cis20.2
design and implementation of software applications II
spring 2008
session # IV.1
software engineering and development processes

topics:

- software engineering: review and overview
- software lifecycle: processes and models
- software development: phases, issues and strategies

software engineering

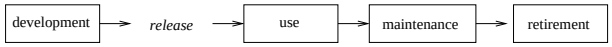
- in school, computer science teaches how to write ideal software
- in the real world, software is usually late, overbudget and broken
- on average, software lasts much longer in the real world than either hardware or employees
- the real world is a harsh environment, and software is fundamentally brittle
- remember the real-world examples from last term (ariane-501, therac-25)

overview of software engineering

- Stephen Schach: "Software engineering is a discipline whose aim is the production of fault-free software, delivered on time and within budget, that satisfies the user's needs."
- includes:
  - requirements analysis
  - human factors
  - functional specification
  - software architecture
  - design methods
  - programming for reliability
  - programming for maintainability
  - team programming methods
  - testing methods
  - configuration management

software lifecycles

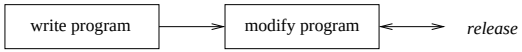
- software is not a build-one-and-throw-away process
- that's far too expensive
- we need to implement a *process* so that software is maintained correctly
- this is called the *software life cycle*:



- there are several *process models*:
  - *build-and-fix model*
  - *waterfall model*
  - *iterative model*
  - *evolutionary model*

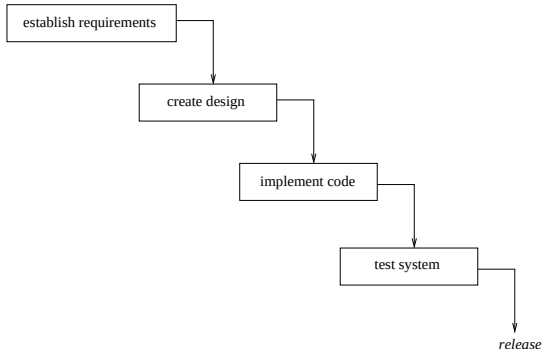
build-and-fix model

- the oldest model
- probably what you've been doing when you write your homework...



waterfall model

- developed as software evolved into large projects, involving many lines of code, many files and many programmers working together on the same large project...

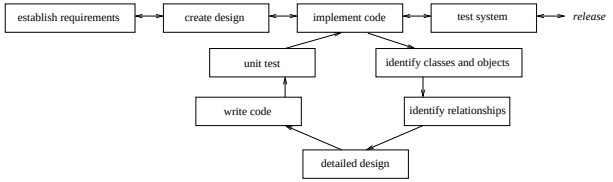


iterative model

- developed after it was recognized that the waterfall model was unrealistic
- each step can be (and usually is) revisited
- especially common in large companies, where multiple people are working on the same project and the people who, for example, "establish requirements" are not the same people who "create design" or "implement code" or "test system"

evolutionary model

- evolved, again from companies where large software projects are developed and maintained, particularly after the introduction of the "object-oriented" way of thinking
- emphasizes *modularity* and allows for software re-use as well as testing of individual modules to make sure that each piece is robust and correct before it is added to the whole



software development phases

- 7 basic phases (Schach):
  - requirements (2%)
  - specification/analysis (5%)
  - design (6%)
  - implementation (module coding and testing) (12%)
  - integration (8%)
  - maintenance (67%)
  - retirement
- percentages in ()'s are average cost of each task during 1976-1981
- testing and documention should occur throughout each phase
- note which is the most expensive!

software development issues

- code re-use
- portability
- interoperability
- scalability

code re-use

- libraries
- APIs
- system calls
- objects
- frameworks
- design patterns

software portability

- portability pitfalls
  - hardware differences
  - operating system differences
  - number storing/handling differences
  - compiler differences
  - library differences
- language portability
  - Java
    - meant to be portable using a JVM (Java Virtual Machine)
    - write once, run anywhere (sorta... kinda...)
  - C#
    - also uses a "JVM"
    - emphasizes mobile data, not mobile code
    - uses XML everywhere

interoperability

- service/web service models
  - CORBA = Common Object Request Broker Architecture
    `http://www.omg.org/gettingstarted/specintro.htm`
  - SOAP = Simple Ojbect Access Protocol
    `http://www.w3schools.com/soap/default.asp`
  - EJB = Enterprise JavaBeans technology
    `http://java.sun.com/products/ejb/`
- define abstract services
- allow programs in any language to access services in any language in any location
- data objects (rather than code objects)

scalability

- familiarity with patterns can help
- no need to scale beyond capabilities of the machine
- avoid unnecessary barriers
- handle overloading gracefully
- scale from single connection to forking processes, to threads, to thread pool...