Architecture and Interface Design
CIS 423/523

Stuart Faulk
Spring 2016

Outline

• Reminder: turn in metric forms!
• Update on integrations?
• Revisit architecture and interface design
• Meeting topics
From Tuesday

• DSD: coordination at a distance introduces specific kinds of risks
• Risks manifest themselves as a loss of control
  – Intellectual: not building what the stakeholders want
  – Managerial: exceeding time/cost budgets
• Software Engineering: purpose is to maintain control
  – Identify specific risks for the DSD project and organization
  – Introduce methods to mitigate risks, maintain control

Identified issues

• Requirements
  – Incomplete
  – Inconsistent understanding, assumptions
• Architecture
  – Inconsistent design goals
  – Different design decisions at different sites
  – Inconsistent specifications of components and relations
• Result: cannot integrate components as written
### Qualities Established in Architecture

<table>
<thead>
<tr>
<th>Behavioral (observable)</th>
<th>Developmental Qualities</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Performance</td>
<td>• Modifiability (ease of change)</td>
</tr>
<tr>
<td>• Security</td>
<td>• Portability</td>
</tr>
<tr>
<td>• Availability</td>
<td>• Reusability</td>
</tr>
<tr>
<td>• Reliability</td>
<td>• Ease of integration</td>
</tr>
<tr>
<td>• Usability</td>
<td>• Understandability</td>
</tr>
<tr>
<td></td>
<td>• Provide independent work assignments</td>
</tr>
<tr>
<td></td>
<td>• Subsetability (extend/contract)</td>
</tr>
</tbody>
</table>

Properties resulting from the properties of components, connectors and interfaces that exist at run time.

Properties resulting from the properties of components, connectors and interfaces that exist at design time whether or not they have any distinct run-time manifestation.

*Which address distinct DSD issues/risks?*

### Architecture Definition

“The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.”

From *Software Architecture in Practice*, Bass, Clements, Kazman

Remember as: **Components, Interfaces, and Relations**
Choose Structures

- Module Structure
  - Decomposition of the system into work assignments or information hiding modules
  - Most influential design time structure
    - Modifiability, work assignments, maintainability, reusability, understandability, etc.
- Uses Structure
  - Determine which modules may use one another's services
  - Determines subsetability, ease of integration (e.g. for increments)
- Deployment structure
  - Determine constraints on how the system will be deployed
  - Constraints on platforms, networks, etc.

Work-Breakdown Structure Goals

Goal: distribution of work on components requiring least inter-team communication
In practice this means...

- Want to decompose into components that
  - Are team work assignments
  - Implement all requirements
- Inter-component dependencies are minimized
  - Expose only what is necessary
  - Encapsulates properties likely to change
- Services provided by each component are well defined
- Together the components implement the complete set of requirements

Module Structure

- Architectural model: called the Module Structure
- Components
  - Called modules
  - Leaf modules are work assignments
- Relations
  - “submodule-of”
  - The set of submodules of any module X partition X’s functionality
  - Constrained to be acyclic tree (hierarchy)
- Module interfaces
  - Modules at the leaves of the tree provide the methods implementing
    the system’s functionality
  - Module interfaces specify everything one must know to use the
    module’s services correctly
  - Modules encapsulate properties other modules should not depend on
Module Hierarchy

Remote Authentication Module Structure

1. Authentication Server Module
   The Authentication Server includes programs that need to be changed if the methods for accessing the remote authentication services change or the procedures for providing authentication services change.

   Services
   Provides services supporting remote to client APIs over the web. This includes services for creating an account for an individual, verifying the system to recognize images of the individual, and authentication services determining whether an image is consistent with previously trained images of a registered individual.

   Secrets
   How to use the services provided by other modules to provide authentication services. How remote access is handled.

2. Facilitator
   The Facilitator includes programs that need to be changed if the methods for providing facial recognition change.

   Services
   Provides facial recognition services to other components of the system. This includes services for training the facial recognition algorithms to recognize an individual based on a set of images. It also includes services for recognizing an individual in a provided image where previous training has occurred.

   Secrets
   The methods or algorithms used to train the system to recognize an individual's face as well as the algorithms for recognizing a face in a given image.
3. Database
The Database includes programs that need to be changed the kinds of data stored by the Authentication Server application change or the methods of storage change.

Services
Provides services for storing and retrieving account information, data, and images associated with an individual account.

Secrets
Algorithms and data structures used to store or retrieve data.

4. Client API
There may be a different client API module for each platform where Remote Authentication services are provided. A Client API includes programs that need to be changed if the specific platform for the API changes.

Services
Provides all of the services of the Remote Authentication application in the form of a native API (library) for a specific platform (e.g., Android mobile device).

Secrets
Hides the algorithms and methods used for accessing the web-based services provided by the Authentication Server on the given platform.

Interface Specification

Module Interface Specifications
– Documents all assumptions users can make about the module’s externally visible behavior (of leaf modules)
  • Access programs, events, types, undesired events
  • Design issues, assumptions
– Document purpose(s)
  • Provide all the information needed to write a module’s programs or use the programs on a module’s interface (programmer’s guide, user’s guide)
  • Specify required behavior by fully specifying behavior of the module’s access programs
  • Define any constraints
  • Define any assumptions
  • Record design decisions
Abstract Interfaces

• An *abstract interface* defines the set of assumptions that one module can make about another

• While detailed, an abstract interface specification does not describe the implementation
  – Does not specify algorithms, private data, or data structures
  – Preserves the module’s secrets

• One-to-many: one abstract module specification allows many possible implementations
  – Developer is free to use any implementation that is consistent with the interface
  – Developer is free to change the implementation

• “Abstract” does not mean “vague” or “imprecise”

Why these properties?

**Module Implementer**

• The specification tells me exactly what capabilities my module must provide to users

• I am free to implement it any way I want to

• I am free to change the implementation if needed as long as I don’t change the interface

**Module User**

• The specification tells me how to use the module’s services correctly

• I do not need to know anything about the implementation details to write my code

• If the implementation changes, my code stays the same

*Key idea:* the abstract interface specification defines a contract between a module’s developer and its users that allows each to proceed independently

© S. Faulk 2013
Interface Decisions

• There are two distinct considerations
  • Interface Design
    – Choices about precisely how the interface should be constructed to provide its services; specifically, which are the best abstractions to use
    – Choices about which kinds of decisions the interface should hide to limit dependencies
  • Interface Specification
    – How best to define the contract so it is clear and unambiguous to all parties
    – How to preserve the rationale for the design (why this is a good design)

Module Interface Design Goals

• General design goals addressed by module interface design
  – Support architectural goals: Independent work assignments, maintainability, understandability, testability, etc.
  – Addressed by two module interface design goals
  1. Control dependencies
     – Encapsulate anything other modules should not depend on
     – Hide design decisions and requirements that might change (data structures, algorithms, assumptions)
  2. Provide services
     – Provide all the capabilities needed by the module’s users
     – Provide only what is needed (complexity)
     – Provide problem appropriate abstraction (useful services and states)
     – Provide reusable abstractions
Applying Principles

• Most-solid first: make first those decisions least likely to change
  – Which design decisions are least likely to change?
  – Equivalently: would this decision be true of any implementation we will do? Would it be common to all applicable implementations?
• Information hiding: encapsulate decisions other modules should not depend on
  – Which decisions are likely to or do we want to be able to change?
• Abstraction
  – Which capabilities are essential to the problem that must be solved?
  – Which details are not necessary to the overall problem?

Example: Facilitator Module

• Which services should the interface provide and what kinds of abstractions should we use?
• Which decisions should the interface hide?
Documenting Design Decisions

- How do we capture important design decisions?
- What reasons support the idea our basic design will not change?
  - What is the basis for the abstraction?
  - What would any good design have in common?
- For each module:
  - What must the module provide other modules?
  - What must it hide?

Summary

- Goal: divide the development work among distributed teams such that
  - All teams are fully employed
  - Work assignments are clear
  - Dependencies, hence communication, are minimized
  - System works when integrated
- Each invokes potential risks
- Work breakdown and dependencies addressed by
  - Module structure architecture
  - Module interface specifications

© S. Faulk 2022
Architecture Meeting

• Prepare and agree on an agenda in advance
• Choose a moderator if possible
• Goals
  – Identify where requirements inconsistencies are causing issues. Identify who should resolve open questions (e.g., customer)
  – Identify/resolve remaining inconsistencies in module structure, e.g. assumptions about responsibilities or flow
  – Identify/resolve detailed inconsistencies in module interface specifications
  – Create a list of action items and due dates if possible. Otherwise choose people to do this.
• This is a very American view of how a meeting should progress, may not work

Questions