Hugbúnaðarverkefni 2 / Software Project 2

11. Software Architecture

HBV601G – Spring 2019

Matthias Book

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

In-Class Quiz 9 Prep

- Please prepare a small scrap of paper with the following format:

ID: ____@hi.is   Date: ____

a) ____   e) ____
b) ____   f) ____
c) ____   g) ____
d) ____

- During class, I'll show you questions that you can answer with a number
- Hand in your scrap at end of class
- All questions in a quiz have same weight
- All quizzes (8-10 throughout semester) have the same weight
  - Your worst 2 quizzes will be disregarded
- Overall quiz grade counts as optional question worth 7.5% on final exam

Assignment 4: Schedule and Deliverables

- On **Thu 11 Apr**, **demonstrate** and **explain** your product to your classmates in a 10- to 15-minute-presentation:
  1. **Product:** What does your product do? Demonstrate the key features of your system.
  2. **Architecture:** How does your product work? Explain architecture & key design decisions.
  3. **Process:** How did you build the product? Relate and interpret challenges you faced.

- On **Sun 14 Apr**, submit your **final product** in Ugla, including:
  - Complete source code and installation instructions
  - Slides of your final presentation

- Your product does not need to satisfy all the criteria you listed in your initial requirements document, but the key features should work.

Assignment 4: Final Presentation Format

- All teams will present their work in the common area on the ground floor of Tæknigarður.
- Each team will get a table to set up a laptop and/or mobile device(s) for an hour to demonstrate their product to other teams and tutors visiting their table
  - Teams 1-10: 13:20-14:20 (setup begins 13:00)
  - Teams 11-20: 14:25-15:25
  - Teams 21-30: 15:30-16:30
- During their timeslots, teams repeat their 10- to 15-minute presentation 3-4 times as different teams are visiting their table.
- Please attend the other timeslots as well to visit other teams' tables and learn about their product ideas, technologies used, and challenges they experienced.

Assignment 4: Grading

- Your product will be graded during the presentation attended by your tutor.

- Team grade refers to the visible parts of the presentation
  - i.e. information on slides, impression the product makes, etc. – Criteria:
    - Final product implements key features, and is running smoothly (75%)
    - Critical retrospective given of chosen process, architecture and technology (25%)

- Presenter grade refers to the audible parts of the presentation
  - Criteria: how well the presenter explains things, answers questions etc.
  - Presentation to tutor must be given by team member who has not presented any assignments yet (the other presentations may be given by other team members)
  - If all team members have gotten a presentation grade already, the presentation quality will be reflected in the team grade.

In-Class Quiz 8 Solution: Android Threading

- Note whether the following properties apply to
  (A)syncTasks, (I)ntentServices, (H)andlerThreads
  (note that some properties may apply to several threading mechanisms):

a)	Executed in app’s background thread	A
b)	Executed in background thread of its own	I, H
c)	Scheduling controlled by Android OS	A, [I]
d)	Scheduling controlled by developer	I, H
e)	Started via intent	I
f)	Started via method invocation	A, H
g)	Independent of particular activities	I
h)	Tied to particular activity	A, H

Software Architecture

see also:

Larman: Applying UML and Patterns, Ch. 13 & 33

Definition: Software Architecture

A software architecture is

- the set of significant **decisions about the organization** of a software system,
- the **selection of the structural elements** of which the system is composed,
    - and their **interfaces**,

together with

- their behavior, as specified in the **collaborations** among those elements,
- the **composition** of these structural and behavioral elements
    - into progressively larger subsystems,

and

- the architectural **style** that guides this organization
    - of the elements and their interfaces, their collaboration, and their composition.

Architectural Analysis

- How do we come up with a software architecture?

- **The essence of architectural analysis** is to
  - **identify** **factors** that should influence the architecture,
  - **understand** their **variability** and **priority**, and
  - **resolve** them by making architectural **decisions**.

- Challenge: It’s difficult to…
  - Know what questions to ask
  - Weigh the trade-offs
  - Know the many ways to resolve an architecturally significant factor
  - Decide on the best way under the given circumstances

Examples of Issues to be Resolved at Architectural Level

- How do **reliability** and **fault-tolerance** requirements affect the **design**?
  - For what remote services will fail-over to local services be allowed? Why?
  - Do the local and remote services work exactly the same? What are the differences?
- How do the **licensing** costs of purchased subcomponents affect **profitability**?
  - Should we integrate a high-quality third-party persistence framework that will charge a fee on all transactions, go with a low-cost open-source alternative, or develop or own?
- How do the **adaptability** and **configurability** requirements affect the **design**?
  - What variations of business rules need to be reflected in the implementation?
  - What degree of evolution should be accommodated? How should variations be specified?
- How do **availability** and **dependability** requirements influence the **effort**?
  - Very strict availability requirements may induce huge costs for redundant hardware, hot-swap capability, failover mechanisms, backups, etc. Does this effort correlate with the risk?

Common Steps in Architectural Analysis

- Goal: Understand **influence, priorities** and **variability** of architectural drivers

1. Identify and analyze **architectural drivers***, i.e. requirements that have an architectural impact
   - Especially non-functional (quality) requirements
   - But also functional requirements, esp. regarding expected variability or change
   - Can be found e.g. in
     - Vision and Scope document
     - Business Rules document
     - Supplementary Specification document
     - Use Case document (sections on special requirements, technology variations, open issues in fully-dressed format)
2. Make **architectural decisions**
   - Analyze alternatives
   - Create solutions that address the impact:
     - Build a custom solution
     - Buy a third-party solution
     - Remove the requirement
     - Hire an expert
     - …
3. Document decisions
     - So people will not inadvertently undermine design decisions later
     - or spend time pursuing rejected options

*** Alternative term in some literature: “architectural factors”**

Making Architectural Decisions

- Collecting and describing architectural drivers is comparatively easy.
- Addressing/resolving them in light of **interdependencies** and **trade-offs** is much more difficult.
  - Highly dependent on application domain and technology
  - Business goals and stakeholder interests become central to technical decisions
  - Requires consideration of more large-scale/global goals and trade-offs than local-scale UI or OO design decisions
  - Requires knowledge and critical consideration of many areas:
    - Architectural styles and patterns, technologies, products, pitfalls, domain knowledge, trends…
- Hierarchy of **goals** to guide **priority** of architectural decisions:
  1. Inflexible constraints, e.g. safety and legal compliance – unavoidable
  2. Business goals, e.g. particular features, deadlines etc. – some flexibility
  3. All other goals (are often derived from business goals) – some leeway for interpretation

Characteristics of Software Architecture

- Architectural concerns are especially **related to non-functional requirements**, but their resolution **permeates the implementation of the functional requirements**.
- Architectural concerns **require awareness** of the **business context, stakeholder goals**, as well as **requirements' variability** and **evolution**.
- Architectural concerns involve **system-level, large-scale issues** whose resolution usually involves **large-scale, fundamental decisions** that are **extremely costly to change** later on.
- Architectural decisions depend on the **conception** and **critical evaluation** of **alternative solutions**.
- Architectural decisions usually involve **interdependencies** and **trade-offs**.

Architectural Structures

see also:

- Bass et al.: Software Architecture in Practice, Ch. 1-3

Definition: Architectural Structures

= “Perspectives” on a system

Can be described in design documents, UML diagrams, etc.

- Software architecture is the set of structures needed to **reason** about a system.
- A structure is a set of software **elements** and the **relations** between them.
  - **Module structures** show how a system is to be statically structured as a set of code or data units that have to be constructed or procured.
  - **Component-and-connector structures** show how a system is to be dynamically structured as a set of modules having runtime behavior (components) and interactions (connectors).
  - **Allocation structures** show how software structures are to be mapped to the system’s organizational, developmental, installation and execution environments.
- Each structure has the potential to **influence quality attributes** of the system.
  - e.g. availability, interoperability, modifiability, performance, security, testability, usability
- A system’s architecture is **established through a series of design decisions**.
  - e.g. responsibility allocation, resource management, binding times, technology choices
- These **occur in** and **affect** technical, project, business, professional **contexts**.

Definition: Architectural Structures

"Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled."

Eoin Woods

- Software architecture
- A structure is a set o
  - **Module structures** show
    as a set of code or data units tha
  - **Component-and-connector structures** show how a system is to be ... y structured as a set of modules having runtime behavior (components) and interaction ... ctors).
  - **Allocation structures** show how software structures are to be mapped to the sys ... 's organizational, developmental, installation and execution environments.
- Each structure has the potential to **influence quality attributes** of the system.
  - e.g. availability, interoperability, modifiability, performance, security, testability, usability
- A system's architecture is **established through a series of design decisions**.
  - e.g. responsibility allocation, resource management, binding times, technology choices.
- These **occur in** and **affect** technical, project, business, professional **contexts**.

Module Structures

- Module structures show how a system is to be **statically structured** as a **set of code or data units** that have to be constructed or procured.
- Focus on modules and relationships established at **design time**
  - e.g. layers such as database, business logic, user interface; subdivided further into components, which are subdivided into classes, which are subdivided into methods
  - ➢ **Which modules are there? How are they composed? How do they rely on each other?**
- Examples of module structures:
  - **Decomposition structure:** Shows how modules are recursively composed of submodules
    - Relevant for localizing responsibilities expressed in functional requirements, organizing the project, supporting modifiability through encapsulation, etc.
  - **Uses structure:** Shows how modules are depending on other modules
    - Relevant for setting priorities, tracing faults, supporting extensibility and reusability, etc.

Component-and-Connector Structures

- Component-and-connector structures show how the modules **behave and interact** with each other at run time to carry out the system's **functions**.
- Focus on module instances and their relationships manifested at **run time**
  - i.e. components such as services, peers, clients, servers, filters etc., and connectors such as call-returns, process synchronization operations, pipes, etc.
  - ➢ **How are the module instances hooked together at runtime? How do they interact?**

- Examples of component-and-connector structures:
  - **Service structure:** Shows how (possibly independently engineered) services can interoperate with each other using a service coordination mechanism
    - Relevant for supporting interoperability, reliability etc.
  - **Concurrency structure:** Shows where opportunities for parallelism exist and where resource contention may occur
    - Relevant for supporting performance, testability, availability etc.

Allocation Structures

- Allocation structures show how **software structures** are to be **mapped to** the **system's** organizational, developmental, installation & execution **environments**.
- Focus on non-software structures in the system's **environment**
  - e.g. devices, CPUs, file systems, networks, development teams, etc.
  - ➢ **Who builds the modules? Where are they run/stored? What infrastructure do they use?**

- Examples of allocation structures:
  - **Work assignment structure:** Shows how responsibility for implementing the modules is assigned to teams, and which expertise is required on each team
    - Relevant for staffing, project management, collaboration infrastructure etc.
  - **Deployment structure:** Shows how modules are allocated to hardware components for processing and communication, and how they may migrate between hardware components
    - Relevant for supporting performance, data integrity, security and availability, especially in distributed and parallel systems

Design Decisions Shaping Architectural Structures

- A **module structure** defines e.g.:
  - What is the primary functional responsibility assigned to each module?
  - What other modules is a module allowed to use?
  - What other modules does it actually use and depend on?
  - What modules are related to other modules by inheritance relationships?
- An **allocation structure** defines e.g.:
  - What processor is each module instance executed on?
  - In what directories or files is each module stored during development, testing and building?
  - What is the assignment of each module to development teams?
- A **component-and-connector structure** defines e.g.:
  - What are the major executing modules and how do they interact at runtime?
  - What are the major shared data stores?
  - Which parts of the system are replicated?
  - How does data progress through the system?
  - What parts of the system can run in parallel?
  - Can the system's structure change as it executes, and if so, how?
  - What run-time properties (performance, security, availability etc.) does the system exhibit?

Architectural Design Decisions

see also:

- Bass et al.: Software Architecture in Practice, Part II

Recap: Architecture's Impact on Quality Attributes

- A system's architecture determines the system's **quality attributes**, e.g.
  - Availability
  - Interoperability
  - Modifiability
  - Performance
  - Security
  - Testability
  - Usability

- A system's architecture is established in a series of **design decisions**, e.g.
  - Allocation of responsibilities
  - Coordination model
  - Data model
  - Management of resources
  - Mapping of architectural elements
  - Variations and binding times
  - Technology choices

Architectural Design Decisions Shaping the Module Structure

- Decisions about the **allocation of responsibilities**
  (i.e. where functional requirements will manifest themselves)
  - Identifying the important responsibilities, including basic system functions, architectural infrastructure, satisfaction of quality attributes
  - Determining how these responsibilities are allocated to modules at design time and run time

- Decisions about the **data model**
  (i.e. the representation of entities whose processing is the main purpose of the system)
  - Choosing the major data abstractions, their operations and properties
    - i.e. how to create, initialize, access, persist, manipulate, translate, and destroy data entities
  - Compiling metadata needed for consistent interpretation of the data
  - Organizing the data
    - e.g. object-oriented vs. relational representation, conversion between both representations

Architectural Design Decisions
Shaping the Component-and-Connector Structure

- Decisions about the **coordination model**
  (i.e. how modules interact through designed mechanisms)
  - Identifying the modules that must cooperate, or are prohibited from cooperating
  - Determining the properties of the cooperation, e.g. timeliness, currency, consistency etc.
  - Choosing the communication mechanisms between modules and systems
    - e.g. stateful vs. stateless, synchronous vs. asynchronous, guaranteed vs. best-effort

- Decisions about **variations** and **binding times**
  (i.e. allowable range, time and mechanism of variations of a system)
  - Identifying variation points in which functionality/properties of the system can be changed
  - Deciding on the range of supported variations per variation point
  - Deciding on the binding time of variations, and the actor choosing a variation
    - e.g. at design time by the developer, at deploy time by an install wizard, at run time by the user…
  - Choosing mechanisms to specify and execute the variation
    - e.g. in source code, compiler directives, make files, configuration files, graphical user interface…

Architectual Design Decisions Shaping the Allocation Structure

- Decisions about the **mapping between architectural elements** *(i.e. between elements of development and execution; and software and hardware elements)*
  - Mapping design-time modules and run-time instances
  - Assigning run-time elements to devices or processors
  - Assigning entities in the data model to data stores

- Decisions about **management of resources** *(i.e. arbitrating the use of shared resources, e.g. CPU, storage, peripherals etc.)*
  - Identifying resources to be managed, and determining limits for each
  - Determining which modules should manage which resource
  - Determining how resources are shared, and what arbitration mechanisms are required
  - Determining and avoiding the impacts of saturation of different resources

Architectural Design Decisions
Shaping All Structures

- Decisions about **technology choices**
  (i.e. selection of suitable technologies to support preceding architectural decisions)
  - Identifying candidate technologies that can realize decisions made in preceding categories
  - Determining whether adequate internal and external expertise is available for a technology
  - Determining side effects of a technology, e.g. in coordination or resource management
  - Determining compatibility of a new technology with existing technology stack
  - Determining whether available tools adequately support technology choices
  - Choosing a suitable technology to support a particular requirement or architectural decision

In-Class Quiz 9: Architectural Design Decisions

Decisions about…

a) the allocation of responsibilities
b) the coordination model
c) the data model
d) the management of resources
e) the mapping between architectural elements
f) technology choices
g) variations and binding times

…determine:

(one answer per question)

1. the arbitration of the use of shared resources, e.g. CPU, storage, peripherals etc.
2. the representation of entities whose processing is the main purpose of the system
3. the association of elements of development and execution; and of software and hardware elements
4. where functional requirements will manifest themselves
5. how modules interact through designed mechanisms
6. the allowable range, time and mechanism of variations of a system
7. the selection of suitable technologies to support the above architectural decisions