System Modeling

- Classes and levels of system models
- General principles of modeling
- Descriptive models
  - UML – purpose and conventions
  - UML component & deployment diagrams
- Analytical models
  - queuing models
  - Markov models
  - discrete event simulations
- Prototyping projects

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 predictions

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-ups and "Proof-of-Concept"s
  - built to sell an idea
    - convince someone that the idea will work

General 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
  - a good form is 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 relatively 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

Universal Modeling Language

- a family of related graphical notations
  - for representing software system designs
  - particularly those built in object oriented style
- supports a wide range of uses
  - a design sketching language
  - a system specification language
  - a programming language
- also has a textual (XML) representation
  - enabling development of CAD tools

Design Model Based Tools

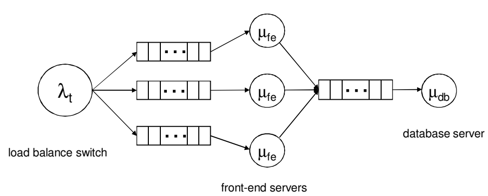
- Design Visualization Tools
  - browse through hierarchies of models
  - selecting views
  - filtering displayed contents
- Design Validation Tools
  - style & standards conformance checkers
  - consistency checkers
  - interface based test case generators

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

Queuing Models*

- Mathematical models of traffic and servers

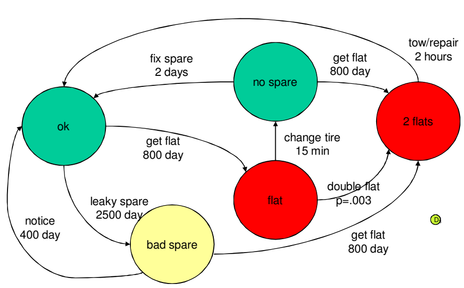


(Queuing Models) (don't try these at home)

- 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)

Markov Availability Models*

- Mathematical models of state transitions



(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

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

Prototype to reduce Risk

- User Interface Prototypes
  - Do we have the U/I requirements right?
- Mechanism Prototypes
  - Do we know how to build this?
  - How well will it work?
- Process Validation
  - Do we really know how to process this?
- Tool and Platform Evaluation
  - How much trouble will this new stuff cause?

Proof of Concept

- You need help 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 development problems
  - these doubts must be assuaged
- well designed proof-of-concept can do it

For Next Lecture (design reviews)

- McConnell 20.2-5
  - why design reviews are so important
- McConnell 21.3-4
  - description and discussion of methodology
- Wiegers: Inspections
  - principles and detailed techniques
- Wiegers: Seven Deadly Sins
  - how reviews most commonly go wrong

Supplementary Slides

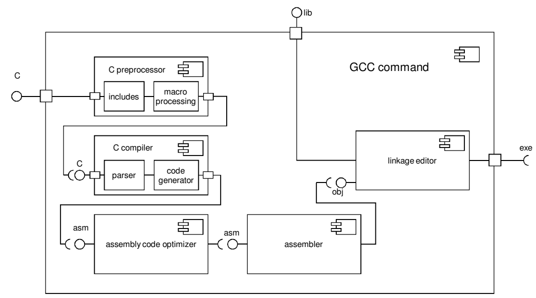
Automatic Code Generation

- 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

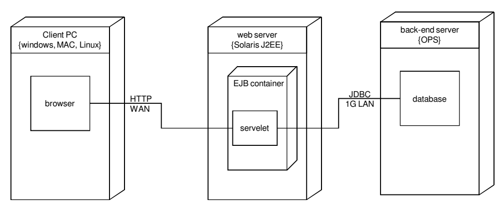
Component Models*



(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.