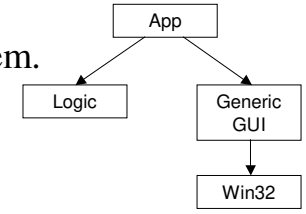
# Architecture Definition

- 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.






# 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)
  - inheritance structures (inherits)
  - …

# In Practice

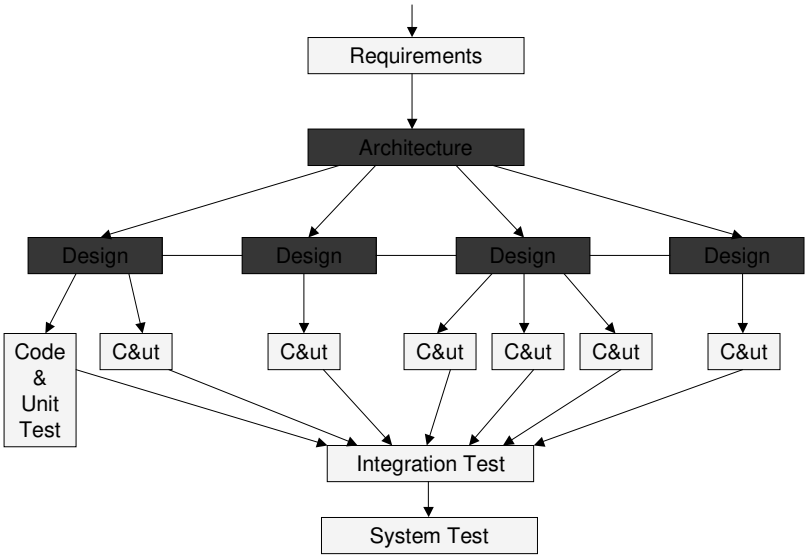
- Divide into two levels:
  - System-Level ***Architecture***
  - Programming-Level ***Design***

[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)
    - Calls constraints
    - Major strategies (e.g., for persistence, IPC, …)



## Software Design

- We are now considering how to lay down code.
- 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

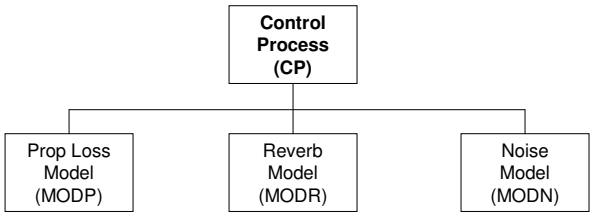
*But, … N.B. no standard terminology*



## Documentation of an Architecture

- Golden Rule of Software Development:
  - 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
- Must religiously keep architecture document up-to-date
  - Very hard to do in practice: takes effort
  - Therefore keep it simple as possible (but no simpler)

How do we describe an architecture?




- What is the nature of the components?
- What is the nature of the links?
- Does the layout have any significance?
- How does it operate at runtime
  - Dataflow
  - Control flow
- Can we evaluate this architecture?

**Must Be Clear!**



# Two Main Architectural Structures

- Modular structure
  - Purely static
  - Disappears at run-time
- Structures that survive through execution
  - E.g., pipes, processes, networks, objects, …
- 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.
- Be utterly clear
- you only have an hour in which to do it.
- your target audience is knowledgeable professionals in the field, but unfamiliar with the domain.
- They will wish to evaluate your choices



## 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
  - Problem with these tools:
    - 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
  - Formal UML
  - Reflects and in-synch with program structure
  - Simplify and divide into small chunks for presentation
  - Add written explanations.



## The Waterfall Model

- **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.

## Documentation In Practice

- As much requirements as you can manage without getting bogged down.
- As much architecture as you can manage without getting bogged down
- Some design
- Some code
- More design
- More code
- Refine architecture
- Fix requirements
- …



## Why is architecture important?

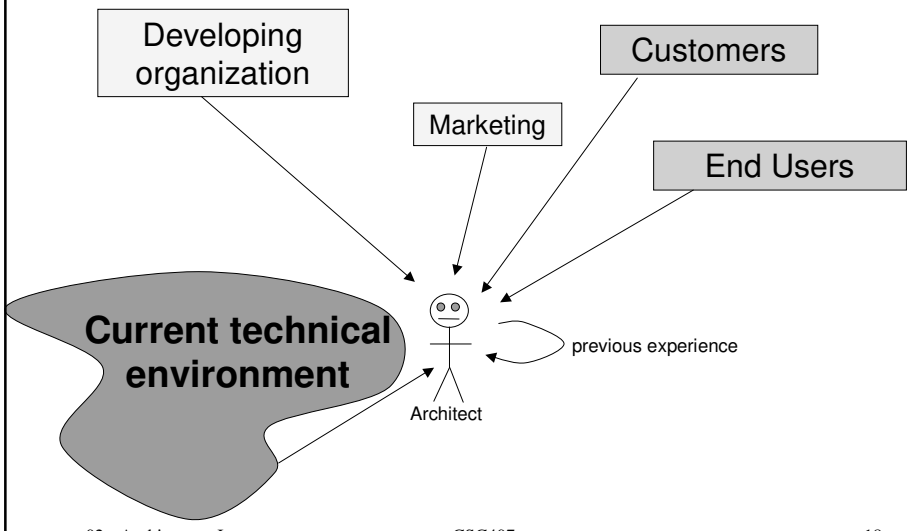
- Manifests early design decision
  - most difficult to get correct and hardest to change
  - defines constraints on the implementation
  - inhibits or enables quality attributes
- Defines a work-breakdown structure
  - organization (especially important for long-distance development)
  - estimation
- 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

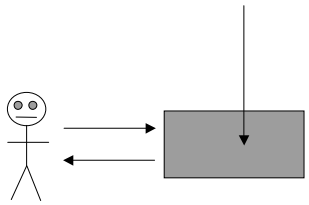


## Where does architecture come from?



# 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
- 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

## Architectural Qualities

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

## 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.