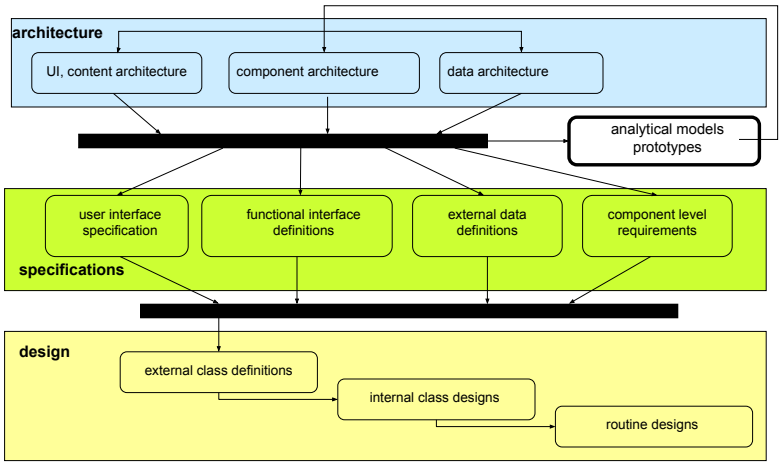
System Modeling

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

Model Hierarchy/Succession



Models

- are smaller and simpler than the real thing
  - making them less expensive to build/change
  - 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

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 (you?) that the idea will work

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 descriptions 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

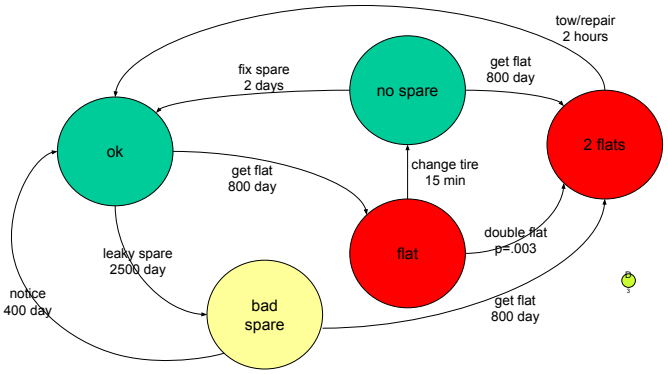
- Design Visualization Tools
  - browse through hierarchies of models
  - selecting views
  - filtering displayed contents
- Design Validation Tools
  - style & standards conformance checkers
  - consistency checkers
  - interface based test case generators

Simple Mathematical Models (just do it, back-of-the-envelope)

- 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

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

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

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

Proof of Concept

- You need support to succeed
  - managers, investors, customers, your self
- 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

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

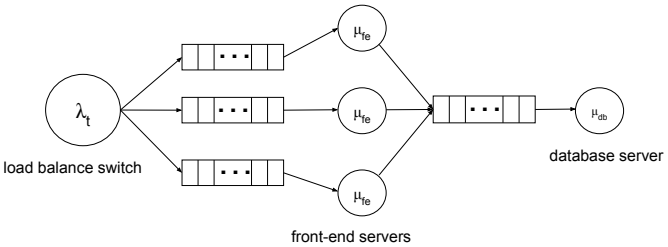
For the next lecture

- 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

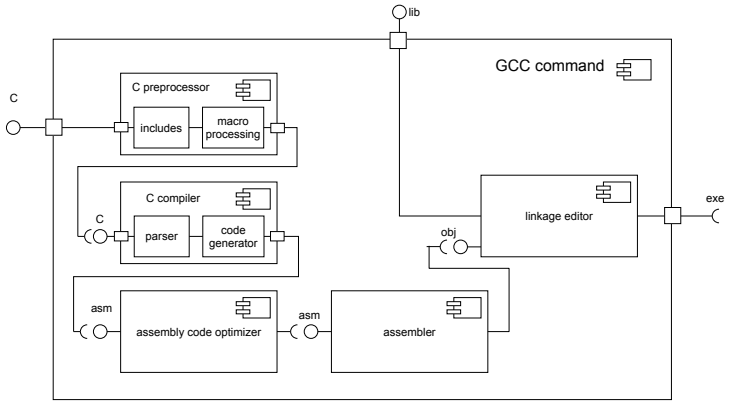
Automatic Code Generation

- 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

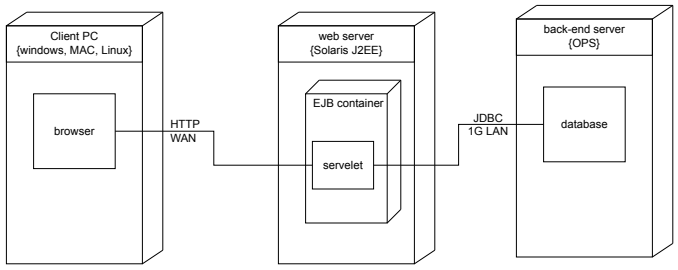
Component Models*



(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

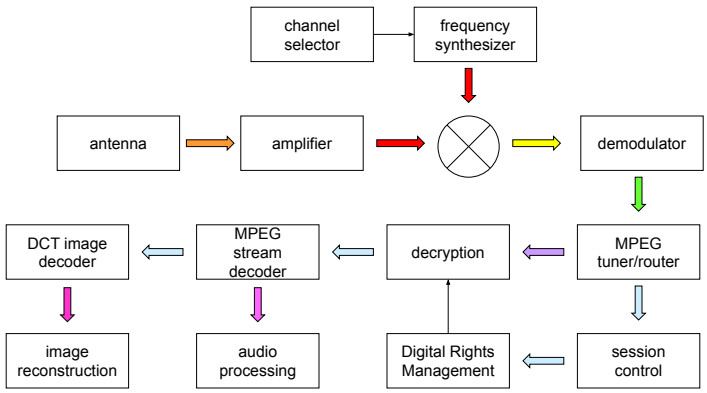
Component Deployment Models*



(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

Discussion Slides

Deeper A_1

- What is the difference between a descriptive model for a user and a descriptive model for a developer?

  User models describe the system's external appearance and function. Developer models describe internal structure and operation.

Deeper A_2

- Give examples of requirements questions that could be answered by analytical modeling?

  Task structure and user interface questions are often best answered by prototype usability questions.

  Estimates of required performance can often be modeled by simple mathematical modeling.

Deeper

- What is the difference between a product mock-up and an engineering proof of concept?
  - *The purpose of a product mock-up is to show potential supporters/customers what the product would look like.*
  - *The purpose of an engineering proof-of-concept is to demonstrate that we have a viable solution to what has here-to-fore been considered to be a hard problem.*

Deeper

- Give examples of a consumer product or service mock-up.
  - *A hand-made prototype of a new personal music player.*
  - *Simulated content and interactions with a new web service.*
  - *A Tivo-like product implemented on a work-station rather than in a custom appliance.*

Deeper

- What is the wisdom behind modeling with a purpose?
  - *If you are very clear about your goals:*
    - *it is more likely that they have been well refined.*
    - *you are likely to chart a more direct path to them.*
    - *you are less likely to be distracted by tangential issues.*
    - *you will be more likely to recognize whether or not you have achieved them.*
  - *If you're modeling without clear goals, you may spend lots of time building the wrong things.*

Deeper

- Examples of descriptive model purpose?
  - *Management … which pieces need to be built*
  - *Development … what each component does*
  - *Support … how components will be deployed*
  - *Training … how to use the system*
- Examples of analytical model purpose?
  - *Management … to size a task*
  - *Development … to validate an approach*
  - *Support … how system must be configured*

Deeper

- What is the wisdom behind traveling light?
  - *Ongoing maintenance is a killer, and models are no exception. It takes work to keep models up-to-date with changing requirements and design.*
  - *The fewer things you decide to maintain, the less time you will spend maintaining them.*
  - *Discarding a model after it has served its purpose is a good thing.*
  - *Relatively few models are worth keeping.*

Deeper

- Traveling light for descriptive models?
  - *Figure out which design models will be the most useful, and then build only those.*
  - *Understand which models will be superceded by the code, and which (few) will be kept for training purposes.*
- Traveling light for analytical models?
  - *Precisely understand the key question, and model only what is necessary to answer it.*
  - *Understand which few models will have ongoing value.*

Deeper B_5

- What is the wisdom behind using multiple models?
  - *This is where "one size fits all" meets "divide and conquer".*
  - *Smaller and simpler models that attempt to directly address particular problems will generally do so more effectively.*
  - *Moreover, it will often be the case that N smaller models can be built much more quickly and easily than a single super-model to address the same issues.*

Deeper B_6

- Interpret multiple for descriptive models?
  - *Use multiple types of models (behavioral, structural) to illustrate different aspects of the system.*
  - *Separate models for independent subsystems will be simpler and easier to understand.*
- Interpret multiple for analytical models?
  - *Don't try to get one model to answer all your questions. Build different kinds of models to answer different kinds of questions.*

Deeper B_7

- What is the wisdom behind putting content over representation?
  - *Standard representations are seldom requirements to be imposed on projects. They are tools that have been developed to help solve particular problems.*
  - *If a standard representation doesn't quite meet your needs, or if it seems to heavy for your problem, you are usually free to improvise a more appropriate representation.*

Deeper B_8

- What does content over form mean for descriptive models?
  - *Don't worry about strict adherence to a particular modeling language. Represent your system in any way that works.*
- What does content over form mean for analytical models?
  - *Don't feel obligated to use a more formal or powerful modeling technique if a simpler one will (reliably) give you the answer you need.*

Deeper D_3

- What can we do if a Markov model tells us we will spend too much of our time in an unavailable state?
  1. *speed up the repair time for that state.*
  2. *reduce the number of incidents that lead to that state.*
  3. *change the design so that the system can continue to provide service while in that state.*

Deeper D_4

- Why might we prefer to test a component like a scheduler in a simulator (vs. in a real system)?
  - *A real system might not be available (yet).*
  - *A simulator can force decisions through the scheduler many orders of magnitude faster than a real system would, enabling us to collect data against a wider range of traffic patterns in a shorter period of time.*

Deeper D_5

- Why might random traffic be better than sampled traffic?
  One sample may not be representative of all likely usage scenarios. Random traffic may provide fuller exercise.
- Why might captured traffic be better than randomly generated traffic?
  Real traffic isn't random. There are patterns to requests, and better algorithms often exploit knowledge of these patterns.

Deeper E_1

- How do you decide what your "proof of concept" prototype should do?
  Whom do you need to convince?
  What doubts that need to be assuaged?
  What demonstration would clearly assuage them?