Overview

We aspire to a clean separation of specification from implementation, but pragmatics often induce coupling

- Separating specification from implementation
  - Idea emerged in 70’s, motivated variously by
    - Contracting and waterfall development
    - Information hiding
    - Formalization
  - Reactions in the 80’s – the devil is in the details
    - Information isn’t truly independent
    - People don’t act that way
  - Some recovery in the 90’s, driven by
    - Improved formal techniques
    - Better control of complexity, e.g. inheritance

Papers

- DeRemer & Kron 76:
  Programming-in-the-large vs Programming-in-the-small
- Greenspan & al 82:
  Capturing more world knowledge in the requirements specification
- Nii 86:
  Blackboard Systems
- Nuseibeh & al:
  Expressing the Relationships between Multiple Views in Requirements Specifications
- Allen & Garlan 94:
  Formalizing Architectural Connections

Timeline
DeRemer/Kron 76: Programming-in-the-large

RS: Radical solution? Not quite, it’s a method/tool, not a solution
ET: Enhanced tool/method? Not quite, it was pretty radical
RT: Radical tool/method?

A radical method for programming-in-the-large is described, based on the development of a new language for describing module relationships. In comparison with the use of programming-in-the-small languages it offers improved abstraction, encapsulation, and checking, which have been demonstrated in preliminary tests, but it leaves a number of side effects to be addressed including the need for more sophisticated compiling systems, and the knowledge of different languages for large vs small programming issues.

- Question: Development method: Is there a systematic way to manage the complexity of large software systems?
- Result: MIL: Distinguish description of system organization from the code
- Validation: Example refuting the possibility of separation (both ways)

This was the seminal paper that made MILs a recognized research area.

Programming Language History

- Algol-class languages provide only “block structured” scope
  > Hierarchically nested blocks
  > Name visibility in your block and higher
  > All names in a higher block visible in all its sub-blocks
- Data abstraction languages rejected this nesting in favor of blocking the implementation of a data abstraction from accessing local variables of the environment that defined it
  > Explicit enumeration of component dependencies
  > Language simply describes distribution of access to resources

Greenspan 82: Capturing world knowledge

EM: Enhanced model (but might be Radical model)

Existing software requirements models are deficient in dealing with conceptual entities and nonfunctional properties of representing real-world requirements. An enhanced software requirements [model] is described, capable of providing more accurate analyses / predictions of conceptual entities and nonfunctional properties in representing real-world requirements.

The model has been tested by comparing analyses / predictions with empirically measured values of conceptual entities and nonfunctional properties.

- Question: Development method: Is there a better way to capture real-world knowledge in requirements?
- Result: Model requirements via conceptual entities; capture “non-functional” requirements; use aggregation, classification, generalization
- Validation: Existence of tool; narrative assessment of methodology, underlying model, uniformity

Two read it: 1 ET, 1 EM

Nusibeh & al 93: Multiple Views in Specs

ET: Enhanced tool/method

The effectiveness of using multiple viewpoints and partial requirements specs in supporting the design of software systems has been demonstrated. An enhanced tool / method (that explicitly represents five aspects of each requirement element) is described for the design of software systems based on using multiple viewpoints and partial requirements specs. Examples are provided confirming the effectiveness of its support for using multiple viewpoints and partial requirements specs in design.

- Question: Development method: Is there a better way to represent requirements precisely?
- Result: ViewPoints, a representation technique for requirement methods and for elements of requirements, checking for relations between ViewPoints; lazy evaluation for consistency.
- Validation: Example of defining a method as ViewPoint templates and analysis of the mechanism for showing relations among ViewPoints

Two read it: 2 ET
A radical solution to the problem of problem solving with diverse noisy knowledge sources is described, based on a novel architecture, the blackboard. In comparison with implementations with normal control flow it offers the ability to adjust the order of computation to the state of the solution, which have been demonstrated in preliminary tests (actually, in several practical examples), but it leaves a number of side effects to be addressed including xxx. Strategies are suggested for addressing these side effects.

**Question:** Feasibility -- Is it possible to solve complex problems full of noisy data when you can’t predict the order of computation?

**Result:** Novel architecture, the blackboard

**Validation:** Implementation and evaluation: Discuss how examples run in prototype implementation

No one summarized this one (of the eight received by 1:00AM)

---

**The Blackboard Model**

- **Knowledge Sources**
  - World and domain knowledge partitioned into separate, independent computations
  - React to changes in blackboard

- **Blackboard Data Structure**
  - Entire state of problem solution
  - Only means by which knowledge sources interact to yield solution

- **Control**
  - Knowledge sources are self-activating

---

**Blackboard Architecture**

- **General framework to structure and control problem-solving behavior involving multiple, diverse, and error-ful knowledge sources**

- **Independent processes achieve cooperative problem-solving**
  - various levels of abstraction
  - limited processing allocated to most promising actions
  - diverse problem-solving components
  - focus-of-control mechanism

- **Diversity ==> search among multilevel partial solutions**
Blackboard Problem Characteristics

- No direct algorithmic solution
  - Multiple distinct kinds of expertise
  - Many options for what to do next
  - Heterogeneous domain vocabulary
- Uncertainty
  - Error and variability in both input & knowledge
  - Moderate to low signal-to-noise ratio in data
  - Uncertainty interferes with algorithmic solutions
- "Best-effort" or approximate solution often good enough
  - Find parts of a problem that can be solved separately
- Large factorable solution spaces
- Common applications involve uncertainty
  - Signal processing or interpretation
  - Problem-solving (e.g., planning)
  - Compiler optimization also a candidate

Problem-Solving Models

Central question: What pieces of knowledge should be applied, and when, and how?

- Backward reasoning:
  - Works from goal back to initial state
  - Example: program verification (deterministic)
- Forward reasoning:
  - Works from initial state toward goal
  - Example: expression simplification by transformation
- Opportunistic reasoning:
  - Works whichever direction seems most productive
  - Example: trig identities

Allen/Garlan 94: Formalizing Architectural Connections

ET: Enhanced tool/method

The effectiveness of architecture description languages in supporting the design of software system structures has been demonstrated. An enhanced tool/method (that formalizes the connector part of an ADL) is described for the design of software system structures based on architecture description languages. Examples are provided confirming the effectiveness of its support for architecture description languages in design.

- Question: Development method: Is there a more rigorous way to represent architectural connectors?
- Validation: Formal analysis
- Four read it: 1.5 ET, 1 RS, 1 “formal model”, .5 “specific solution”