UNIVERSITÉ BISHOP'S UNIVERSITY

CS 410 / CS 560: Software Engineering

Software Life Cycle

Dr. Mohammed Ayoub Alaoui Mhamdi
Bishop’s University
Sherbrooke, Qc, Canada
malaoui@ubishops.ca

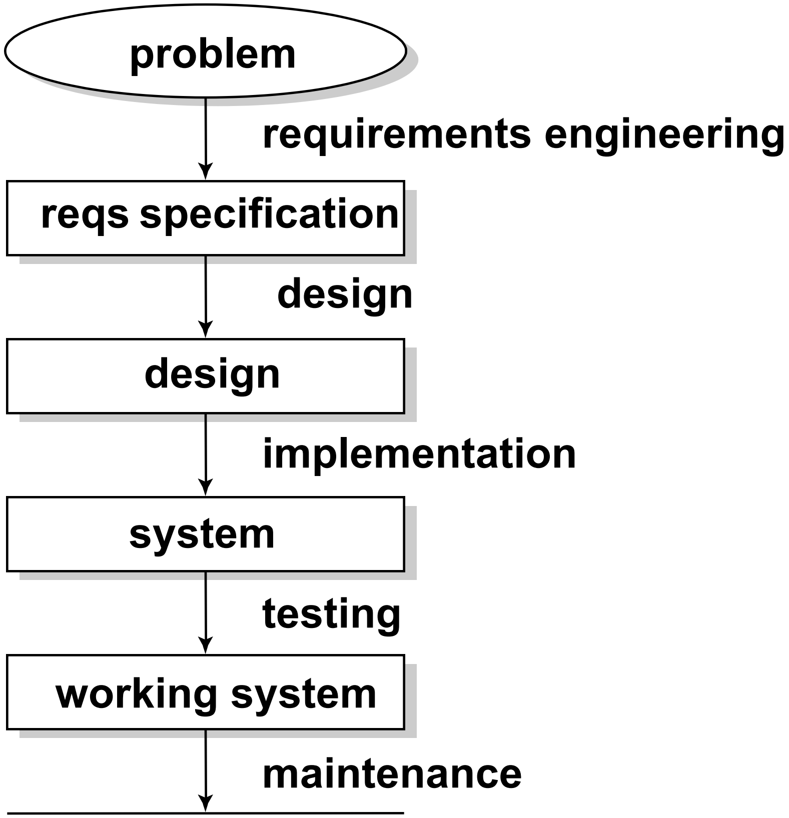
Main issues

- Discussion of different life cycle models
- Maintenance or evolution

Introduction

- software development projects are large and complex
- a phased approach to control it is necessary
- traditional models are document-driven: there is a new pile of paper after each phase is completed
  - They assume that software development proceeds in an orderly, sequential manner,
  - The pile of paper that is produced in the course of the project guides the development process
- evolutionary models recognize that much of what is called maintenance is inevitable
- latest fashion: agile methods, eXtreme

Simple life cycle model



Simple life cycle model

- software development projects are large and complex
- a phased approach to control it is necessary
- traditional models are document-driven: there is a new pile of paper after each phase is completed
  - They assume that software development proceeds in an orderly, sequential manner.
- evolutionary models recognize that much of what is called maintenance is inevitable
- latest fashion: agile methods, eXtreme Programming

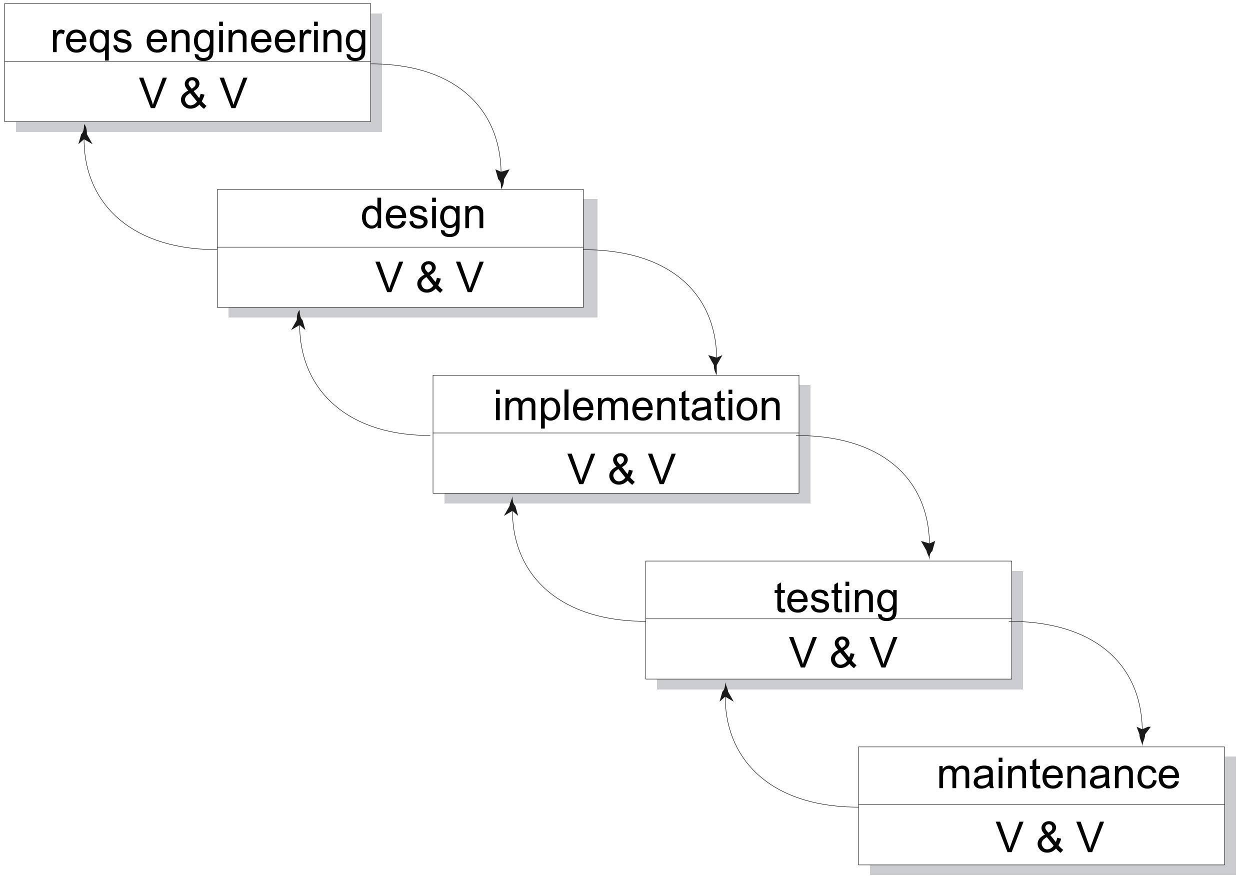
Point to ponder #1

- Why does the model look like this?
- Is this how we go about?

Simple Life Cycle Model

- document driven, planning driven, heavyweight
- milestones are reached if the appropriate documentation is delivered (e.g., requirements specification, design specification, program, test document)
- much planning upfront, often heavy contracts are signed
- problems
  - feedback is not taken into account
  - maintenance does not imply evolution

Waterfall Model


reqs engineering

V & V

design

V & V

implementation

V & V

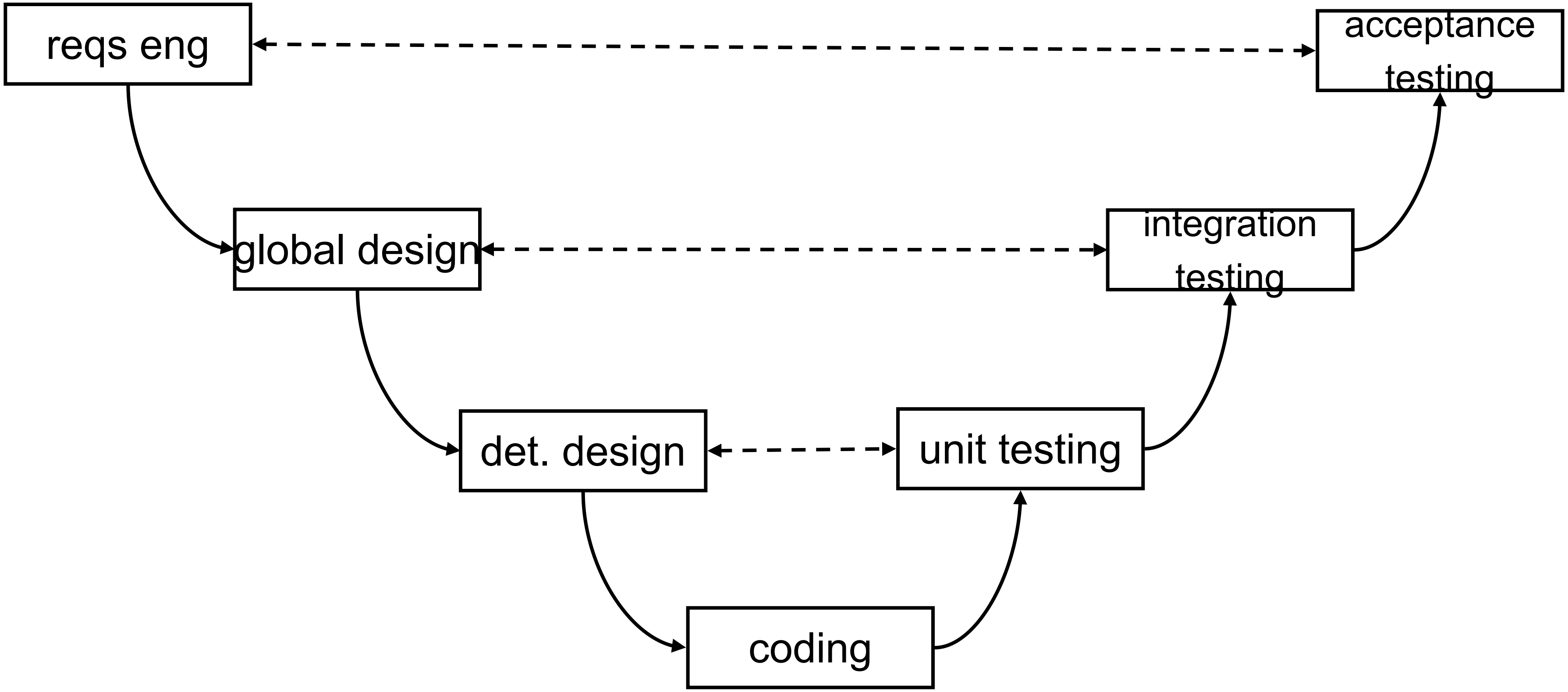
testing

V & V

maintenance

V & V

V-Model


reqs eng
acceptance testing
global design
integration testing
det. design
unit testing
coding

Waterfall Model

- includes iteration and feedback
- validation (*are we building the right system*?) and verification (*are we building the system right*?) after each step
- user requirements are fixed as early as possible
- problems
  - too rigid
  - developers cannot move between various abstraction levels

Activity versus phase

Phase *Activity*	Design	Implementation	Integration testing	Acceptance testing
Integration testing	**4.7**	**43.4**	**26.1**	**25.8**
Implementation (& unit testing)	**6.9**	**70.3**	**15.9**	**6.9**
Design	**49.2**	**34.1**	**10.3**	**6.4**

Lightweight (agile) approaches

- prototyping
- incremental development
- XP

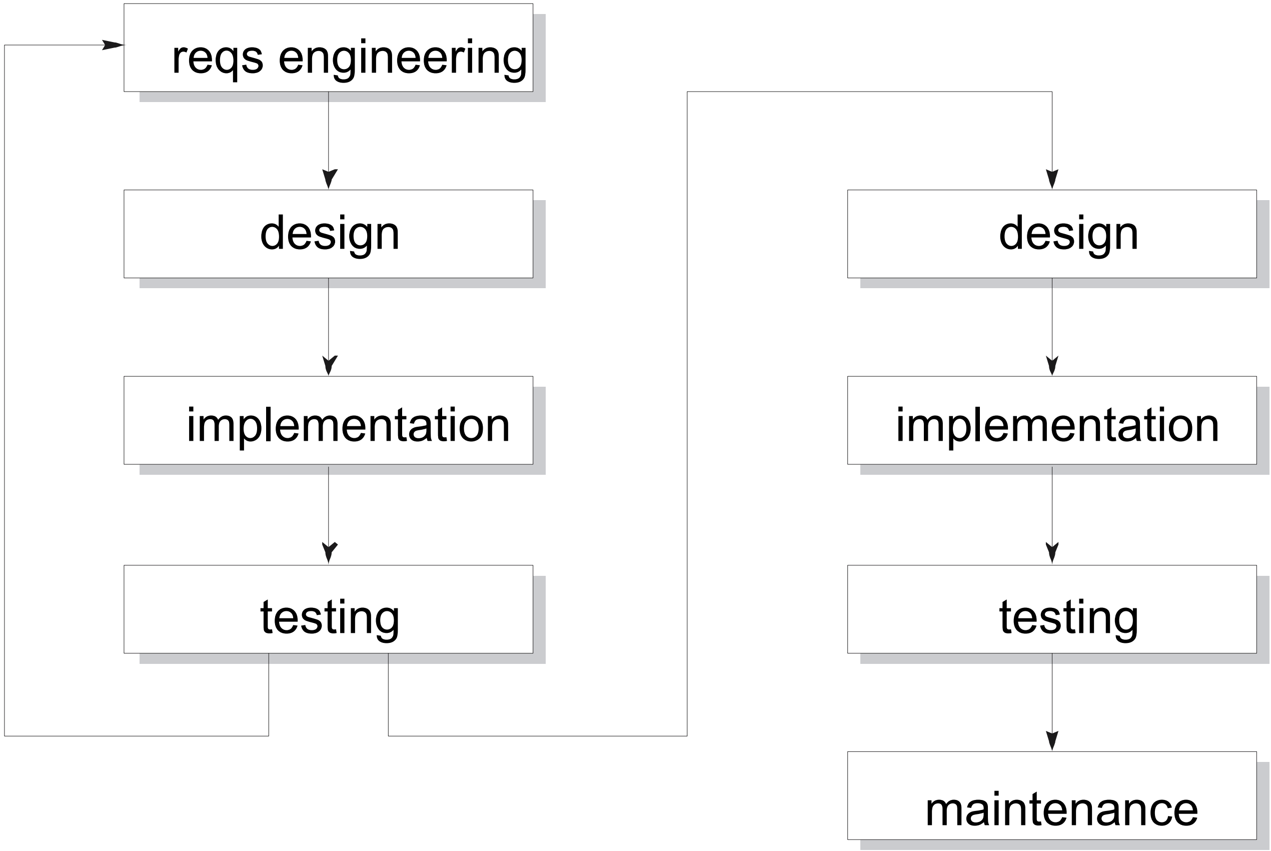
The Agile Manifesto

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

Prototyping

- requirements elicitation is difficult
  - software is developed because the present situation is unsatisfactory
  - however, the desirable new situation is as yet unknown
- prototyping is used to obtain the requirements of some aspects of the system
- prototyping should be a relatively cheap process
  - use rapid prototyping languages and tools
  - not all functionality needs to be implemented
  - production quality is not required

Prototyping as a tool for requirements engineering



Prototyping

- *throwaway prototyping*: the n-th prototype is followed by a waterfall-like process (as depicted on previous slide)
  - creation of a model that will eventually be discarded rather than becoming part of the final delivered software.

- *evolutionary prototyping*: the nth prototype is delivered
  - is a lifecycle model in which the system is developed in increments so that it can readily be modified in response to end-user and

Point to ponder #2

What are the pros and cons of the two approaches?

Prototyping, advantages

- The resulting system is easier to use
- User needs are better accommodated
- The resulting system has fewer features
- Problems are detected earlier
- The design is of higher quality
- The resulting system is easier to maintain
- The development incurs less effort

Prototyping, disadvantages

- The resulting system has less features
- The performance of the resulting system is worse
- The design is of less quality
- The resulting system is harder to maintain
- The prototyping approach requires more experienced team members

Prototyping, recommendations

- the users and the designers must be well aware of the issues and the pitfalls
- use prototyping when the requirements are unclear
- prototyping needs to be planned and controlled as well

Incremental Development

- a software system is delivered in small increments, thereby avoiding the Big Bang effect
- the waterfall model is employed in each phase
- the user is closely involved in directing the next steps
- incremental development prevents overfunctionality

XP - eXtreme Programming

- Everything is done in small steps
- The system always compiles, always runs
- Client as the center of development team
- Developers have same responsibility w.r.t. software and methodology

13 practices of XP

- Whole team: client part of the team
- Metaphor: common analogy for the system
- The planning game, based on user stories
- Simple design
- Small releases (e.g. 2 weeks)
- Customer tests
- Pair programming
- Test-driven development: tests developed first
- Design improvement (refactoring)
- Collective code ownership
- Continuous integration: system always runs
- Sustainable pace: no overtime
- Coding standards

Maintenance or Evolution

- some observations
  - systems are not built from scratch
  - there is time pressure on maintenance

- the five laws of software evolution
  - law of continuing change
  - law of increasingly complexity
  - law of program evolution
  - law of invariant work rate
  - law of incremental growth limit

Purposes of process modeling

- facilitates understanding and communication by providing a shared view of the process

- supports management and improvement; it can be used to assign tasks, track progress, and identify trouble spots

- serves as a basis for automated support (usually not fully automatic)

Caveats of process modeling

- not all aspects of software development can be caught in an algorithm
- a model is a model, thus a simplification of reality
- progression of stages differs from what is actually done
- some processes (e.g. learning the domain) tend to be ignored
- no support for transfer across projects

Summary

- Traditional models focus on *control* of the process
- There is no one-size-fits-all model; each situation requires its own approach
- A pure project approach inhibits reuse and maintenance
- There has been quite some attention for process modeling, and tools based on such process models