**Design and Definition of Data Abstractions**

Mary Shaw
September MMI

http://spoke.compose.cs.cmu.edu/ser01/

---

**Overview**

*A central problem in software design and development is managing intellectual complexity of the product*

- Techniques for managing complexity:
  - Abstraction
  - Information hiding
  - Separation of concerns, divide and conquer
- Abstraction: suppressing details
  - Good abstraction: suppressing the right details
  - Right details: the ones that don’t matter at the moment
- Today’s papers:
  - Two Parnas papers on information hiding
  - Early Booch paper on objects, which build on information hiding idea
  - Shaw survey of abstraction evolution

---

**Software Technology Maturation Points**

| Major Technology Areas | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Software Engineering   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Compiler Construction   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Knowledge-Based System  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Metrics                |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Technology Concepts    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Modern Database Systems |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Knowledge Technology   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Data Models            |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Distributed Systems    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| NEW DATA TYPE          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Consolidated Technology|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|                           |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

---

**Transition Points for Abstract Data Types**

- Basic research ==> concept formation
  - 1968: formulation of information hiding
- Concept formation ==> development & extension
  - 1973: abstract data type models
- Development & extension ==> internal exploration
  - 1977: incorporation in programming languages
- Internal exploration ==> external exploration
  - 1980: incorporation in other technologies
- External exploration ==> Popularization
  - late 80’s: object models, C++, Java
Composable Software Research at Carnegie Mellon

**Growth in Data Abstraction Granularity**

- 1950: Programming-any-which-way
- 1960: Programming-in-the-small
- 1970: Programming-in-the-large
- 1990: Programming-in-the-world
- 2000: architectural chunks
- 1990: packages
- 1980: abstract data types
- 1970: generic definitions
- 1960: data structures
- 1950: types in languages
- 1950: mnemonics, macros

**Growth in Abstraction Granularity**

- 1950: Programming-any-which-way
- 1960: Programming-in-the-small
- 1970: Programming-in-the-large
- 1990: Programming-in-the-world
- 2000: architectural chunks
- 1990: packages
- 1980: abstract data types
- 1970: generic definitions
- 1960: data structures
- 1950: types in languages
- 1950: mnemonics, macros

**Growth in Specification Power**

- 1950: prose
- 1960: signatures
- 1970: formal syntax
- 1980: module interconnection languages
- 1990: software engineering
- 2000: formal semantics
- 1990: formal specifications
- 1980: algebraic and model specs for ADTs
- 1970: concrete complexity
- 1960: software engineering
- 1950: structured programming

**Approaching a Paper**

- What kind of problem is it solving?
- What’s the real-world setting or motivation?
- What’s the research model?
  - How is it related to the real world?
- How mature is the idea?
- What is the research hypothesis?
- What is the research strategy?
- What’s the result?
  - How does it solve the problem in the research model?
  - How does it satisfy the research hypothesis?
- Do you believe the result, and why? On what evidence?
- How does the result map to the real-world setting?
Finding the Thread of the Argument

Real World
Practical problem

Real World
Solution to practical problem

Research Setting
Idealized problem

Research Setting
Solution to idealized problem

Research product
(technique, method, model, system, …)

Parnas: On the Criteria (1972)

• Real-world problem: Modular decomposition allows modules to be written independently and systems to be composed from parts. This should lead to shorter development time, product flexibility, and better understandability. However, criteria for partitioning a system into modules are unstated/unclear but the criteria matter.

• Research problem: KWIC (Keyword in context) index: Accept a set of lines, create the interesting “circular shifts”, and sort the result.

• Experiment: Create two modular decompositions, one conventional and one based on information hiding. Compare properties of the two.

• Result: Information hiding wins on properties examined, but it may lose efficiency if tool support isn’t provided.

• Impact: Example suggests strategy applicable to system design -- localize design decisions. Also, indicates need to minimize intrusion of implementation in specifications. Note: this sets the stage for abstract data types (localize representation decisions) and lazy evaluation.

Parnas: On the Criteria

Key elements of the result

• Information hiding: Instead of organizing modules around processing steps, organize them around design decisions that are likely to change.

• Module interfaces: Simplify them -- functional interfaces are easier to maintain than data representations (which are likely to change).

• Specification content: Resist temptation to say unnecessary things in specification -- they often constrain implementation.
Composable Software Research at Carnegie Mellon

Finding the Thread of the Argument

Real World
Practical problem

Research Setting
Idealized problem

Real World
Solution to
practical
problem

Validation Task 2:
Does the result help to solve the practical problem?

Research Setting
Solution to
idealized problem

Validation Task 1:
Does the product solve the idealized problem?

Research product
(technique, method, model, system, …)

Parnas et al: Modular Structure (1985)

• Real-world problem: Software engineering research ideas don’t get used.
• Research problem: As nearly as possible like real world -- the A7E flight program, regarded as a good example.
• Experiment: Replicate existing system using academic ideas, especially information hiding. This is preliminary report, on the design.
• Result: Information hiding was applicable, after refinement.
• Impact: Realism of example, detail of specification, analysis of practical issues show relation to practice

Parnas et al: Modular Structure

Key elements of the result
• Three structures: Programmers must deal with 3 structures.
  > Module structure: decomposition into work assignments (groups of related programs) and assumptions each team may make about other modules
  > Uses structure: which programs require the presence of others
  > Process structure:
• Keep secrets: Localize decisions that may change; access functions provide more flexibility than data formats.
• Lessons:
  > Scalability: Many modules ⇒ need hierarchy and structured documentation
  > Info can’t always be hidden ⇒ “restrict” interfaces that expose info
  > Boundary between specification and implementation is slippery

A7E Module Structure

module hides

• Hardware-Hiding Module
  > Extended Computer Module # processors, instr set, concurrency
  > Device Interface Module characteristics of peripherals
• Behavior-Hiding Module
  > Function Driver Module rules governing values of outputs
  > Shared Services Module behavior that must be shared
• Software Decision Module
  > Application Data Type Mod data reps and code to manipulate them
  > Physical Model Module physical models and implementations
  > Data Banker Module how values are produced

Parnas et al: Modular Structure (1985)

• Real-world problem: Software engineering research ideas don’t get used.
• Research problem: As nearly as possible like real world -- the A7E flight program, regarded as a good example.
• Experiment: Replicate existing system using academic ideas, especially information hiding. This is preliminary report, on the design.
• Result: Information hiding was applicable, after refinement.
• Impact: Realism of example, detail of specification, analysis of practical issues show relation to practice

Parnas et al: Modular Structure

Key elements of the result
• Three structures: Programmers must deal with 3 structures.
  > Module structure: decomposition into work assignments (groups of related programs) and assumptions each team may make about other modules
  > Uses structure: which programs require the presence of others
  > Process structure:
• Keep secrets: Localize decisions that may change; access functions provide more flexibility than data formats.
• Lessons:
  > Scalability: Many modules ⇒ need hierarchy and structured documentation
  > Info can’t always be hidden ⇒ “restrict” interfaces that expose info
  > Boundary between specification and implementation is slippery

A7E Module Structure

module hides

• Hardware-Hiding Module
  > Extended Computer Module # processors, instr set, concurrency
  > Device Interface Module characteristics of peripherals
• Behavior-Hiding Module
  > Function Driver Module rules governing values of outputs
  > Shared Services Module behavior that must be shared
• Software Decision Module
  > Application Data Type Mod data reps and code to manipulate them
  > Physical Model Module physical models and implementations
  > Data Banker Module how values are produced

Parnas et al: Modular Structure (1985)

• Real-world problem: Software engineering research ideas don’t get used.
• Research problem: As nearly as possible like real world -- the A7E flight program, regarded as a good example.
• Experiment: Replicate existing system using academic ideas, especially information hiding. This is preliminary report, on the design.
• Result: Information hiding was applicable, after refinement.
• Impact: Realism of example, detail of specification, analysis of practical issues show relation to practice

Parnas et al: Modular Structure

Key elements of the result
• Three structures: Programmers must deal with 3 structures.
  > Module structure: decomposition into work assignments (groups of related programs) and assumptions each team may make about other modules
  > Uses structure: which programs require the presence of others
  > Process structure:
• Keep secrets: Localize decisions that may change; access functions provide more flexibility than data formats.
• Lessons:
  > Scalability: Many modules ⇒ need hierarchy and structured documentation
  > Info can’t always be hidden ⇒ “restrict” interfaces that expose info
  > Boundary between specification and implementation is slippery

A7E Module Structure

module hides

• Hardware-Hiding Module
  > Extended Computer Module # processors, instr set, concurrency
  > Device Interface Module characteristics of peripherals
• Behavior-Hiding Module
  > Function Driver Module rules governing values of outputs
  > Shared Services Module behavior that must be shared
• Software Decision Module
  > Application Data Type Mod data reps and code to manipulate them
  > Physical Model Module physical models and implementations
  > Data Banker Module how values are produced

Parnas et al: Modular Structure (1985)

• Real-world problem: Software engineering research ideas don’t get used.
• Research problem: As nearly as possible like real world -- the A7E flight program, regarded as a good example.
• Experiment: Replicate existing system using academic ideas, especially information hiding. This is preliminary report, on the design.
• Result: Information hiding was applicable, after refinement.
• Impact: Realism of example, detail of specification, analysis of practical issues show relation to practice

Parnas et al: Modular Structure

Key elements of the result
• Three structures: Programmers must deal with 3 structures.
  > Module structure: decomposition into work assignments (groups of related programs) and assumptions each team may make about other modules
  > Uses structure: which programs require the presence of others
  > Process structure:
• Keep secrets: Localize decisions that may change; access functions provide more flexibility than data formats.
• Lessons:
  > Scalability: Many modules ⇒ need hierarchy and structured documentation
  > Info can’t always be hidden ⇒ “restrict” interfaces that expose info
  > Boundary between specification and implementation is slippery

A7E Module Structure

module hides

• Hardware-Hiding Module
  > Extended Computer Module # processors, instr set, concurrency
  > Device Interface Module characteristics of peripherals
• Behavior-Hiding Module
  > Function Driver Module rules governing values of outputs
  > Shared Services Module behavior that must be shared
• Software Decision Module
  > Application Data Type Mod data reps and code to manipulate them
  > Physical Model Module physical models and implementations
  > Data Banker Module how values are produced

Parnas et al: Modular Structure (1985)

• Real-world problem: Software engineering research ideas don’t get used.
• Research problem: As nearly as possible like real world -- the A7E flight program, regarded as a good example.
• Experiment: Replicate existing system using academic ideas, especially information hiding. This is preliminary report, on the design.
• Result: Information hiding was applicable, after refinement.
• Impact: Realism of example, detail of specification, analysis of practical issues show relation to practice

Parnas et al: Modular Structure

Key elements of the result
• Three structures: Programmers must deal with 3 structures.
  > Module structure: decomposition into work assignments (groups of related programs) and assumptions each team may make about other modules
  > Uses structure: which programs require the presence of others
  > Process structure:
• Keep secrets: Localize decisions that may change; access functions provide more flexibility than data formats.
• Lessons:
  > Scalability: Many modules ⇒ need hierarchy and structured documentation
  > Info can’t always be hidden ⇒ “restrict” interfaces that expose info
  > Boundary between specification and implementation is slippery

A7E Module Structure

module hides

• Hardware-Hiding Module
  > Extended Computer Module # processors, instr set, concurrency
  > Device Interface Module characteristics of peripherals
• Behavior-Hiding Module
  > Function Driver Module rules governing values of outputs
  > Shared Services Module behavior that must be shared
• Software Decision Module
  > Application Data Type Mod data reps and code to manipulate them
  > Physical Model Module physical models and implementations
  > Data Banker Module how values are produced

Parnas et al: Modular Structure (1985)

• Real-world problem: Software engineering research ideas don’t get used.
• Research problem: As nearly as possible like real world -- the A7E flight program, regarded as a good example.
• Experiment: Replicate existing system using academic ideas, especially information hiding. This is preliminary report, on the design.
• Result: Information hiding was applicable, after refinement.
• Impact: Realism of example, detail of specification, analysis of practical issues show relation to practice

Parnas et al: Modular Structure

Key elements of the result
• Three structures: Programmers must deal with 3 structures.
  > Module structure: decomposition into work assignments (groups of related programs) and assumptions each team may make about other modules
  > Uses structure: which programs require the presence of others
  > Process structure:
• Keep secrets: Localize decisions that may change; access functions provide more flexibility than data formats.
• Lessons:
  > Scalability: Many modules ⇒ need hierarchy and structured documentation
  > Info can’t always be hidden ⇒ “restrict” interfaces that expose info
  > Boundary between specification and implementation is slippery
Composable Software Research at Carnegie Mellon

**Parnas et al: Modular Structure**

- **Problem type:** Method (Is there a better way to do module decomposition?)
- **Research model:** Minimize difference between practical and research setting; demonstrate feasibility of method on real problem.
- **Hypothesis:** Research results are slow to be adopted because it's too risky to use unproven techniques; research examples are too unlike real ones; research ideas are not fully worked out.
- **Strategy:** Step up to a real example.
- **Validation:**
  - **Criteria:** simple modules; independent implementations; cost of change related to likelihood; major changes should be sets of independent module changes; understandable specs; ability to find relevant modules; sets in hierarchy small enough to reason about.
  - **Evidence from the experiment:** Authors find method helps satisfy criteria.

**Booch: Object-Oriented Development (1986)**

- **Real-world problem:** Functional development doesn’t support information hiding, natural concurrency, response to problem changes.
- **Research problem:** Sea buoy: Collect and provide weather and navigation data. Also, cruise control: maintain speed of car.
- **Experiment:**
  - Review history and generalize
  - Work out example in object-oriented style. [For motivation, sketch two designs, one functional and one object-oriented, and compare the two.]
- **Result:** (a) synthesized model of objects; (b) example of object-oriented design.
- **Impact:**

**Booch: Object-Oriented Development**

**Common properties of object concept**
- **State:** Object has state.
- **Actions:** Object is characterized by actions that it suffers and that it requires of other objects
- **Classes:** Object is an instance of some (possibly anonymous) class
- **Name:** Object is denoted by name
- **Visibility:** Object has restricted visibility of and by other objects
- **Specification and implementation:** Object may be viewed either by its specification or by its implementation

**Booch: Object-Oriented Development**

**Problem type:** Improvement (Is there a better way to do module decomposition?)

**Characterization (What, exactly, do we mean by X ?)**
- **Research model:** History, survey, and example
- **Hypothesis:**
  - Object-oriented design is superior to functional decomposition,
  - Objects draw on prior results, and draw credibility from those results
- **Strategy:** Present an example and comment on it. Relate history to see how prior work contributes to object orientation.
- **Validation:** Qualitative observation and persuasive narration. For synthesis, paper cites other sources re the benefits of object-oriented design.
• Idea: Identify the methodological “signature” of the research and state major points in a predictable way
  > Existing model-type models are deficient in dealing with properties of solution strategy.
  > An enhanced model-type is described, capable of providing more accurate analyses/predictions of properties in solution strategy designs.
  > The model has been tested by comparing analyses/predictions with empirically measured values of properties.

Pro Forma Abstracts

• Idea: Identify the methodological “signature” of the research and state major points in a predictable way
  > Existing state models are deficient in dealing with dependability of object-oriented designs.
  > An enhanced state model is described, capable of providing more accurate analyses/predictions of dependability in object-oriented designs.
  > The model has been tested by comparing analyses/predictions with empirically measured values of dependability.

Pro Forma Abstracts

• Idea: Identify the methodological “signature” of the research and state major points in a predictable way
  > Existing state models are deficient in dealing with dependability of object-oriented designs.
  > An enhanced state model is described, capable of providing more accurate analyses/predictions of dependability in object-oriented designs.
  > The model has been tested by comparing analyses/predictions with empirically measured values of dependability.

• Risk: Filling in slots mechanically and missing major points.
• Suggestion: In addition to Newman’s skeleton, add a sentence describing the essence of the result:
  > (add between 2 & 3) This enhanced model adds probabilities of
    thereby enabling analysis of system security or dependability.