SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

07 Introduction to Modules (Ch. 4)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

January 18, 2018

07 Introduction to Modules (Ch. 4)

- Administrative details
- Unix command of the day: `ps`
- What is a software module?
- Program families
- Components of a module
- The module interface
- The module implementation
- Information hiding
- Examples of modules
- Example of a vector module
- Homework exercise on intersection

Administrative Details

- Assignment 1
  - Part 1: January 22, 2018
  - Partner Files: January 28, 2018
  - Part 2: January 31, 2018
- Questions on assignment?
- Rather than work with the files elsewhere and copy over at the last minute, you should work with the repo versions
- Frequent commits are a great habit to get into

Unix Command of the Day

- `ps` displays the currently running processes
- `ps -e -f` displays **e**very process in **f**ull output format
- You can use ps to find the process id
- You can use the process id to `kill` the process

```
[smiths@mills ] sleep 100 &
[smiths@mills ] ps
PID TTY TIME CMD
25493 pts/2 0:00 tcsh
27182 pts/2 0:00 sleep
27184 pts/2 0:00 ps
[smiths@mills ] kill 27182
```

What is Design?

- Provides structure to any artifact
- Decomposes system into parts, assigns responsibilities, ensures that parts fit together to achieve a global goal
- Design refers to
  - Activity
    - Bridge between requirements and implementation
    - Structure to an artifact
  - Result of the activity
    - System decomposition into modules (module guide)
    - Module interface specification (MIS)

Two Important Goals

- Design for change (Parnas)
  - Designers tend to concentrate on current needs
  - Special effort needed to anticipate likely changes
  - Changes can be in the design or in the requirements
  - Too expensive to design for all changes, but should design for likely changes
- Product families (Parnas)
  - Think of the current system under design as a member of a program family
  - Analogous to product lines in other engineering disciplines
  - Example product families include automobiles, cell phones, etc.
  - Design the whole family as one system, not each individual family member separately

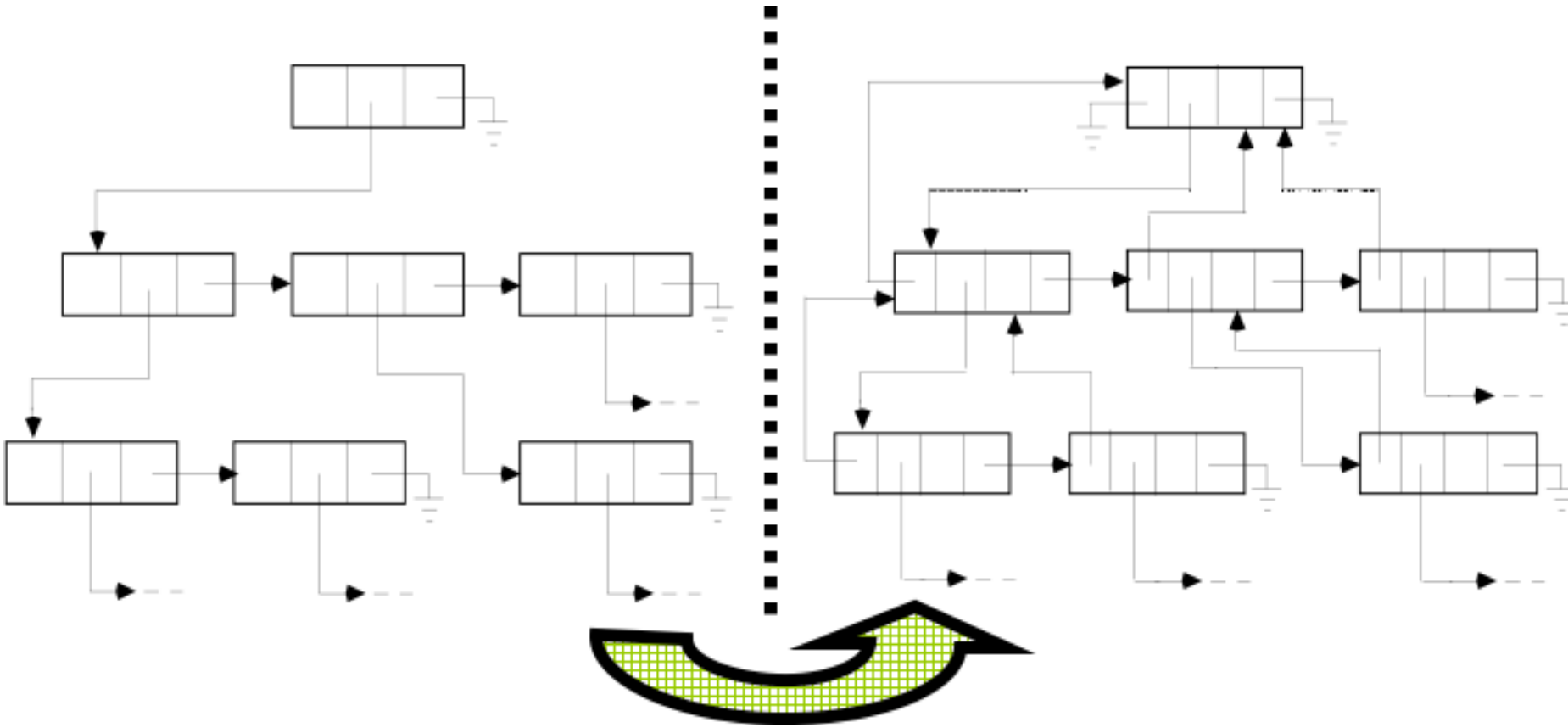
Sample Likely Changes

What are some common examples of likely changes for software?

Sample Likely Changes

- Algorithms – like replacing inefficient sorting algorithm with a more efficient one
- Change of data representation
  - From binary tree to threaded tree
  - Array implementation to a pointer implementation
  - Approx. 17% of maintenance costs attributed to data representation changes (Lientz and Swanson, 1980)
- Change of underlying abstract machine
  - New release of operating system
  - New optimizing compiler
  - New version of DBMS
  - etc.
- Change of peripheral devices

Binary Tree to Threaded Tree



Sample Likely Changes

- Change of "social" environment
  - Corresponds to requirements changes
  - New tax regime
  - EURO versus national currency in EU
  - New language for user interface
  - y2k
- Change due to development process (prototype transformed into product)

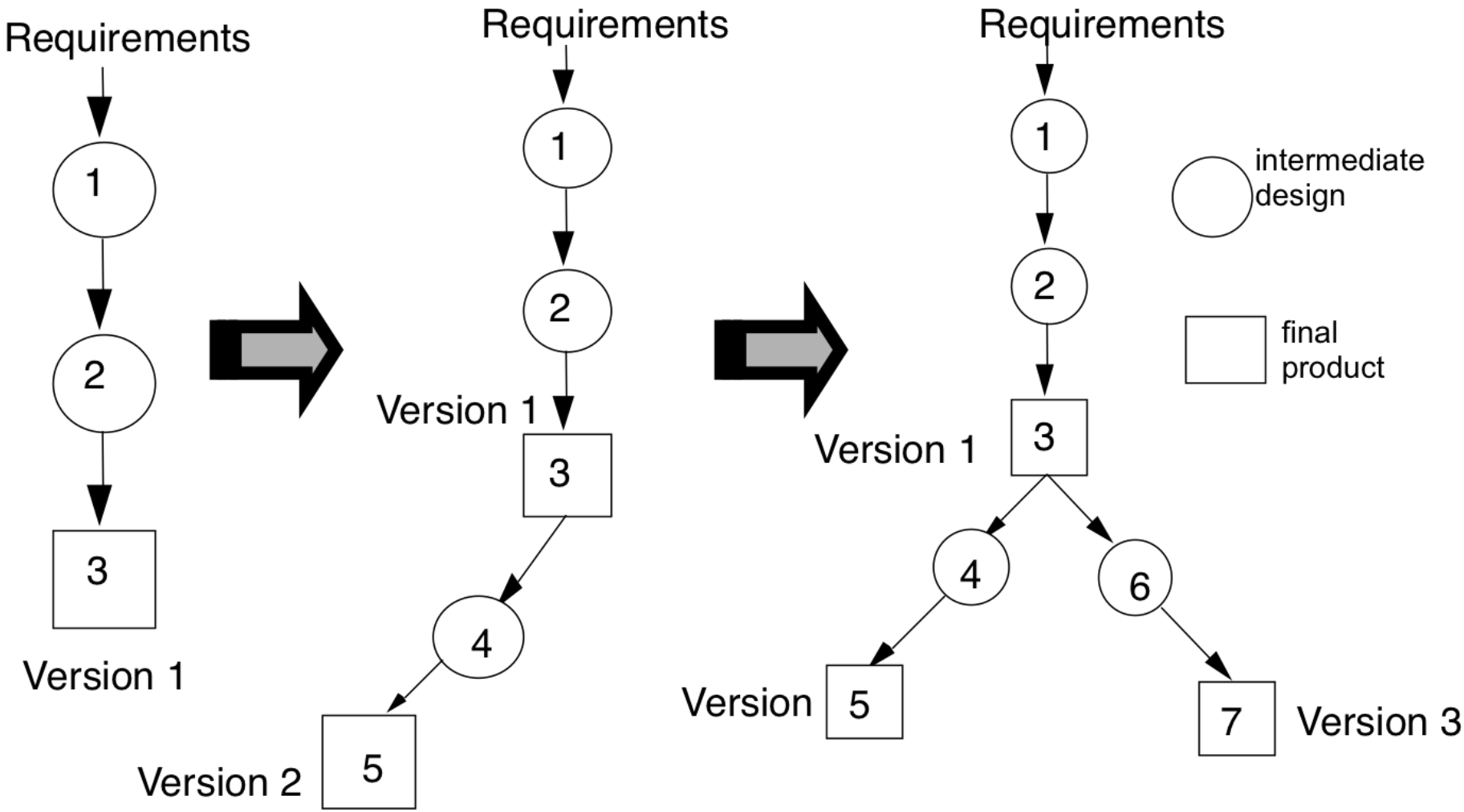
Product Family Examples

- What are some examples of product families?
- What are some examples of program families?
- Could Mosaic be part of a family of related programs?

Product Families

- Different versions of the same system
  - A family of mobile phones
    - Commonalities include use of communication, interface, screen, keyboard, etc.
    - Variabilities include different network standards, user interaction language, camera resolution, etc.
  - Facility reservation system
    - Commonalities include reserve something for a time period, user interface, etc.
    - Variabilities include context (hotel, university, etc), reserve space versus equipment, fee or not, etc.
- Design the whole family as one system, not each individual member of the family separately

Sequential Completion: The Wrong Way



Product Families

- Different versions of the same system
  - A family of mobile phones
    - Commonalities include use of communication, interface, screen, keyboard, etc.
    - Variabilities include different network standards, user interaction language, camera resolution, etc.
  - Facility reservation system
    - Commonalities include reserve something for a time period, user interface, etc.
    - Variabilities include context (hotel, university, etc), reserve space versus equipment, fee or not, etc.

How to Do Better

- Anticipate definition of all family members
- Identify what is common to all family members, delay decisions that differentiate among different members (variabilities)

Components of a Module

- A software modules has two components
  1. An interface that enables the module's clients to use the service the module provides
  2. An implementation of the interface that provides the services offered by the module

The Module Interface

- A module's interface can be viewed in various ways
  - As a set of services
  - As a contract between the module and its clients
  - As a language for using the module's services
- The interface is exported by the module and imported by the module's clients
- An interface describes the data and procedures that provide access to the services of the module

The Module Implementation

- A module's implementation is an implementation of the module's interface
- The implementation is hidden from other modules
- The interface data and procedures are implemented together and may share data structures
- The implementation may utilize the services offered by other modules

Information Hiding

- Basis for design (that is modular decomposition (Module Guide))
- Implementation secrets are hidden to clients
- Secret can be changed freely if the change does not affect the interface
- *Encapsulate changeable design decisions as implementation secrets within module implementations*

Examples of Modules

- Record
  - Consists of only data
  - Has state but no behaviour
- Collection of related procedures (library)
  - Has behaviour but no state
  - Procedural abstractions
- Abstract object
  - Consists of data (fields) and procedures (methods)
  - Consists of a collection of constructors, selectors, and mutators
  - Has state and behaviour

Examples of Modules Continued

- Abstract data type (ADT)
  - Consists of a collection of abstract objects and a collection of procedures that can be applied to them
  - Defines the set of possible values for the type and the associated procedures that manipulate instances of the type
  - Encapsulates the details of the implementation of the type
- Generic Modules
  - A single abstract description for a family of abstract objects or ADTs
  - Parameterized by type
  - Eliminates the need for writing similar specifications for modules that only differ in their type information
  - A generic module facilitates specification of a stack of integers, stack of strings, stack of stacks etc.

Example MIS for a Vector: Syntax

Access Routine Syntax

Routine name	Inputs	Outputs	Exceptions
Vect	real, real		
xcoord		real	
ycoord		real	
mul	real		
sum	real, real		
dot	real, real	real	
mag		real	
angle		real	
orthog	real, real	boolean	

Vector MIS Continued: Semantics

State variables

xc : real

yc : real

State invariant

none

Assumptions

Vect is called before any other access routine

Local Constants

TOLERANCE $= 1 \times 10^{-5}$

Vector MIS Continued

Access routine semantics

Vect(x, y):

- transition: $xc, yc := x, y$
- exception: none

xcoord():

- output: $out := xc$
- exception: none

ycoord():

- output: $out := yc$
- exception: none

Vector MIS Semantics Continued

mul(r):

- transition: $xc, yc := r \cdot xc, r \cdot yc$
- exception: none

sum(x, y):

- transition: $xc, yc := x + xc, y + yc$
- exception: none

Vector MIS Semantics Continued

dot(x, y):

- output: $out := xc \cdot x + yc \cdot y$
- exception: none

mag():

- output: $out := \sqrt{\text{s\_dot}(xc, yc)}$
- exception: none

angle():

- output: $out := \cos^{-1}\left(\frac{\text{s\_dot}(0,1)}{\text{s\_mag}()}\right)$
- exception: none

Vector MIS Semantics Continued

orthog(x, y):

- output: $out := (|\text{s\_dot}(x, y)| \leq \text{TOLERANCE})$
- exception: none

Homework: Formal Version of Intersect

In natural language, how would you describe the intersection of two circles?

How would you write a formal version of circle intersection to make it unambiguous? From A1-2017, how would you write the semantics for `intersect(c)`?