

Software Architecture

Perspectives on an Emerging Discipline

CS 532 Software Design

Learning Objective

- . . . to give an appreciation of *Software Architecture* as an emerging and important facet of the upstream portion of the *Software Life Cycle*. As a phase that comes after requirement elicitation/specification and before *Software Design*, it's an important tool/discipline useful to the software engineering practitioner.
-

Frederick T Sheldon

Assistant Professor of Computer Science
University of Colorado at Colorado Springs

Emerging Issues of Architectural Design

⊗ What is Software Architecture?

⊕ *Structural and organizational issues about systems*

- Global control, communication protocols, synchronization and data access
- Allocation of resources (e.g., function → design elements)
- Design element composition (info hiding, coupling, cohesion)
- Physical distribution
- Scaling and performance
- Dimensions of evolutions
- Selection among design alternatives

Example Architectures

- ⊗ Client - Server model
- ⊗ Remote Procedure Call (RPC) structuring
- ⊗ Abstraction layering
- ⊗ Distributed Object Oriented approach
- ⊗ Pipeline - Filter framework

Architecture Patterns

- ⊗ Collection of idioms, patterns, and styles of software system organization that serves as a shared, semantically rich vocabulary
- ⊗ Example Pipelined Architecture:
 - ⊕ Streamed transformation
 - ⊕ Function behavior can be derived compositionally from the behavior of constituent filters

Frameworks for Understanding

- ⊗ Software architecture structures serve as frameworks for understanding the big picture (broader issues):
 - ⊕ System level concerns
 - ⊕ Global flow rates, patterns of communication
 - ⊕ Executive control structure, scalability
 - ⊕ System evolution
- ⊗ Properties can be fleshed out

Architecture of Software Systems

⊗ Defines the system in terms of computational components and interactions among the components

⊕ Components are...

- Clients / Servers
- Databases
- Filters
- Layers in a hierarchy of elements/components

Software Design Levels

(See Fig. 1.1)

⊗ Architecture

⊕ Overall association of system capability with components

⊗ Code

⊕ Algorithms, data structures, language primitives, etc...

⊗ Executables

⊕ Memory maps, stacks, register allocations, ISAs

⊗ Problem is, SW is understood at the level of

⊕ Intuition

⊕ Anecdote

⊕ Folklore

One Possible Solution

⊗ Improve the precision of understanding at the SW Architecture level

⊕ Programs, modules, systems

- Rich collection of *interchange representations* and protocols to connect components and system patterns to guide the compositions

An Engineering Discipline for Software

⊗ What is engineering?

- ⊕ Creating cost effective solutions . . .
- ⊕ . . . to practical problems . . .
- ⊕ . . . by applying scientific knowledge . . .
- ⊕ . . . Building things . . .
- ⊕ . . . In the service of mankind.

Engineering Is . . .

- ⊗ Relies on codifying scientific knowledge about a technological problem domain
- ⊗ Provides answers for common questions that occur in practice
- ⊗ Engineering shares prior solutions rather than relying on virtuoso problem solving

Routine and Innovative Design

- ⊗ Routine design involves solving familiar problems
 - ⊕ Reusing portions of prior solutions.
- ⊗ Innovative design involves finding novel solutions to unfamiliar problems.
- ⊗ Software in most application domains is treated more often as original than routine...
 - ⊕ Certainly more so than would be necessary if we captured and organized what we already know!

Model for Evolution of Engineering

(Fig. 1.2)

- ⊗ Engineering emerges from the commercial exploitation that supplants craft
- ⊗ Exploiting technology depends on . . .
 - ⊕ Scientific engineering
 - ⊕ Management
 - ⊕ Marshaling of resources
- ⊗ Engineering must return workable solutions!

Maturity of Supporting Science

⊗ Research on ADTs:

- ⊕ Specifications (abstract models and algebraic axioms)
 - ⊕ Software Structure (bundling representations with algorithms)
 - ⊕ Language issues (protecting integrity of information not in specifications)
 - ⊕ Integrity constraints (invariants of data structures)
 - ⊕ Rules for composition (declarations)
- ⊗ The whole field of computing is only 40yrs old... many theories are emerging in the research pipeline

Interaction Between Science and Engineering

(Figure 1.3)

⊗ Models and theories

⊕ Improved practice

⊕ New problems

- Ad hoc solutions

- Novel solutions

⊗ Folklore

⊕ Codification

⊕ Models and theories

Evolution of Software Engineering

(Figure 1.4)

- ⊗ Where does current SE practice lie o the path to engineering?
 - ⊕ In some cases it's a craft
 - ⊕ Yet in others it's a commercial practice
 - ⊕ And, in isolated examples, one could argue that professional engineering is taking place!

Codification Through Abstract Mechanisms

- ⊗ Conversion from an intuition (i.e., get the data structure right [ADT]) to a theory involve understanding the following:
 - ⊕ The software structure (a representation packaged with its primitive operators)
 - ⊕ Specifications (mathematically expressed as abstract models or algebraic axioms)
 - ⊕ Language issues (modules, scope, user-defined types)
 - ⊕ Integrity of the result (invariants of data structures and protection from other manipulation)
 - ⊕ Rules for combining types (declarations)
 - ⊕ Information hiding (protection of properties not explicitly included in specifications)

Status of Software Architecture

The Bad News...

- ⊗ SW Architects have been unable to exploit commonalties in system architectures
- ⊗ Make principled choices among design alternatives
- ⊗ Specialize general paradigms to specific domains
- ⊗ Teach their craft to others

Status of Software Architecture

The Good News...

- ⊗ The issues and problems are being addressed in such areas as...
 - ⊕ Module interface languages (MIL)
 - ⊕ Domain specific architectures
 - ⊕ Software reuse
 - ⊕ Codification of organizational patterns for SW
 - ⊕ Architecture description languages
 - ⊕ Formal underpinnings for architectural design
 - ⊕ Architectural design environments

Some Open Problems

- ⊗ Choosing the appropriate architecture for a given problem or domain
 - ⊕ Rules of style - dictate how to package components. But,...
 - ⊕ Interfaces make incompatible assumptions
 - E.g., in UNIX sort is available as a filter and a procedure.

Open Architectures

- ⊗ Some architectures are carefully documented and widely disseminated
 - ⊕ ISO's interconnection reference model
 - ⊕ NIST/ECMA Reference model (PCTE)
 - Generic SEE framework
 - ⊕ X-Windows (distributed Window I/F architecture)
 - Based on event triggering and callbacks