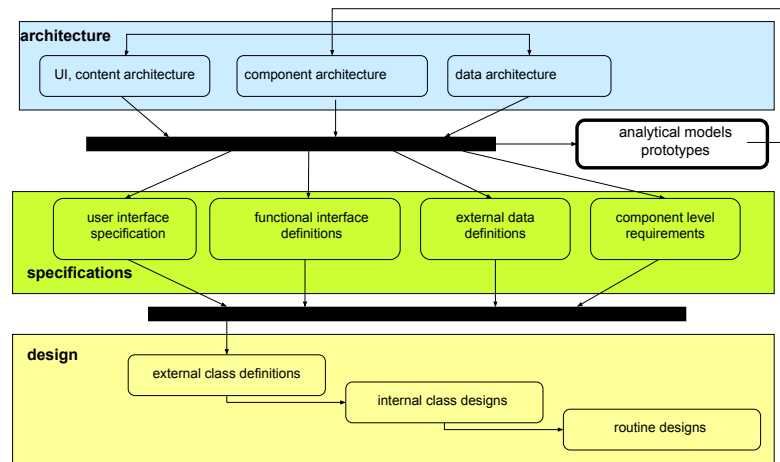


System Modeling

- General principles of modeling
- Descriptive models
- Analytical models
- Prototypes and Proofs of Concept
- Prototyping Exercise

Model Hierarchy/Succession



General Types of Models

- Descriptive models
 - built to facilitate communication
 - help users understand what will be built
 - help developers understand what to build
- Analytical Models
 - built to answer questions or reduce doubt
 - clarification and validation of requirements
 - questions about a proposed implementation
- Mock-up, Prototype, Proof-of-Concept
 - built to clarify, evaluate or sell an idea
 - convince someone that the idea will work

Models

- are smaller and simpler than the real thing
 - making them less expensive to build
 - making them more portable
 - making them easier to understand
- often model only subsets of whole system
 - stripping away layers complicating details
 - permitting system to be understood in parts
- may expose otherwise invisible processes
 - making those processes easier to understand
- are only approximations of reality
 - they may lead to inaccurate notions/predictions

Agile Modeling Principles

- Model with a purpose
 - be clear why you are building each model
- Travel light
 - maintain as few models as possible
 - know which (few) models are keepers
- Use multiple models
 - don't try to make one model serve all needs
- Content is more important than format
 - best form is the one that achieves your goals

System Views

- Complex systems are hard to comprehend
 - more information than the mind can hold
- We have to decompose them
 - into hierarchies of systems and subsystems
 - PC = { case, power supply, motherboard, devices }
 - motherboard = { processor, bus, memory, support chips }
 - bus = { addressing, data transport, arbitration }
 - into (seemingly) independent systems
 - skeletal, muscular, circulatory, neural, digestive ...
 - into components on orthogonal axes
 - pragmatic, moral, legal, aesthetic, political, ...
- A system modeling language must be able to
 - describe hierarchies of models
 - describe different types (and aspects) of models

Design Model-Based Tools

- Design Visualization Tools
 - browse through hierarchies of models
 - selecting views
 - filtering displayed contents
- Design Validation Tools
 - style & standards conformance checkers
 - consistency checkers
 - interface based test case generators

Analytical Models and Prototypes

7

Simple Mathematical Models (just do it)

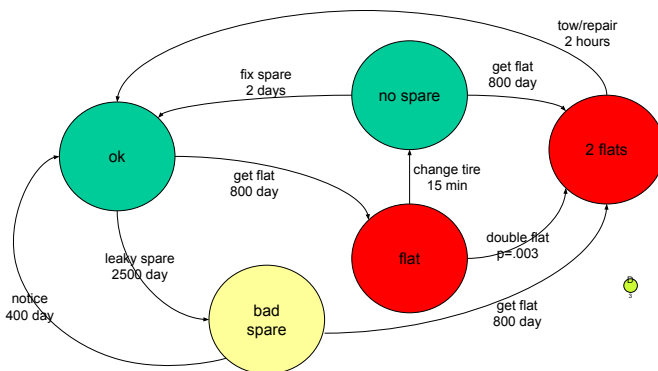
- Algebraic Models
 - make up equations to describe situations
 - don't be afraid to estimate parameter values
 - use ranges and independent estimates
- Probabilistic Models
 - estimate event likelihood, frequencies
 - outcome expectancies (*cost x probability*)
- Combinatoric Models
 - how many possible combinations are there

Analytical Models and Prototypes

8

Markov Availability Models

- Mathematical models of state transitions



Analytical Models and Prototypes

9

(Markov Models) (there are nice tools for these)

- Given:
 - a state machine representation of a system w/mean transition rates (and/or probabilities)
- Assuming:
 - all events follow *standard* distribution
 - transitions have no memory of past events
- Model will yield numeric solutions for:
 - % of time system will spend in each state
 - mean visit duration for each state

Analytical Models and Prototypes

10

Prototype to reduce Risk

- User Interface Prototypes
 - do we have the U/I requirements right?
 - will the users find it to be usable?
- Mechanism or Process Prototypes
 - do we know how to build/do this?
 - how well will it work?
- Tool and Platform Evaluation
 - how much trouble will this new stuff cause?
- Be clear on what risk you are addressing

Analytical Models and Prototypes

11

Keep your prototypes lean

- minimum possible implementation
 - simplest model that answers the question
 - fake everything you possibly can
- don't fall in love with it
 - plan on throwing it away when you're done
 - don't yield to the temptation to polish it
- But don't be afraid to make mistakes
 - prototypes are the place to take chances
 - but don't hesitate to abandon bad ideas

Analytical Models and Prototypes

12

Proof of Concept

- You need support from others to succeed
 - managers, investors, customers
- They may be afraid to help you
 - if you fail, they will suffer losses too
 - they may doubt your ability to succeed
 - the proposed product might not be compelling
 - you might not be able to deliver it on schedule
 - there may be unsolvable technical problems
 - these doubts must be assuaged
- well designed proof-of-concept can do it

E₁

Exercise – in your teams

- consider user facing interfaces (5-7 min)
 - where understanding/usability might be an issue?
 - how would you mock-up what?
 - who would you ask to do what with it?
- consider tools and mechanisms (5-7 min)
 - what might be hard to master, or might not work?
 - what do they impose on your architecture?
 - what things do you need to be sure of?
 - what is the simplest project to get the answers?
- be prepared to discuss what you discover

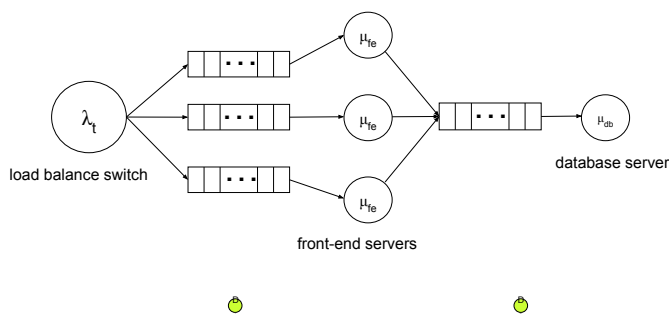
For the next lecture

- McConnell 3.5-3.6, 5.2
 - “goodness” in architecture and design
- Foote: Big Ball of Mud (digest)
 - the forces of anti-architecture
- SEI: Definition of Architecture
- Garlan: S/W Architecture
 - Elements of S/W Architecture
- Wikipedia: Mechanism/Policy Separation
- Kampe: Interface Stability
- Kampe: Software Testability

Supplementary Slides

Queuing Models*

- Mathematical models of traffic and servers



(Queuing Models) (don't try these at home)

- Given:
 - a system of input queues and servers
 - request arrival and processing rates
- Assuming:
 - arrivals have a *standard* distribution
 - processing time is same for all events in queue
- Model will yield closed-form solutions for:
 - queue length (distribution function)
 - waiting time (distribution function)

Discrete Event Simulations

- simulate dynamic system behavior
 - for systems other techniques can't model
 - e.g. future events depend on past details
 - for questions other techniques can't answer
 - e.g. "Why are there so many cache misses?"
 - to exercise scheduling/routing algorithms
 - run simulated traffic through a real algorithm
- there are very abstract models
 - may be written in special simulation language
 - code may be very different from real system

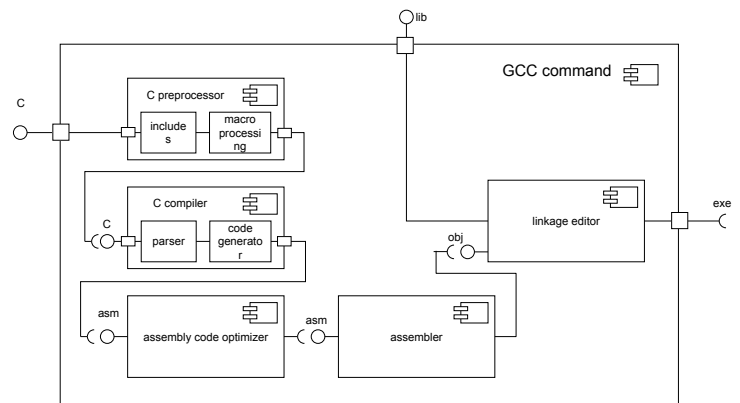
Automatic Code Generation

- class model compilers
 - generate module skeletons
 - declarations for classes, methods, properties
 - stubbed routines to permit basic build/test
- rough code generators
 - simple code from activity diagrams
 - calls from interaction diagrams
- state language compilers
 - generate final code from state machines

UML General Conventions

- Squares represent logical things
 - classes, objects, packages, components
 - box ornaments determine the type of thing
 - name of thing appears at top, inside the square
- Arrows represent relationships
 - communication solid, dependency dashed
 - arrow heads determine type and direction
 - relationships (and end connectors) can be named
- Circles represent interfaces
- they can be named
- 3D boxes represent physical containers
- UML diagrams can be nested and composed

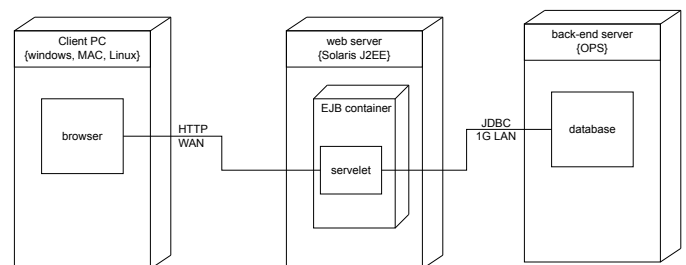
Component Models*



(UML Component Models)

- Most UML models are detail oriented
 - class, activity & interaction diagrams
- Component models take a black box view
 - system is composed of multiple black boxes
 - they are interconnected with one-another
 - some connectors support standard interfaces
- Component models focus entirely on
 - the independent components
 - their interconnections and interfaces
- Well suited for high level architecture

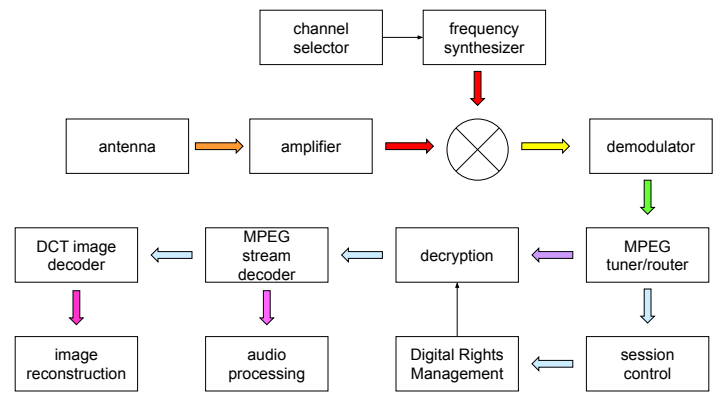
Component Deployment Models*



(Component Deployment Models)

- Most UML structural models are logical
 - classes and software components
- We must also model physical systems
 - hardware components
 - physical interconnections between them
- And the deployment of logical functionality
 - which software runs on which hardware
 - distributed and client/server applications
 - which software runs in which container
 - application servers, virtual machines, etc.

Digital TV Data Flow Model*



(Data Flow Models)

- UML is designed for OO-software
 - components are comprised of related objects
 - objects have properties and methods
 - methods have associated algorithms
 - interactions are via discrete messages
- Data Flow Models take different view
 - follow input, through processing, to output
 - all components are processing in parallel
 - processing is continuous rather than discrete
 - view system in terms of data transformations