3: Software Architecture

Intro

A lot of attention has been given to requirements engineering, but it's less interesting than this field – how we make things and put things together.

Civil Engineering vs. Software Architecture

These tend to have a lot in common, and make good metaphors for one another.

Definitions

Architectural drivers:

- The set of quality attributes that shape a system's software architecture

Documenting

Different views for different stakeholders.

Views

You need multiple views because you can't capture everything in 1 view.

There are 3 questions to ask of the architecture:

1. How is the system structured in terms of implementation units?
2. How is the system structured in terms of runtime behaviour and interactions?
3. How does the software relate to non-software structures in its environment?

Book recommendation: The Architecture of Open Source Applications

We want to reason about different diagrams and what they mean.

Module Views

Modules are units are implementation – a module view documents module structures of a system's architecture.

Talk about static structure.

There are relations among modules:

- is-part-of
- depends-on
- is-a

Uses

Module views are used for:

- Construction
  - e.g. provide blueprint for source code
- Analysis
  - e.g. requirements traceability, impact analysis (impact of changes)
- Communication
  - large systems are written by many people, and you need to communicate to each other how the systems are structured
  - explain system functionality and structure

Component & Connector Views

A component is a runtime entity that can be deployed independently – a C&C view documents elements that have a runtime presence. E.g. since classes are blueprints for objects and aren't running, they're not components.

How are components implemented?

- Services
- Threads, processes
- Clients, servers

How are connectors implemented?

- Pipes or queues

- Request/reply protocols
- Direct/indirect/event-driven invocation

Ports and Roles

Ports are the interfaces of components.

Roles are the interfaces of connectors.

Why use C&C views?

- Show how the system works
- Guide development
- Analysis

Relations

1. Attachment
2. Interface Delegation

Allocation Views

Allocation views map software elements to non-software elements.

Non-software elements are the environment, e.g.:

- computing and communication hardware
- file management systems
  - software but not part of your system

Software elements have properties that are required from the environment (e.g. games requiring certain specs from a computer).

The environmental element has properties that are provided to the software.

Architectural Significant Requirements

Distinguishing Architectural from Non-Architectural Design

The purpose of architecture is to ensure the satisfaction of a system's quality attributes. To that end, an architect makes architectural design decisions.

To distinguish architectural from non-architectural design, ask:

- "Does this permit the system to achieve its quality attributes, or prohibit it from achieving them?"

How Not to Distinguish Them

All architecture is design, but not all design is architectural. Architectural does not mean, for example:

- Lack of detail
- Big designs

ARS Characteristics

An ASR must have the following characteristics:

[...]

Quality Attributes

Here are 6 important quality attributes:

1. Availability
2. Modifiability

Availability

Availability is concerned with:

- System failure and its consequences
- Failure detection and prevention
- Frequency of failure and repair time

System failure is when a system stops delivering a specified service – this is visible to the user.

Faults are invisible to the user – if not handled correctly, they may cause system failures.

Modifiability

Concerned with the cost of change.

Important points are:

- what can change?
- when can changes be made?
- who makes changes?

Performance

Concerned with timing: how long does a system take to respond to an event?

Events are generated by users, external systems, or within the system itself.

Events can happen:

- periodically (fixed pattern)
- stochastically (probabilistic pattern)
- sporadically (no pattern)

Security

Concerned with resisting unauthorised usage while servicing legitimate users.

An attack is an attempt to breach security:

- unauthorised attempt to access the system
- attempt to deny services to legitimate users

Testability

Ease with which software can be made to demonstrate its faults [...]

Usability

Concerned with how easy a user can accomplish a desired task, and the support provided by the system.

Usability is not just having a nice interface – undo / cancel options, for example.

Concerns:

- learning system features
- using a system efficiently
- minimising impact of errors
- adapting to users' needs
- increasing confidence and satisfaction

4 Common Architectural Styles

1. Layers
2. Client-Server
3. Peer-to-Peer
4. Pipes & Filters

Styles vs. Patterns

Architectural styles and patterns are more or less the same thing, but are documented differently.

Patterns are common solutions to recurring problems in a given context. A set of patterns forms a pattern languages.

Why Study These?

Almost all systems use one or more styles.

Architectural Styles

The choice of architectural style has consequences for a system's quality attributes. The degree of the effect may depend on implementation details.

These 4 styles have some associated benefits and drawbacks.

Each system has an architecture, whether it's documented or not.

1: Layers

The layered (or n-tier) style decomposes a system into a set of layers or tiers.

Each layer represents a different level of abstraction to deal with issues or data.

Each layer has a clearly defined interface which can be tested and hides implementation details.

Example: OSI 7-layer reference model

Benefits and Drawbwacks

Layers improve modifiability & portability. Layers having well-designed interfaces, which facilitates maintainability and testability.

However, performance may suffer due to levels of indirection. It's also not easy to establish the right levels of granularity.

2: Client-Server

Decomposes a system into a server that provides a service, and 1 or more clients that use that service.

Both client and server are independent processes.

Server must handle multiple requests.

Example: X Window System

Benefits and Drawbacks

You get modularity and separation of concerns. Scalability can be easy be adding more servers. You can also reuse existing connectors.

However, a server provides a single point of failure if there's only one. You may also get traffic congestion if too many requests arrive at once.

3: Peer-to-Peer

Similar to client-server, but each client also acts as a server.

Benefits and Drawbacks

It's very scalable – no single node is a bottleneck. You also get resilience (decentralised nature of the network makes it hard to take it down).

Propagation of updates is not instant. Also, there is no single trusted server.

4. Pipes and Filters

This style decomposes a large processing task into a sequence of smaller, independent processing steps.

Each processing step is a filter, with defined input and output interfaces.

Filters are connected with pipes – a pipe can be (e.g.) a buffer or a file.

Benefits and Drawbacks

- potential performance gain when filters can be run in parallel
- filters are reusable, as they are loosely-coupled modules with well-defined interfaces

- Can reuse existing connectors: e.g. Linux pipes
- security is difficult: must be implemented for each filter
- usability: usually pipes aren't interactive, which is a problem
- error handling is difficult