Software Architecture Maturation

Charles Shelton
17-939
November 7, 2001

Software Architecture: The Next Abstraction

- **Software Architecture**
  - Large software systems are exponentially more complex
  - Humans need abstraction principles to build large software systems

- **Humans handle complexity with abstraction**
  - Digital circuits ->
  - Logic gates ->
  - Computer architecture (processors) ->
  - Assembly language ->
  - High level languages ->
  - *Software architecture?*

The Problem is Recognized

  - Structured programming alone is insufficient for large software systems
  - Proposed solution: Separate structural composition problem from functionality with a separate Module Interconnection Language (MIL)

- **1984 – Modular Decomposition of A7E Avionics System (Parnas)**
  - Modular decomposition of a large system results in an explosion of modules
  - Proposed solution: Maintain system structure in a well-documented module guide to capture all module relationships

- **1989-90 – Aha! The problem is software complexity!**
  - Software system design requires a new abstraction principle dealing with higher-level system structure
  - *Software Architecture*

Ad-hoc Software Architecture

- **System designers already used some form of abstraction**
  - How we got any large working software systems at all
  - “Box-and-Line” diagrams with text
  - Vehicle for communicating design decisions
  - **Problem:** Limited connection to lower-level system design beyond the designer’s particular thought processes

- **Virtuoso designers**
  - Brooks advised one lead system architect for a large project
    - There must be one consistent high-level vision for the system
    - Avoid accidental complexity associated with system integration
  - **Problem:** Virtuoso designers are neither plentiful nor cheap
**Architectural Styles**

- Ad-hoc architectures lead to patterns that consistently work
- Patterns become well-known and informally understood
  - Pipes and filters
  - Object oriented
  - Layers
  - Publish-Subscribe
  - Blackboard
  - Client/Server
- No formal definition of architectural styles, but “everyone knows what you mean”

**Research Directions**

- Ultimate goal for software architecture as an abstraction:
  - A standard (graphical) representation (human understanding) that maps to a formal specification (transformation to lower-level) that facilitates quantitative and qualitative analysis (high-level evaluation)
- Formalizing the abstraction principles
  - Unambiguous representation of components and connectors
  - Formal style definitions
  - Generating source code from high-level abstraction
  - Architecture Description Languages (ADLs)
- Standardizing architectural representation
  - Graphical representation (UML)
  - Documenting design decisions
  - Communication tool for implementers
- Architectural Analysis (not discussed)
  - Qualitative – Architectural Tradeoff and Analysis Method (ATAM)

**Architecture Description Languages**

- Embody formal architectural abstraction principles in a language
  - Primitives for components and connectors
  - Formal style representation
  - Compilable
  - Formal analysis support
  - Explicitly deal with system structure and composition properties
- Outgrowth of Module Interconnection Languages (MILs)
  - MILs primarily for configuration and integration support
  - Separated system structure from functionality
  - Connectors not recognized as more than “glue code”

**Architectural Representation**

- Multiple views
  - Kruchten’s 4+1 Views
  - Soni et al. found multiple views in industrial projects
  - Issues:
    - Relating the different views
    - Which views are appropriate for which system aspects
- ADL tool support
  - Graphical tools that map icons to ADL primitives
  - ADL’s don’t agree on abstraction principles
- UML
  - Map architectural concepts to UML notation
  - No semantic support
**Current Industry Practice**

- *Still ad-hoc, but focused ad-hoc*
  - Software architecture recognized as an important activity crucial to system construction
  - Industry resists high-learning-curve formal techniques (ADL’s)
  - UML becoming de facto standard graphical representation technique for system design, but not a perfect fit for architecture representation
  - IEEE has a recommended practice document for software architecture

- *Anecdotal evidence*
  - Recently talked with industry person using UML tools for architecture

**Architectural Frameworks**

- *Middleware*
  - Distributed systems need a well-defined communication abstraction
  - Imposes constraints on the architecture
    - CORBA – client/server, object oriented
    - Java RMI – object oriented
    - Jini – publish/subscribe

- *Product Families*
  - Elements of similar systems recognized and organized into a knowledge base
  - Usually within a company that does a certain system, not shared
  - New product architecture gets “instantiated” from family architecture constraints

**Conclusions**

- *Software architecture has become a key focus for large software systems*
  - Research has become mainstream
  - Industry has recognized need for architecture representations, but not the formal abstraction principles

- *Software architecture research state-of-the-art*
  - Formalized architecture descriptions (ADLs)
  - Consistent representations
  - Mapping representations to formalisms

- *Industry practice*
  - Consistent representation and documentation (UML)
  - Informal style/pattern use
  - Architectural frameworks

- *Challenges to further maturation*
  - Standardizing and exploiting formal architecture descriptions
  - Unified model and mapping for multiple architecture views
  - Industry acceptance of support tools