site Customer – Advantages

- Can give quick and knowledgeable answers to real development questions
- Makes sure that what is developed is what is needed
- Functionality is prioritized correctly

On-Site Customer – Disadvantages

- Difficult to get an On-Site Customer
- The On-Site customer that is given may not be fully knowledgeable
- May not have authority to make many decisions
- Loss of work to the customer's company

12 – Coding Standards

- All code should look the same
- It should not possible to determine who coded what based on the code itself

Coding Standards – Advantages

- Reduces the amount of time developers spend reformatting other peoples' code
- Reduces the need for internal commenting
- Call for clear, unambiguous code

Coding Standards – Disadvantages

- Degrading the quality of inline documentation

XP – Advantages

- Built-In Quality
- Overall Simplicity
- Programmer Power
- Customer Power
- Synergy Between Practices

XP – Disadvantages

- Informal, little, or no documentation
- Scalability
- Contract Issues
- Misconception on the cost of change
- Tailoring

Application – Advantageous

- Highly uncertain environments
- Internal projects
- Joint ventures

Application – Disadvantageous

- Large, complex environments
- Safety critical situations
- Well understood requirements
- Distant or unavailable customer

Web resources

- www.junit.org
- www.xprogramming.com
- www.extremeprogramming.org
- www.refactoring.com
- www.pairprogramming.com

Our project

- Fully develop a system that translates guitar tablature from ASCII to PDF
- We will call it TAB2PDF
- We will use an extreme programming approach whenever possible

ASCII Tablature

```
TITLE=Moonlight Sonata
SUBTITLE=Ludwig van Beethoven
SPACING=5

|--------------------------|--------------------------|
|-----1-----1-----1-----1-|-----1-----1-----1-----1-|
|---2-----2-----2-----2---|---2-----2-----2-----2---|
|-2-----2-----2-----2-----|-2-----2-----2-----2-----|
|-0-----------------------|--------------------------|
|--------------------------|-3-----------------------|

|--------------------------|--------------------------|
|-----1-----1-----3-----3-|-----3-----1-----0-----0-|
|---2-----2-----3-----3---|---1-----2-----2-----1---|
|-3-----3-----3-----3-----|-2-----2-----2-----0-----|
|-------------5------------|--------------------------|
|-1-----------------------|-0-----------0------------|
```

PDF Tablature

To get started

- Look at the sample input and output files posted on the course website
- Download the iText library for dynamically creating PDF files
  - http://itextpdf.com
- Attempt to create a Hello World PDF file
- Lots of examples at the site above

Intentionally vague requirements

- In a real software development project, requirements are vague and ever-changing
- The exact requirements will be refined iteratively by meeting with the “customer” on a weekly basis

Teams

- Teams are assigned randomly by the “manager”
- As enrollment in the course changes in the first few weeks, the “manager” will rearrange the teams
- Same as a real software project!

Team 1

- Chayka, Dmytro
- D'Costa, Ryan
- Mangal, Alistair
- Pilay, Jorge

Team A

- Carr, Matt
- Choi, Youn
- Dutta, Dev
- Vieira Leite, Guilherme

Workload

- This course requires 8-10 hours per week per student
- Have to start working immediately
- In the second hour of each lecture, each team will present their progress to the instructor and receive feedback
  - “Customer” on site!

Evaluation

- 15% - Midterm prototype + Presentation (due Mar 6)
- 10% - Requirements Document (due Apr 8)
- 10% - Design Document (due Apr 8)
- 10% - Testing Document (due Apr 8)
- 5% - User Manual (due Apr 8)
- 20% - Evaluation of final code (due Apr 8)
- 20% - Final presentation (due Apr 3)
- 10% - Participation (in class and at weekly presentations)