Systems-Level Architecture

A Re-Introduction

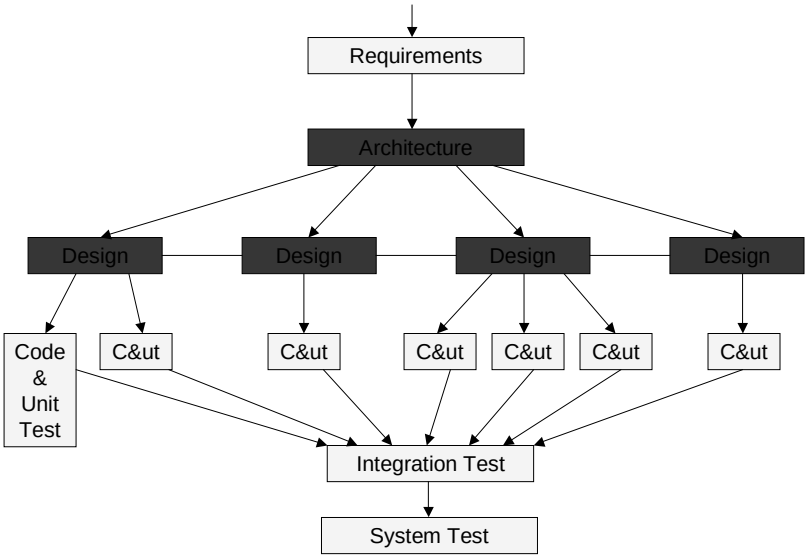
Level of Design

- Divide into two levels:
  - System-Level ***Architecture***
  - Programming-Level ***Design***
- You know what design is
  - OOD + written text = one example
- Next we will discuss architecture

Architecture & Design

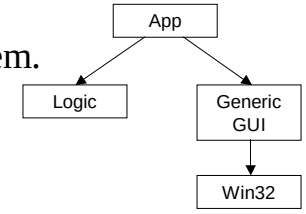
- Architecture
  - High-level
  - Major decisions
  - Not even thinking about programming
- Design
  - "Laying out" the programming language code used to implement the architecture
  - Organizing programming language concepts

Design & Architecture in the Development Process



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)
  - …

Link to Design in Java-Speak

- Referring to the modular structure
  - The "system" is the whole thing
  - A "sub-system" is the division into components of
    - the system
    - or of a sub-system
  - At the lowest level,
    - leaf sub-system = package
  - A package contains
    - a set of classes
  - Traditional to use the (hierarchical) directory structure to represent the system breakdown.
    - = hierarchical package structure in Java
  - Coupling and Cohesion (information hiding) guide the architectural division into sub-systems.
  - Must constrain who calls whom
    - imports, exports

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, …)

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 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)

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)

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.

Sample Systems Architecture Document

- Introduction
  - purpose of current document
  - what we are building and why
  - references to other documents
  - important business considerations
- Technical Requirements
  - platforms, portability, hardware available, existing systems
- Application architecture
  - user-facing apps
- Architectural paradigm
  - general idea of the system
- Data Architecture
  - how to load/store/write/warehouse data
- Run-Time & Physical architecture
  - processes/threads communications, allocation to hardware
- Technology choices
  - databases, languages, app/web servers, libraries, networks & IPC
- Module architecture
  - source code, re-use strategy

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
  - architecture document provides the vocabulary
- 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

Must Answer

- 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.

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 Paradigms in Common Use

- Monolithic Systems
  - single/multi threaded
- Client/Server
  - roll your own
- Classic RDBMS C/S
  - ex. Java JDBC
- Distributed Systems
  - ex. Java RMI
- N-tiered systems
  - ex. Java EJB