3.2 Structuring Complex Software with the Module Guide

Text for Chapter 3.2

[PCW85] Parnas, D. L., Clements, P. C., and Weiss, D. M. *The modular structure of complex systems*. IEEE Trans. Softw. Eng. **11**(3), 259–266 (Mar. 1985).

Information hiding; the modules to decompose into.

Additional Background for Chapter 3.2

[Lam88] Lamb, D. A. *Software Engineering: Planning for Change*. Prentice-Hall (1988).

Chapter 5: information hiding; the modules to decompose into.

Why the Gap Between Information Hiding in Theory and in Practice?

(before start of SCR project)

1. idea is impractical for real problems?
2. responsible managers unwilling to bet on unproven idea? (startup problem)
3. examples in papers too unlikely to practical problems?
4. idea needs refinement or extension for complex projects?
5. practitioners not intellectually capable of application?

Why the Gap Between Information Hiding in Theory and in Practice?

1. idea is impractical for real problems?
   - *no*
2. responsible managers unwilling to bet on unproven idea? (startup problem)
3. examples in papers too unlikely to practical problems?
4. idea needs refinement or extension for complex projects?
5. practitioners not intellectually capable of application?
   - *no*

Bridging the Gap

2. responsible managers unwilling to bet on unproven idea? (startup problem)
   - started *SCR project* as an example
3. examples in papers too unlikely to practical problems?
   - SCR: A-7E flight operational program is realistic
4. idea needs refinement or extension for complex projects?
   - see below

Structuring Complex Software Systems Into Modules

- *many* implementation decisions, *many* details
- therefore *many modules*
- $\leq$ 25 modules:
  - not difficult to know:
    - ▷ which modules affected by a change
    - ▷ whether coverage complete
  - careful inspection
- hundreds of modules??
  - information hiding alone does not work here!

Needed: the Software Module Guide Document

- tree-structured hierarchy
- additional goals by hierarchy and guide:
  - well-defined concern: easily find relevant modules without looking at all the others
  - number of branches at each node small enough such that designers can argue convincingly that
    - ▷ no overlapping responsibilities of submodules
    - ▷ all responsibilities of module are covered
  - again: understand responsibility of a module without understanding its internal design

The Software Module Guide Document

- how responsibilities are allocated among the major modules
- the criteria used to assign a particular responsibility
- scope and contents of the individual design documents
- large example will follow

When to Write the Software Module Guide

- start after SW behaviour specification (SOF) is complete
- refine top-level modules as concurrent work assignments
  - each refinement step renders more concurrent design work assignments
- the module interface specification writers work out the details
- the module internal design follows

Tracing Requirements

- software module guide derived from SW behaviour specification (SOF)
- easy to trace requirements to modules
- easy to trace back a design decision to the requirements

Access to a Module's Access Programs

- any program may use any access program of any module in the guide
  - independent of relative positions in hierarchy
  - but see also the "uses hierarchy" in Chapter 3.4 later on!

Module Interfaces May Change

- module interfaces are (higher-level) design decisions
  - may change
  - like module contents are design decisions
- encapsulate these interfaces in higher-level modules
- don't mention these sub-modules in guide
  - don't use sub-modules outside this module
- additional local module guide for this module

Difficulties During Structuring

- unstable information that cannot be encapsulated
  - → "restricted" modules
- need to locate "secret" modules in the guide
  - → "hidden" modules

Restricted Modules

- a problem:
  - we should confine information about hardware that could be replaced
  - diagnostic information about that hardware must be communicated to display modules
- restrict use of such modules
  - mark by "(R)" in module guide
  - try to avoid using restricted modules because of potentially high costs of change

Hidden Modules

- often: existence of certain sub-modules is a secret
  - not in the global guide
  - no use outside this module
- sometimes: existence of sub-module is a secret, but guide should clearly state where certain functionality is
  - mention these sub-modules in guide
    - ▷ mark by "(H)" as hidden
  - still no use outside the module

Two Kinds of Module Secrets

- primary secret
  - hidden information specified to the software designer
- secondary secrets
  - implementation decisions made by the designer when implementing

The Classes of Modules in the A-7E Software Module Structure

top-level decomposition:

1. *hardware-hiding* module } secret is in software requirements document
2. *behaviour-hiding* module } secret is in software requirements document
3. *software decision* module } secret is not a requirement

- this top-level decomposition is valid for *nearly all SW systems!*

- hardware-hiding module
  - any programs affected by replacing a device
    - ▷ with different interface
    - ▷ with same general capabilities
  - implements virtual hardware used by rest of software
  - even for "non-embedded" software
    - ▷ any programs affected by likely changes in the operating system
  - primary secrets:
    - ▷ the hardware-software interfaces described in the requirements document
  - secondary secrets:
    - ▷ data structures and algorithms used to implement the virtual hardware

- behaviour-hiding module
  - any programs affected by changes of the required behaviour
  - these programs determine the values to be sent to the "virtual hardware" output devices
  - primary secrets:
    - ▷ the required behaviour

- software decision module
  - hides software design decisions based upon
    - ▷ mathematical theorems
    - ▷ physical facts
    - ▷ programming considerations (efficiency, accuracy)
  - secrets and interfaces determined by software designers
    - ▷ secrets are *not* in the requirements document
  - likely reason for changes here:
    - ▷ improve performance
    - ▷ not: externally imposed changes

Fuzziness in the Top-Level Classification

1. line between requirements and design decided when requirements are written
   - example: requirements can specify an explicit weapon trajectory model or just accuracy requirements
2. line between hardware characteristics and software design
   - software tasks could be cast into hardware
   - software decision module or hardware-hiding module?

3. software design decisions may not be appropriate anymore because of changes in
   - the hardware
   - the behaviour of the system
   - the behaviour of its users
4. all software modules include software design decisions
   - changes in any module may be motivated by efficiency or accuracy considerations

- such fuzziness is not acceptable!

Eliminating Fuzziness in the Top-Level Classification

- by referring to a precise software requirements document
  - specifies the lines between behaviour, hardware, and software decisions

ad 1: line between requirements and design
- if requirements specifies algorithm: algorithm is not software design decision
- if requirements specifies constraints only: program that implements algorithm is part of software design decision module

ad 2: line between hardware characteristics and software design

- ○ interface specified in software requirements document
- ○ draw line based on likelihood of changes
  - ▷ if likely to cast this software in hardware:
    classify as hardware-hiding module
  - ▷ otherwise: software design module
- ○ conservative stance in SCR project:
  - ▷ drastic changes less likely than evolutionary changes
  - ▷ slight changes to hardware:
    hardware-hiding modules affected only
  - ▷ radical changes software→hardware:
    some software decision modules eliminated or reduced in size

ad 3: software design decisions may not be appropriate anymore because of changes in [. . .]

- ○ module only in software decision module if
  it remains useful even when requirements document is changed
  (although possibly less efficient)

ad 4: all software modules include software design decisions

- ○ module only in software decision module if
  its secrets do not include information from the requirements document