Advanced Software Architecture

Lecture – Managing Assumptions

Professor Jörgen Hansson
Department of Computer Science and Engineering
Chalmers University of Technology
jorgen.hansson@chalmers.se

Lecture Outline

- Garlan’s work
  - Experiences
  - Recommendations
- Lago and Vliet’s approach
Literature Sources

- Lago and Vliet, 'Explicit Assumptions Enrich Architectural Models’ (mandatory)
- Garlan, Allen, and Ockerbloom, 'Architectural Mismatch: Why Reuse is so Hard’ (mandatory)
- Garlan, Allen, and Ockerbloom, 'Architectural Mismatch: Why Reuse is Still so Hard’ (mandatory)
- Ostacchini and Wermelinger, 'Managing assumptions during agile development’.

Other (not mandatory)
- King and Turnits, 'The Landscape of Assumptions’
- Steingruebl and Peterson, 'Software Assumptions Lead to Preventable Errors’
- Lehman, 'The Role and Impact of Assumptions in Software Development, Maintenance and Evolution’

Experiences from AESOP

- **Purpose**: An implementation platform for experimenting with architectural development environments. AESOP would generate an environment based on a described set of architectural styles. The environment would be tailored to the development of the systems.

- AESOP produces the environment by combining (i) a set of style definitions with (ii) shared infrastructure.
  - Shared infrastructure is incorporated into each environment
  - The elements of style definitions are descriptions of
    - (i) architectural design vocabulary
    - (ii) visualization of design elements suitable for manipulation by a graphical editor
    - (iii) a set of architectural analysis tools to be integrated with the environment
Experiences from AESOP cont’d

• Infrastructure was intended to be built through reuse of
  – OBST – an object-oriented database
  – InterViews – a toolkit for constructing graphical user interfaces
  – SoftBench – an event-based tool integration
  – MACH RPC Interface Generator (a.k.a. MIG).

• Estimated work: one man-year over six months
• Reality: five man-years over two years, i.e., five times more and over a four times longer time span than estimated.

• What happened?

Reality Check

The main difficulties were all related to integration problems!!!! Specifically,

• Excessive code
  – Binary code of user interfaces > 3Mb after stripping
  – Database server was 2.3Mb after stripping

• Poor performance
  – Tool-to-Database communication, e.g., saving a small architectural diagram took minutes
  – Excessive code contributed poor performance
  – Startup-times took several minutes
  – No shared libraries in OS

• Need to modify external packages
  – E.g., client-event loop in SoftBench to work with InterView
  – Reverse-engineering memory allocation routines in OBST to communicate object handles to external tools
Reality Check cont’d

- Need to reinvent existing functions
  - E.g., bypass InterView’s support for hierarchical data structures; build separate transaction mechanism.

- Unnecessarily complicated tools
  - The need only required architectural tools (on top of the infrastructure) to be logically simple sequential programs
  - Tools’ standard interfaces to environment required them to handle multiple, independent threads of computation simultaneously.

- Error-prone construction process
  - Automated build routines were no longer applicable due to the significant changes

What were the root causes?

All of the problems could be attributed to architectural mismatch, specifically conflicting assumptions among the parts!!!

For purposes of reasoning, consider
(i) View the system as a configuration of components and connectors
(ii) Components are computational and storage entities of the system
(iii) Connectors determine interactions between components

Four main categories of assumptions that can contribute to architectural mismatch. These form a taxonomy.
Garlan et al.’s Taxonomy

Nature of components
- Includes assumptions about the substrate on which the component is built (infrastructure)
- Which components will control the computation sequencing (control model)
- The way the environment will manipulate the data managed by a component (data model)

Nature of connectors
- Assumptions about patterns of interaction characterized by a connector (protocol), and
- About the kind of data communicated (data model)

Garlan et al.’s Taxonomy cont’d

Global architectural structure
- Assumptions about the topology of the system, and
- About the presence or absence of particular components.

Construction process
- Assumptions related to, e.g., the order in which components and connectors are produced by instantiating a generic building block
Garlan et al.’s take on “the way forward”

- Provide techniques for bridging mismatches
  - Modifying components and connectors
  - Multiple versatile components and connectors to (i) replace functions and responsibilities of original architectural elements, or (ii) act as mediators between original elements.
  - Wrappers around components and connectors to provide a new interface
    - Negotiated interfaces
- Develop sources of architectural design guidance
  - Find ways to codify and disseminated principles and rules for software composition

1994 vs. 2009

- Today’s systems routinely build on many layers of reusable infrastructure (distributed data access), interact with users through standard interfaces (e.g., web browsers), and use open source software.
- Development environments improved
  - Direct access to reuse libraries (e.g., eclipse)
  - Exploit services created in a global virtual environment
Garlan et al.’s take on “the way forward”

- Make architectural assumptions explicit
  - Problem: architectural assumptions of reusable components are never documented
  - Vocabulary is missing; no convention or systematic approach to document architectural assumptions
  - ADLs represent a promising approach
  - Formal foundations and advances for software architectures are helpful

- Construct large software pieces using orthogonal components
  - Architectural design decisions are often spread throughout the system

Distinct Techniques to Avoid Architectural Mismatch

- Prevent it
  - New developments in architectural specifications (e.g., ADLs), open source practices, and virtualization, and common user interfaces

- Detect when it occurs (early detection is key, allows alternatives)
  - New developments include documentation standards (e.g., “views”), ADLs, standard interface description languages (e.g., SOA and WSDL).
  - Process guidance: increase adoption of spiral models and iterative risk-driven development practices; early architectural prototypes advocated

- Repair when unavoidable
  - E.g., use of Wrappers, adapters, mediators, and bridges
  - New frameworks provide built-in mechanisms such as protocol and data adaptors to integrate legacy components services (that would not have worked otherwise)
New challenges

Systems must support dynamic reconfiguration and adaptation to respond to component failures, variable resources, new user requirements.

Architecture evolution – systems are long-lived. New components must interoperate with old components and connectors; standards and frameworks must change.

Assumptions in Architectural Models
by Lago and Vliet

• It is meaningful to model “architectural variability”, as well as “architectural invariability”, i.e., assumptions about system and environments being made.

Lago and Vliet see three important uses:
• Traceability – tracing assumptions to design and implementation solutions; helps during development and evolution.
• Assessment – what happens if a certain assumption is invalid. (c.f. use of “change cases” to assess the system’s modifiability)
• Knowledge management – model of architecture and its assumptions (in the same way one would model architectural invariants). “externalizes architectural knowledge”; can be reused.
Assumptions vs Requirements vs Constraints

-- an informal distinction (by Lago and Vliet)

- “Requirements are demands on a system, and can be both functional and non-functional”
- “Constraints are limitations on a system”
- Requirements and constraints are conscious desiderata, i.e., defined and intended characteristics of the system.
  - Normally stated by client, user, domain expert.
  - Almost always stated explicitly.
- “Assumptions are the reasons for the architectural decisions that are made arbitrarily….. Made by the developer or the architect of the system…. never made explicit”
- **Hansson’s** view:
  - Assumptions, explicit or implicit, describe conditions that must hold in order for the system and its architecture to function as intended. The assumptions support the system, and if the assumptions are invalid, the system would be outside its operational envelope.

Types of Assumptions

- **(Technical)** assumptions – regards technical environment in which a system is going to run, e.g., programming languages, database systems, operating systems
  - **Technical** environment
  - **Architecture** assumptions
- **Organizational** assumptions – reflect company’s settings and principles, e.g., work-flow, organization structure reflected in development teams etc. Refer to the organization developing or using the system
- **Managerial** assumptions – reflect decisions to achieve business objectives, and concern solutions and operational tasks to achieve organization-level objectives.
  - E.g., mgmt strategies and plans, experience and knowledge base brought into projects etc)
An Experiment to make Assumptions Explicit (Lago and Vlet)

- **Input**: (i) feature models/architectural views of an existing software product (in a product family), (ii) list of identified implicit assumptions for the product family.
- **Activity**: Modeled assumptions with their dependencies on product features.
  - Features – user-visible aspects or characteristics of a system
- **Focus**: Represent the assumption concerning completeness of base technology
- An assumption is considered crosscutting if they affect more than one feature or one structural elements

Cont’d: Approach

- Identify the set of features directly influenced by the assumption
  - Assumption – execution environment is completely specified. This is true if and only if all features are properly configured, i.e., in this case the programming environment (JVM), the distributed communication platform (CORBA), communication technology and telephony networks
- Define dependencies between assumptions and feature model
  - Association \textit{F-Impacts} defines the dependency between assumption and set of generic features involved.
    - Base technology impacts the features in the provided example
  - Association \textit{S-impacts} defines the dependency between assumption and structural elements, e.g., interfaces and components.
  - Association \textit{Realizes} denotes which features needed to realize the assumption
    - E.g., Base technology is realized through resources needed to route
Schematic Description
(also referred to as meta-model)

- Assumption category
  - made of
  - realizes
  - aspect
  - refined into

- Feature
  - <<traces>>
  - provided by

Realizes shows which features needed to fulfill assumption.
Cont’d: Observations and Experiences

- To properly capture assumptions required more features to be included in existing feature models
  - Documenting assumptions provide additional insight into the architecture and system
- Assumptions may belong to different abstraction levels, e.g., consider F-impacts and S-impacts. Features concern “functional entities”; structural entities are captured in architectural view
  - -> assumptions should be captured in multiple architectural views