

**CSci 780**  
**Advanced Software Engineering**

---

**Designing Software for Ease of Extension  
and Contraction**

by David L. Parnas

9/29/2003

1

**Outline**

---

- Problem
- Consequences
- Problems Typically Encountered
- Program Families
- Solution: Design for Program Subsets
- Toward a Methodology
- The "Uses" Relation
- Example
- Summary

9/29/2003

2

**Problem**

---

- Changing requirements are a fact of life for long-lived software
- Most software systems live longer than expected
- Designing software to be easily changed is difficult

9/29/2003

3

**Consequences**

---

- Nothing works until everything works
- Small changes require lots of effort
  - Add a feature
  - Remove a feature (Why is this useful?)
- The resulting system is not what you would have designed from the start

9/29/2003

4

**Problems Typically Encountered**

---

- Excessive information distribution
  - Other modules depend on data which should not be exposed
- Pipeline architecture
  - Later filters depend on format of previous filters
- Components with more than one function
  - Some family members may not need both functions
- Loops in the "uses" relation
  - Nothing works until everything works

9/29/2003

5

**Program Families**

---

- Think in terms of a *family* of programs instead of a single program [Parnas TSE 1972]
- Identify commonalities and likely differences: hardware, I/O formats, available resources, etc.
- Structure the design choices such that the most wide-ranging ones are made first
- Exploit stable, common design decisions

9/29/2003

6

## Solution: Design for Program Subsets

- Identify the smallest feature-minimal system you can
- Identify a series of atomic changes which will add some bit of necessary functionality
- The earlier the change is, the more significant the design decision [Parnas TSE 1972]

9/29/2003

7

## Toward a Methodology (1/4)

- Identify the minimal subset and atomic changes
  - Part of identifying requirements
- Benefits:
  - Forces one to separate unrelated functionality in the design
  - Easier to create prototypes
  - "Grow don't build"
  - More flexible to changes

9/29/2003

8

## Toward a Methodology (2/4)

- Practice information hiding for module design
  - Identify secrets
  - Hide them in modules
  - Make interfaces insensitive to these changes
- Benefits: earlier paper
- Example: publish & subscribe architecture

9/29/2003

9

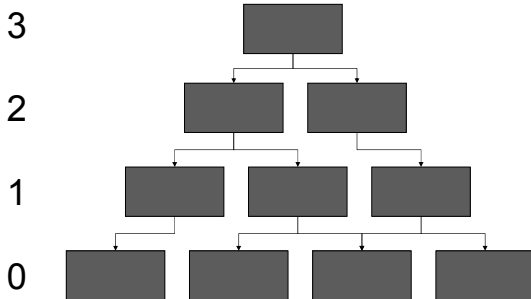
## Toward a Methodology (3/4)

- Break the imperative pipeline mentality
  - "First do this, then do this, then do this"
- Design each module to be a general solution, and use them as a virtual machine

9/29/2003

10

## Example: Virtual Machine



9/29/2003

11

## Toward a Methodology (4/4)

- Design the "uses" structure
  - A "uses" B if A depends on the correct execution of B
  - Avoid cycles in the "uses" relation
- Benefits:
  - Each level is a testable and usable subset
  - Can also treat parts of levels as a subset
  - Each level is a major milestone
  - Solves the "nothing works until everything works" problem

9/29/2003

12

## More on “Uses”

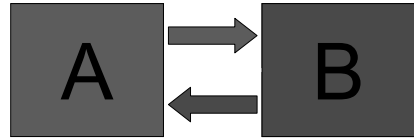
- “Uses” != “invokes”
  - A may invoke B but not use it: A’s spec only says B must be called
  - A may use B but not invoke it: interrupt handling
- Criteria for allowing A to use B:
  - A is essentially simpler because it uses B
  - B is not more complex b/c it is not allowed to use A
  - There is a useful subset containing B and not A
  - There is no useful subset containing A but not B

9/29/2003

13

## Breaking Cycles in the “Uses” Relation

- “Sandwiching”



9/29/2003

14

## Example: Address Processing

- Program to read in, store, and print addresses
- Data: name, address, pay grade, title, etc.

9/29/2003

15

## Subset Analysis

- All data processed by all family members
- Input and output formats likely to change
  - Some systems will use a fixed format
  - Some may need to change at runtime
  - Some may use a grammar to specify format
- Storage may differ from system to system
- Number of addresses in memory may differ

9/29/2003

16

## Determining Key Design Decisions

- Use tables to specify the I/O formats
  - Secret of input and output modules
- Changing part of an address should be cheap
  - Secret of address storage module (ASM)
- File storage support
  - Address file module (AFM) compatible with ASM

9/29/2003

17

## Determining Key Design Decisions (cont)

- Disk storage
  - Block file module (BFM) manages disk storage
- AFM implemented with BSM and ASM
  - ASM will have a “constructor” which takes block data

9/29/2003

18

## Overall Architecture

- ASM, AFM, BFM will serve as the low-level VM
- "Address Input", "Address Output", "Address Storage" are at the next level

9/29/2003

19

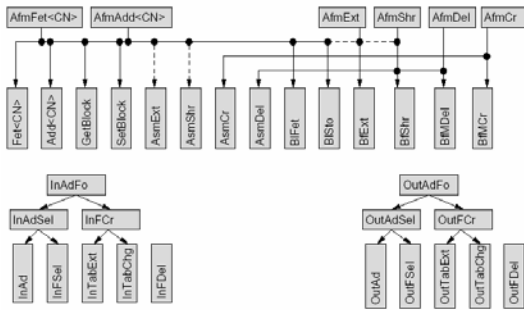
## Module Design (Example: Address Input)

- INAD: Reads in an address and calls ASM or AFM to store it
- INFSL: Select a format to be used by INAD
- ...
- Note: Parnas treats each function as a module

9/29/2003

20

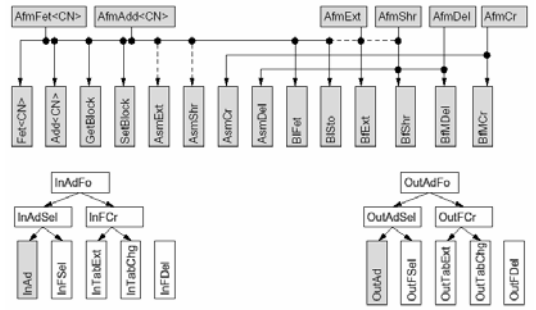
## Uses Relation



9/29/2003

21

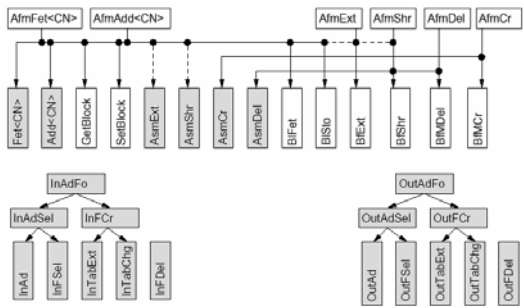
## Example: Single Address Format



9/29/2003

22

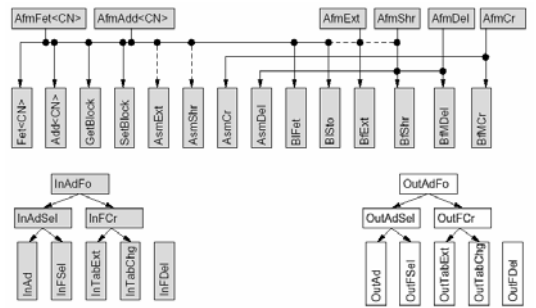
## Example: Small Address Set



9/29/2003

23

## Example: Query Only



9/29/2003

24

## Summary

---

- Design software to be more flexible
- Think in terms of a family of programs
  - Identify possible subsets during requirements
  - Use information hiding
  - Implement using VM approach
  - Make sure the "uses" relation is a DAG
- Order design decisions so that high-impact ones happen first

9/29/2003

25

## Some Thoughts

---

- Flexibility != generality
- Subsets now part of many methodologies
  - "Identifying the minimal subset is difficult because the minimal system is not usually a program that anyone would ask for." — Parnas
- 1991 ICSE most influential paper award
  - Parnas [1995] doesn't think such papers have practical impact (but do affect other researchers)

9/29/2003

26

## Discussion Questions

---

- Is it really wise to plan for changes which may never come to pass? Waste of money?
- Is a virtual machine the only way to break the pipeline mentality?
- True or false: Reuse hurts subsetting
- Does this approach scale?
- Does it work for OO systems?

9/29/2003

27