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

- Swartout/Balzer 1982 (not 72): Inevitable Intertwining
- Lampson 1983/84: Hints on Computer System Design
- Parnas/Clements 1986: “Faking It”
- Balzer 1991: Tolerating Inconsistency
- Rosenblum 1992: Programming with Assertions
Swartout/Balzer 82: Inevitable Intertwining

ET: Enhanced tool/method

The effectiveness of separating specification from implementation in supporting the design of software systems has been demonstrated. An enhanced tool/method (namely making that separation less strict) is described for the design of software systems based on separating specification from implementation. Examples are provided confirming the effectiveness of its support for separating specification from implementation in design.

- Question: Development method: Is there a better way to relate specification to implementation?
- Result: Strict separation is naïve — sometimes implementation limits specification; other times implementation enables richer specification
- Validation: Example refuting the possibility of separation (both ways)

Swartout/Balzer 82: Inevitable Intertwining

EM: Enhanced model

Existing software development models are deficient in dealing with disengagement of specification from implementation. An enhanced software development [model] is described, capable of providing more accurate analyses/predictions of disengagement in [disengaged-specification from implementation designs]. The model has been tested by [comparing analyses/predictions with empirically measured values of disengagement] (actually, by showing existing model has problems via a counter-example, but not really enhancing the model).

- Question: Development method: Is there a better way to relate specification to implementation?
- Result: Strict separation is naïve — sometimes implementation limits specification; other times implementation enables richer specification
- Validation: Example refuting the possibility of separation (both ways)

Parnas/Clements 86: “Faking It”

ET: Enhanced tool/method

The effectiveness of a structured requirement-first process in supporting the design of software systems has been demonstrated. An enhanced tool/method (namely providing documentation that traces requirements to implementation but not enforcing top-down process) is described for the design of software systems based on a structured requirement-first process. Examples are provided confirming the effectiveness of its support for a structured requirement-first process in design.

- Question: Development method: Is there a better way to relate specification to implementation?
- Result: The major benefit of a rational process is internal organization and a set of work products (documentation) that allow people to act as if the process had been rational.
- Validation: Persuasion and 3-line reference to the A7 project

Lampson 83/4: Hints

XH: Experience and/or Heuristic

Studies reported here of system implementations supported by good designers generate a number of findings concerning design heuristics, including the hints given here. They indicate that criteria for good system design are met by the hints given here.

- Question: Characterization: What guidance do we offer for good system design?
- Result: Descriptive model: heuristics with the theme of focusing on the task at hand and respecting implementation considerations.
- Validation: Persuasion — in Brooks’ terms, these are rules of thumb, to be judged by their usefulness
- Eight read it: six “experience/heuristic” and two not classifying but identifying rules of thumb
Balzer 91: Tolerating Inconsistency

RS: Radical solution
A radical solution to the problem of data inconsistency is described, based on flagging inconsistencies rather than prohibiting them. In comparison with strict enforcement of specifications and invariants it offers the ability to allow the system to process unrelated data while deterring the use of the inconsistent portion of the data, which have been demonstrated in preliminary tests, but it leaves a number of side effects to be addressed including good ways to resolve the inconsistencies. Strategies are suggested for addressing these side effects.

• Question: Feasibility -- Is it possible to tolerate inconsistency without losing benefits of specification?
• Result: Technique based on guarded commands
Prototype implementations
• Validation: Implementation and evaluation: Discuss how example runs in prototype implementation
• No one summarized this one (of the eight received by RAM)

The argument of the paper

Before
A+B=C

• Pat and Lou must coordinate changes, or else ...

With Pollution Markers
If not A+B=C, flag problems!!

• Pat and Lou must coordinate changes, or else ...

Rosenblum 92: Programming with Assertions

EM: Enhanced model
Existing assertion characterization models are deficient in dealing with effectiveness of assertions of detecting faults. An enhanced (actually, new) assertion characterization is described, capable of providing more accurate analyses / predictions of effectiveness of assertions in detecting faults designs. The model has been tested by comparing analyses / predictions with empirically measured values of effectiveness of assertions.

• Question: Analysis method: Is there a better way to use assertions to detect faults?
• Result: Classification of assertions according to the kind of system behavior they are designed to capture
• Validation: Assertions used effectively in APP tool; coverage of Perry/Evangelist interface faults

Classification structures

• Aristotelian classifications
> principled, based on hierarchical taxonomies
> partition the space of things being classified
> don’t account for all the details of the real world
> often subverted by people charged with doing the classification

• Prototype-based classifications
> try to distinguish among a number of common cases
> often ad hoc, may not evolve gracefully

• Often have organizational or political significance
• Criteria
> There are consistent, unique classificatory principles in operation
> The categories are mutually exclusive
> The system is complete

Bowker & Star: Sorting Things
Classifications are Context-Sensitive

- Consistent finding of history of science: no such thing as natural or universal classification system. What appears natural in a given context may appear forced elsewhere.
- According to a Chinese emperor, Animals are divided into:
  - belonging to the emperor ➔ included in the present classification
  - embalmed ➔ frenzied
  - tame ➔ innumerable
  - sucking pigs ➔ drawn with a very fine camelhair brush
  - sirens ➔ et cetera
  - fabulous ➔ having just broken the water pitcher
  - stray dogs ➔ that from a long way off look like flies

Borges, cited in Foucalt 1970

Hierarchy vs Design Spaces

- Life
  - Plants
  - Animals
    - Invertebrates
      - Molluscs
      - Arthropods (jointed legs)
        - Crustaceans
        - Insects
    - Vertebrates
      - Reptiles
      - Mammals
      - Ungulates
      - Primates
      - Birds

- Graphic compression
  - amount of compression
  - time to compress/uncompress
  - lossiness
  - assumed structure of data

Different substructure, distinctions along different branches
Global structure, distinctions; unconstrained combinations

Implications for Research Strategies

- Question, Result, and Validation are three dimensions of a design space (cf “facets”)
- Some items are hierarchically structured
- Need principle for generating the items in each dimension