Architecture Definition

- A "software architecture" is the structure (or structures) of a system, which comprise:
  - software components,
  - the externally visible properties of those components,
  - and the relationships among them.
- This is a pretty soft definition
  - I don't think there is any consensus on how to describe an architecture formally.
- An architecture is primarily an artifact used to communicate the overarching structure of a system.
  - Before, during and after the construction of a system.
  - Between various stakeholders.

Components & Structures

- Architecture defines "components"
  - an abstraction
  - suppresses details not pertinent to its interactions with other components
- An architecture comprises more than one structure
  - modular structure (calls/uses)
  - process structure (invokes, communicates with, synchronises with)
  - physical structure (libraries, DLL's, processors)
  - …

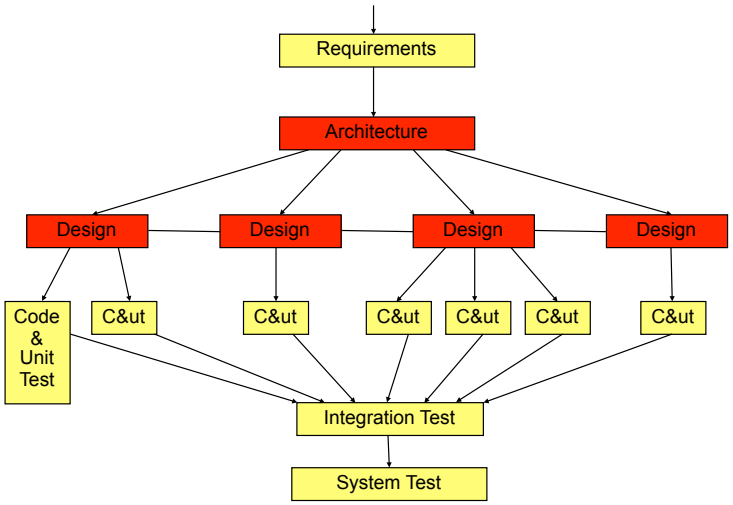
In Practice

- Divide into two levels:
  - System-Level ***Architecture***
  - Programming-Level ***Design***

[User Interface
  - Sometimes also referred to as "design" (or even "architecture")
  - Different topic. Not covered in this course.

]

Design & Architecture in the Development Process



Software Architecture

- Specifying at the highest level the construction of the system:
  - Technology choices
    - Platforms, language, database, middleware, …
  - System construction
    - Overall pattern: Monolithic, RDBMS, client/server, 3-tiered, n-tiered, distributed, …
    - Hardware interfaces (if any)
  - Division into programs
    - E.g. a program for data entry, another for data analysis, a Web-oriented interface, …
  - Division of programs into major subsystems
    - Reuse strategy (shared subsystems)
    - Major strategies (e.g., for persistence, IPC, …)

Software Design

- Mostly about structure of code and data.
- E.g., Object-Oriented
  - What classes? What inheritance amongst the classes?
  - What classes will call what other classes?
  - How are classes grouped into subsystems (e.g. Java packages)?
  - What data members of classes
- Must decide these things at some point during the coding process.
  - Wish to minimize re-writes now and down the line
  - Danger in early over-complexity (c.f. Extreme Programming)

Architecture & Design

- Architecture
  - High-level
  - Major decisions
  - Not even thinking about programming
- Design
  - "Laying out" the programming language code used to implement the architecture
  - Organizing programming language concepts
- Both make decisions amongst many unknowns and attempt to minimize inconsistent invention as construction continues.
- Largely to help the designers and implementors share a vision of what they are creating.

Documentation of an Architecture

- If it's not reviewable (written down), it doesn't exist.
- Architectures sometime suffer from over-elaborate documentation
  - Unnecessary. Simply document your decisions.
  - Most systems don't deserve elaborate architectural documentation
- Dealing with unknowns
  - Indicate they are unknown for the present
  - Cycle back later and add new decisions taken
  - But beware of costs of postponing decisions vs cost of "analysis paralysis"
- Must religiously keep architecture document up-to-date
  - Very hard to do in practice: takes effort.
  - Exposes kludges.
  - I'm afraid to say I have personally experienced only a few big projects that released into production with an up-to-date arch document! (I was development manager in both cases.)
  - Therefore keep it simple as possible (but no simpler).

Two Main Architectural Structures

- Modular structure
  - Purely static
  - Not evident at run-time
  - May or may not be supported by the implementation language(s) and/or runtime.
- Structures that survive through execution
  - E.g., pipes, processes, networks, objects, files..
- Both views need to be considered (not the same)

The Essence of the Architecture Document

- Imagine after the system has been built attempting to describe as cogently and in as compact a form as possible how the system has been put together.
- Your target audience is knowledgeable professionals in the field, but unfamiliar with the domain.
- Must be distillable into a 1 hour pitch that should suffice to get across the basic concepts and structure.
- Stakeholder readers will wish to evaluate your choices
- Development team readers will wish to share your vision of the structure of the system
- Must be clear as a bell.

Unfortunate Reality

- Often one of the few documents written by engineers seen outside the core team.
- Often reviewed when asked if a given infrastructure is worth investing more in
  - For instance when called on carpet to justify further expenditure.
- Especially scrutinized when early drops exhibit performance problems!
- Unfortunately often are quite political documents,
  - written by consultants
    - who are supposed to take the spears..
  - involve vendor decisions, hence scrutinized by extremely astute "sales engineers".

Documentation of a Design

- UML (Unified Modeling Language)
  - Expresses OO design using diagrammatic notation
  - Complete UML for a typical system is very large.
  - A selection must be made for presentation
    - Choose the most illuminating parts
    - Simplify w.r.t. the actual code
    - Divide into small sections (< 1 page)
    - Add written text to describe the whys and wherefores.
- Danger of UML and code getting out of synch over time
  - Automated tools to keep the two in-synch
    - E.g., Rational Rose
  - Tools are not perfect:
    - Steep learning curves to achieve significant automation
    - Not literate
    - Don't work as well as we would want, cumbersome to use
    - Eliding detail is difficult, simplifying (lying) is difficult
    - Selection of parts for presentation is primitive
- Strive to explain (in writing) your choices to another programmer

Documentation

- Architecture
  - Informal diagrams
  - Written explanations
  - Bullet points
- Design
  - UML is somewhat more formal
  - Reflects actual program structure(?)
  - Simplify and divide into small chunks for presentation
  - Add written explanations.
    - lines and bubbles never will replace English

The Waterfall Development Process

- **Requirements → Architecture → Design → Code → Test**
  - Variations: Spiral, prototyping, …
    - All will have architecture and design artefacts
- Dave Parnas: "A Rational Design Process: How and when to fake it"
  - Not important that the steps are followed in this order
  - Only important that after the fact, there are documents that make it ***appear*** as though the process was followed in that order.
  - A little like how a math lecture presents discoveries.

Documentation In Practice

- As much requirements as you can manage without getting bogged down.
  - Requirements always contain conflicts. Resolving them all is very desirable but might take a lot of effort.
  - How is an important topic of research.
- As much architecture as you can manage without getting bogged down
  - Proposed architecture will probably contain important omissions.
- Some design
- Some code
- More design
- More code
- Refine architecture
- Fix requirement

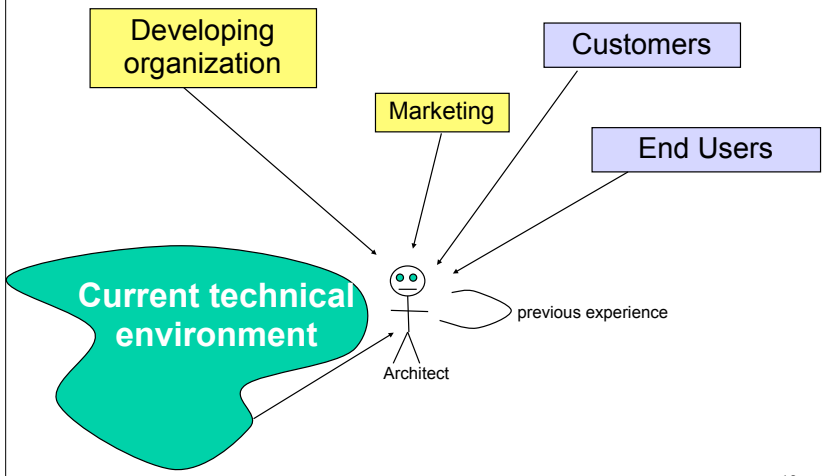
Why is architecture important?

- Manifests pivotal early design decision
  - most difficult to get correct and hardest to change
  - defines constraints on the implementation
  - inhibits or enables quality attributes
- Defines a work-breakdown structure
  - organization (especially important for long-distance development)
  - estimation
- A vehicle for stakeholder communication
  - an architecture is the earliest artefact that enables the priorities among competing concerns to be analysed
- Reviewable
  - architectural errors are vastly more expensive to fix once a system has been coded
  - Can serve as a basis for training new developers
  - As an indication of progress

Why is design important?

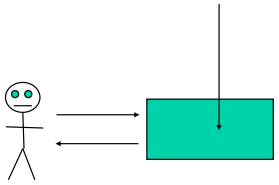
- When dealing with ~100s of packages and ~1000s of classes, coders lose sight of the forest for the trees.
  - Leads to designs that are muddled and inconsistent
    - Buggy, requiring constant re-work
    - Long learning curve for new developers
    - Hard to fix bugs
      - Long time to debug, lots of code to fix, introduce new bugs
    - Hard to change
      - Lots of time to figure out how to change, lots of code to change, introduce lots of new bugs
- Higher-level design descriptions lead to better designs
  - Can grasp the design at its essence and in its entirety
  - Can review and correct early
- Can be used to leverage the skills and experience of better designers across many developers

Where does architecture come from?



What does architecture affect?

- The structure of the developing organisation
- The enterprise goals of the developing organisation
- customer requirements for the next system
- influence later architectural decisions



Architecture process steps

- create the business case
- understand the requirements
  - They will be inconsistent.
  - What does understand mean?
- create the architecture
- represent and communicate the architecture
- evaluate the architecture
- implement based on the architecture
  - ensuring conformance
- enhance/maintain based on the architecture
  - ensuring conformance

Functionality & Quality Attributes

- Functionality usually takes 1st place during development.
- Systems are more frequently re-designed not because they are functionally deficient, but rather because
  - They are difficult to maintain
  - Difficult to port
  - Won't scale
  - Too slow
  - Too insecure
  - Not fault tolerant

System Qualities

- Observable via execution
  - Performance
  - Security
  - Availability
    - Reliability = mttf = mean time to failure
    - Availability = mttf/(mttf + time to repair)
  - Functionality
  - Usability
- Not observable via execution
  - Modifiability
  - Portability
  - Reusability
  - Integrability
  - Testability

Business Qualities

- Time-to-market
- Cost
- Projected lifetime
- Target market
- Rollout schedule
- Use of legacy systems
- Conformance to corporate technology standards.

(businessman, eh?)

Architectural Qualities

- Conceptual integrity
- Correctness
- Completeness
- Buildability
  - Completed by available team in a timely manner

Elegant? Clean? Pragmatic? Trendy?

Architectural Means of Achieving Quality

- Two questions
  - What structure shall I employ to
    - Assign workers
    - Derive a work breakdown
    - Exploit pre-packaged components
    - Plan for modification
  - What structure shall I employ so that the system, at runtime, fulfills its behavioral and quality attributes.