Software Systems Architecture

FEUP-M.EIC-ASSO-2023-2024

Ademar Aguiar, Neil Harrison

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Bowling: what did you learn?

How well did the procedural paradigm fit?

- Why?
- What types of problems fit well with procedural programs?

How well did the OO paradigm fit?

- Why?
- What types of problems fit with OO designs?

How well did the Pipes and Filters fit?

- Why?
- What types of problems fit well with P&F?

Change happens!

- What if the program supports multiple players taking turns?
- What if you add error checking (and report at the end?)

Designing Architectures

The big question!

1. The Design Process
2. Architectural conception
3. Styles and Patterns
4. Architectural conception in a vacuum
5. Putting it all together

How do you design software?

Let's observe it happening in the wild! ☺️

Groups of Two teams:

- One team: design a software system (See Spec)
  - Work together around a board
  - Write your ideas/architecture
- The other: watch everything they do:
  - HOW do they design software?
  - What activities, in what order?
  - Write down what you see

What did you observe?

What you might have observed

Trying to break the system into pieces

- Drawing boxes for the pieces
- Getting it wrong and erasing some boxes

Discussing quality attributes

- Like reliability or performance

Discussing how the pieces communicate with each other

- Should happen, but often doesn't!

Discussing external software to use

Lots of questions that aren't answered

- Should write them down, usually don't

Discussed some things in detail

- Sometimes too much detail (right now)

Often, one person becomes dominant in a group

- Dominates the conversation
- Might also hold the marker most of the time

Essential Activities in System Design

Learn the functional requirements
- (of course!)

Understand the non-functional requirements
- (Thought question: why is this important at the architectural level?)

Architectural Conception
- First ideas of the architecture

Create the physical layout of the system
- If hardware is involved (beyond one computer)

High-level partitioning
- Design chunks of functionality (components)
- Understand how the components interact (what are the connectors?)

Consider technology alternatives

Note: the above list is not ordered, and it's iterative

Architectural Conception

The first, very general, concept of what the system design (the architecture) will look like

Mind Tools:

- Partitioning, separation of concerns
- Abstraction
- Simple machines
- Prior knowledge (Refined experience)

Physical tools

- Whiteboard
- Cards (similar to CRC cards)
- Your colleagues ☺️
- Some visualization tools

Tools that don't work:

- Software tools, such as CASE (Computer-Aided Software Engineering)
- Code generators (wrong level: too detailed)

Weighing Alternatives

Many Architectural Decisions consist of selecting one choice among several alternatives

Particularly true during Conception

- But happens throughout architecture (and development)

Technology Alternatives

- Please give examples

Non-Technology Alternatives

- Please give examples (and why are they architectural?)

Partitioning

The basic action of architectural design

Why?

- Because we can't hold the whole design in our head
- Separation of concerns, reduce coupling
- Convenience for development, multiple people can work on it
- Can isolate/reduce errors
- Might find some parts already done
- Might be able to reuse parts elsewhere
- Ease in maintenance and extension: easier to understand

Separation of Concerns

You can partition a system millions of ways!

So the big questions are:

- What is the best partitioning?
- What makes one partitioning better than another?
- What criteria should you use to partition?

References:

- David Parnas, “On the Criteria to be Used in Decomposing Systems into Modules” (1972!)
- David Parnas, Paul Clements, David Weiss: “The Modular Structure of Complex Systems” (1985)

Characteristics of Good Modules

Easier to understand:

- The modules make intuitive sense
- We have a high-level picture; not distracted by the weeds

Modifications to the system typically involve few modules

- Horror story: telephone system

The interfaces between the modules change relatively infrequently

Low coupling

High cohesion (Everything in a partition should be about the same thing: Single Responsibility Principle)

Modules and Hardware

Many modern systems involve multiple hardware devices

- Hardware units are natural modules
- Or distributed across networks

Some partitions are obvious:

- An app on your phone typically has a phone piece and a server piece
- The obvious hardware partitioning is a good starting point

Some may be less obvious, or even our choice

- We may weigh factors such as cost, performance, capacity, etc.
- And what makes sense to us!

Modules and Data

Data can guide us to partition into modules

Common:

- For persistent data, a module that manages the data (i.e., a database)

Also consider putting operations on a common set of data together into a module

- What examples can you think of?

User Interactions and Modules

User interactions can guide partitioning

Can have modules that deal with the user interaction

Can have different modules for different types of users

Responsibilities and Modules

When partitioning, consider that each module:

- Ideally has ONE major responsibility
- (so you can identify the major responsibilities and make a component for each)
- Of course, the major responsibility may have sub-responsibilities

Can also communication with other module

Can model it by writing on cards

- Similar to CRC (Class, Responsibility, Collaborator) cards
- But at a higher level of granularity

Much of the software architecting activity is related to partitions:

- What are the modules?
- What are the responsibilities of each modules?
- What are the connections between module?

- Does each scenario of usage actually work in the proposed partitioning?

Scenarios

Walk through usage scenarios to validate the partitioning

You can do it once you have a first, rough cut at partitioning

Abstraction

The structured removal (or hiding) of information

Abstraction can encompass sets of closely related things that can be considered generically (think Base Classes)

Or can be a general idea about something, without considering the details

Side note: Diving into details

As you design an architecture, you work mainly at a very high level

Occasionally, you go into more detailed design

- It might answer some question. For example, you might wonder how a particular design might meet a non-functional requirement. You might have to dive down into more detail

Then pop back up to high level

Don’t get stuck in the weeds (the details)

How deep is too deep?

- For example, the data and methods of a given class are “the weeds”.

A Continuum of Detail?

What's the difference between high-level design and detailed design?

Is it just a matter of scale and levels of detail?

Yes and no

Some design is focused on partitioning

- This is mainly large scale and high level
- But it can be small scale and fairly detailed

But some design is algorithm creation

- Pretty much all low-level and detailed

Architecture is *mainly* concerned with the partitioning

Domain-Specific Software Architecture

Just what it sounds: an architecture for multiple similar applications in a particular domain

Pretty close to product families

- Reference architecture of a product family is a specialization or a subset of a Domain-Specific Software Architecure

A combination of:

- A reference architecture for an application domain
- A library of software components for that architecture containing reusable chunks of domain expertise
- A method of choosing and configuring components to work within an instance of the reference architecture

Building analogy:

- Row houses: not identical to each other, but similar
  - Easy to build
  - Easy to customize

A Row of Houses