

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

Winter 2017

14 Mod Decomp Contd (Ghezzi Ch. 4, H&S Ch. 7)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 3, 2017



Module Decomposition

- Administrative details
- Relationship between modules
- The USES relation
- Module decomposition by secrets
- The IS_COMPONENT_OF relation
- Module guide

Administrative Details

- Assignment 2
 - ▶ Files due by 11:59 pm Feb 15
 - ▶ E-mail partner files by 11:59 pm Feb 16
 - ▶ Lab report due by 11:59 pm Feb 27
- Midterm exam
 - ▶ March 1, 7:00 pm, TSH/120
 - ▶ 90 minute duration
 - ▶ Multiple choice - 30–40 questions?
 - ▶ Open book (any paper)

Assignment 2

- Q redefines gravity
- Redefine gravity in games

$$F = \frac{G}{r^2} m_1 m_2 = f(r) m_1 m_2$$

Syntax Circle ADT Module Continued

| Routine name | In | Out | Exceptions |
|---------------------|-------------------------|----------------------------|-------------------|
| new CircleT | PointT, real | CircleT | |
| cen | | PointT | |
| rad | | real | |
| area | | real | |
| intersect | CircleT | boolean | |
| connection | CircleT | LineT | |
| force | real \rightarrow real | CircleT \rightarrow real | |

Access Routine Semantics Continued

`intersect(ci):`

- output:

$\exists(p : \text{PointT} \mid \text{insideCircle}(p, ci) : \text{insideCircle}(p, self))$

- exception: none

`connection(ci):`

- output: $out := \text{new LineT}(c, ci.cen())$

- exception: none

`force(f):`

- output: $out := \lambda c \rightarrow$

$self.area() \cdot c.area() \cdot f(self.connection(c).len())$

- exception: none

Syntax Deque Of Circles Module

| Routine name | In | Out | Exceptions |
|---------------------|-------------------------|------------|-------------------|
| Deq_init | | | |
| Deq_pushBack | CircleT | | FULL |
| Deq_pushFront | CircleT | | FULL |
| Deq_popBack | | | EMPTY |
| Deq_popFront | | | EMPTY |
| Deq_back | | CircleT | EMPTY |
| Deq_front | | CircleT | EMPTY |
| Deq_size | | integer | |
| Deq_disjoint | | boolean | EMPTY |
| Deq_sumFx | real \rightarrow real | real | EMPTY, POS |
| Deq_totalArea | | real | EMPTY |
| Deq_averageRadius | | real | EMPTY |

Semantics Deque Of Circles Module

State Variables

s : sequence of circleT

State Invariant

$$|s| \leq \text{MAX_SIZE}$$

Assumptions

`init()` is called before any other access program.

Access Routine Semantics

Deq_disjoint():

- output $out := \forall(i, j : \mathbb{N} | i \in [0..|s| - 1] \wedge j \in [0..|s| - 1] \wedge i \neq j : \neg s[i].\text{intersect}(s[j]))$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

What happens if s only holds one circle? Does this make sense?

Access Routine Semantics

Deq_sumFx(f):

- output

$$out := +(i : \mathbb{N} | i \in ([1..|s| - 1]) : Fx(f, s[i], s[0]))$$

- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Local Functions

$Fx: (\text{real} \rightarrow \text{real}) \times \text{CircleT} \times \text{CircleT} \rightarrow \text{real}$

$Fx(f, ci, cj) \equiv \text{xcomp}(ci.\text{force}(f)(cj), ci, cj)$

$\text{xcomp}: \text{real} \times \text{CircleT} \times \text{CircleT} \rightarrow \text{real}$

$$\text{xcomp}(F, ci, cj) \equiv F \left[\frac{ci.\text{cen}().\text{xcrd}() - cj.\text{cen}().\text{xcrd}()}{ci.\text{connection}(cj).\text{len}()} \right]$$

Access Routine Semantics Continued

Deq_totalArea():

- output

out := ?

- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Deq_averageRadius():

- output

out := ?

- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Details and Notes

- Doxygen, make, LaTeX, Python (2.7) and PyUnit
- Do NOT change the interface
- Can add `__methodName__`
- Makefile includes rule for doc
- Makefile includes rule for test
- Tag repo as A2Part1 and A2Part2
- Trading of code will be done automatically
- Python specifics:
 - ▶ FULL, EMPTY implemented via inheriting from Exception class
 - ▶ Exceptions should only be used with one argument, a string explaining what problem has occurred.
 - ▶ `Dec.accessProg`, not `Dec_accessProg`, as shown in the specification.
- Monitor all changes pushed to our repo

File and Folder Structure

- A2
 - ▶ doxConfig
 - ▶ Makefile
 - ▶ report
 - ▶ report.tex
 - ▶ report.pdf
 - ▶ src
 - ▶ pointADT.py
 - ▶ lineADT.py
 - ▶ circleADT.py
 - ▶ deque.py
 - ▶ testCircleDeque.py
 - ▶ srcPartner
 - ▶ circleADT.py

Questions

- What relationships have we discussed between modules?
- Are there desirable properties for these relations?

The USES Relation

- A uses B
 - ▶ A requires the correct operation of B
 - ▶ A can access the services exported by B through its interface
 - ▶ This relation is “statically” defined
 - ▶ A depends on B to provide its services
 - ▶ For instance, A calls a routine exported by B
- A is a client of B; B is a server
- Inheritance, Association and Aggregation imply Uses

Relationships Between Modules

- Let S be a set of modules

$$S = \{M_1, M_2, \dots, M_n\}$$

- A binary relation r on S is a subset of $S \times S$
- If M_i and M_j are in S , $\langle M_i, M_j \rangle \in r$ can be written as $M_i r M_j$

Relations

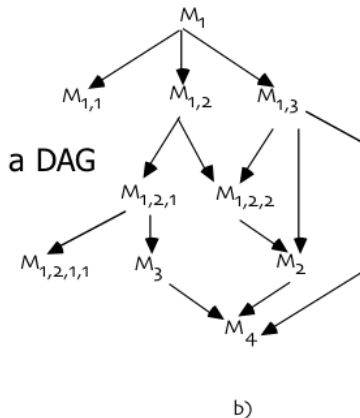
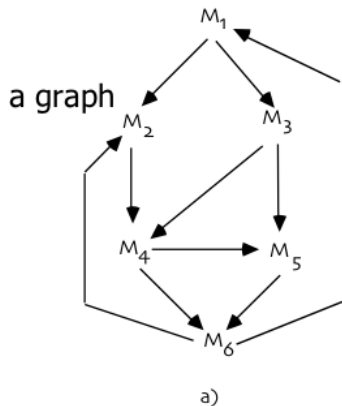
- Transitive closure r^+ of r

$M_i r^+ M_j$ iff $M_i r M_j$ or $\exists M_k$ in S such that $M_i r M_k$ and $M_k r^+ M_j$

- r is a hierarchy iff there are no two elements M_i, M_j such that $M_i r^+ M_j \wedge M_j r^+ M_i$

Relations Continued

- Relations can be represented as graphs
- A hierarchy is a DAG (directed acyclic graph)



Why do we prefer the uses relation to be a DAG?

Desirable Properties

- USES should be a hierarchy
 - ▶ Hierarchy makes software easier to understand
 - ▶ We can proceed from the leaf nodes (nodes that do not use other nodes) upwards
 - ▶ They make software easier to build
 - ▶ They make software easier to test
- Low coupling
- Fan-in is considered better than Fan-out: WHY?

DAG Versus Tree

Is a DAG a tree? Is a tree a DAG?

DAG Versus Tree

Would you prefer your uses relation is a tree?

Hierarchy

- Organizes the modular structure through **levels of abstraction**
- Each level defines an **abstract (virtual) machine** for the next level
- Level can be defined precisely
 - ▶ M_i has level 0 if no M_j exists such that $M_i r M_j$
 - ▶ Let k be the maximum level of all nodes M_j such that $M_i r M_j$, then M_i has level $k + 1$

Static Definition of Uses Relation

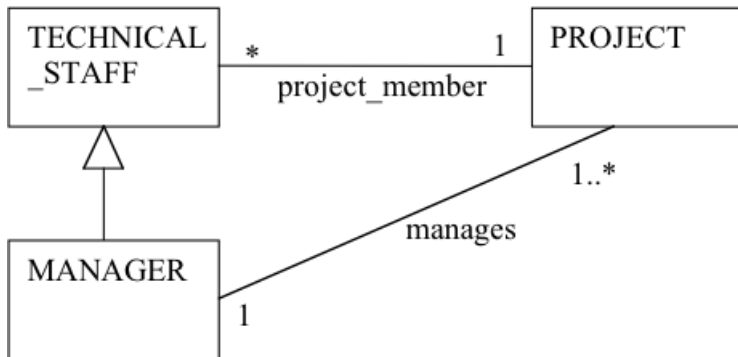
Your program has code like:

```
if cond then ServiceFromMod1 else ServiceFromMod2
```

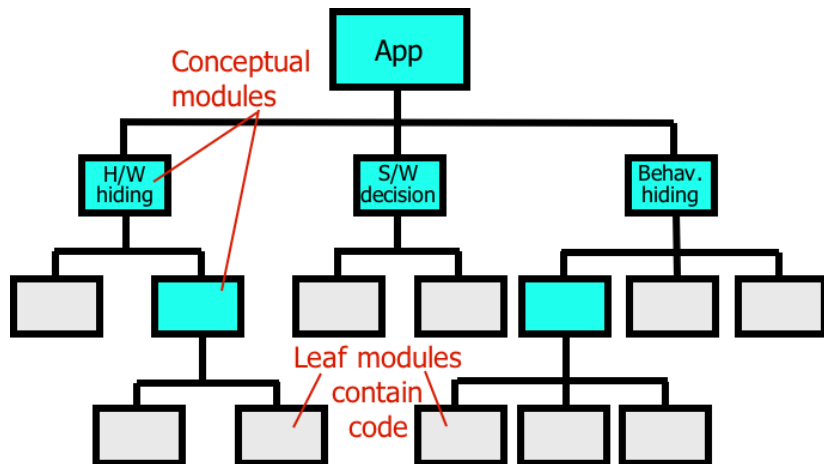
This is the only place where each module is used. Does this mean the uses relation depends on the dynamic execution of the program?

Question about Association and DAG

Is the uses relation here a DAG?



Module Decomposition (Parnas)



Module Decomposition (Parnas)

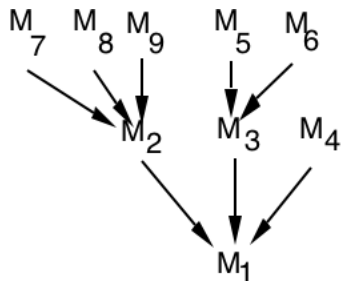
For the module decomposition on the previous slide:

- Does it show a Uses relation?
- Is it a DAG?
- Is it a tree?

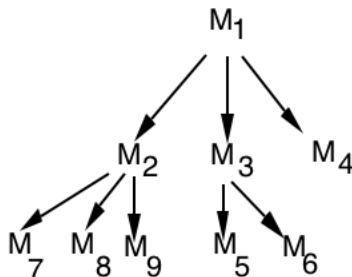
IS_COMPONENT_OF

- The Parnas decomposition by secrets gives an IS_COMPONENT_OF relationship
- Used to describe a higher level module as constituted by a number of lower level modules
- A IS_COMPONENT_OF B means B consists of several modules of which one is A
- B COMPRISES A
- $M_{S,i} = \{M_k | M_k \in S \wedge M_k \text{ IS_COMPONENT_OF } M_i\}$ we say that $M_{S,i}$ IMPLEMENTS M_i
- How is IS_COMPONENT_OF represented in UML?

A Graphical View



(IS_COMPONENT_OF)



(COMPRISES)

They are a hierarchy

Product Families

- Careful recording of (hierarchical) USES relation and IS_COMPONENT_OF supports design of program families
- Attempt to recognize modules that will differ in implementation between family members
- New program family member should start at the documentation of the design, not with the code

Remember - Information Hiding

- Basis for design (i.e. module decomposition)
- Implementation secrets are hidden to clients
- They can be changed freely if the change does not affect the interface
- Try to encapsulate changeable requirements and design decisions as implementation secrets within module implementations
- Decomposition by secrets, not by sequence of steps

Prototyping

- Once an interface is defined, implementation can be done
 - ▶ First quickly but inefficiently
 - ▶ Then progressively turned into the final version
- Initial version acts as a prototype that evolves into the final product

Module Guide

- Part of Parnas' Rational Design Process (RDP)
- When decomposing the system into modules, we need to document the module decomposition so that developers and other readers can understand and verify the decomposition
- Parnas proposed a Module Guide (MG) based on the decomposition module tree shown earlier

RDP - MG

- The MG consists of a table that documents each module's service and secret
- Conceptual modules will have broader responsibilities and secrets
- Following a particular branch, the secrets at lower levels "sum up" to the secret at higher levels
- The leaf modules that represent code will contain much more precise services and secrets
- Only the leaf modules are actually implemented
- The MG should list the likely and unlikely changes on which the design is based

Example

3. Module Hierarchy

| Level 1 | Level 2 | Level 3 | Level 4 |
|--------------------------|----------------------------------|--|---|
| Hardware-Hiding module | Input Device module | Mouse Motion module | |
| | | Keyboard Input module | |
| | Output Device module | File Reading module | |
| | | Screen Display module | |
| File Writing module | | | |
| Master Control module | | | |
| Behavior-Hiding module | Function Drivers module | Frame Display module | |
| | | User Command Detect module | |
| | | Geometry Specification module | Boundary Specification module |
| | | | Subdivision Specification module |
| | | Physical Attributes Specification module | Material Property Specification module |
| | | | Boundary Condition Specification module |
| | | Save_Load module | Save_Load Input File module |
| | | | Write Output File module |
| | Shared Services module | Frame Geometry module | |
| | | Error Handle module | |
| | | Mesh Drawing module | |
| | | Drawing Tools module | |
| Software Decision module | Input Data module | | |
| | Mesh Data module | Combinatorial Grid module | |
| | | Geometric Grid module | |
| | | Physical Attributes module | |
| | Generic Tools module | Grid Function Vector module | |
| | | Edge Iterator module | |
| | | Boundary Iterator module | |
| | | Cell Neighbor Search module | |
| | Mesh Generating Algorithm module | Geometric Mesh Generation module | |
| | | Physical Attributes Assignment module | |

Module Details

- For each module
- Module name
- Secret (informal description)
- Service or responsibility (informal description)
- For “leaf” modules add
 - ▶ Associated requirement
 - ▶ Anticipated change
 - ▶ Module prefix

RDP - MIS

- For each leaf module we need to document its interface and its implementation
- In RDP, the interfaces are documented in the Module Interface Specification (MIS)
- We have already seen MIS examples specified as Module State Machines

References

- Parnas, David L, Software Fundamentals: collected papers by David L. Parnas, edited by Daniel M. Hoffmann and David M. Weiss, Lucent Technologies and Daniel M. Hoffmann, 2001, ISBN 0-201-70369-6
- Parnas, D. L., “On a ‘Buzzword’: Hierarchical Structure”, IFIP Congress 74, North Holland Publishing Company, 1974, pp. 336–339
- Parnas, D. L., “On the Criteria to be Used in Decomposing Systems into Modules”, Communications of the ACM, 15, 12, December 1972, pp. 1053–1058.

References Continued

- Parnas, D. L., “Designing Software for Ease of Extension and Contraction”, Copyright 1979, IEEE Transaction on Software Engineering, March 1979, pp. 128–138,
- Parnas, D. L., Clements, P. C., Weiss, D. M., “The Modular Structure of Complex Systems”, IEEE Transaction on Software Engineering, March 1985, Vol SE-11, No. 3, pp. 259-266 (special issue on the 7th International Conference on Software Engineering)

References Continued

- Parnas, D. L., Clements, P. C., “A Rational Design Process: How and Why to Fake it”, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp. 251-257.
- Parnas, On the design and development of program families, IEEE Transactions on Software Engineering, SE-2(1), March 1976.
- Hoffmann, Daniel, M., and Paul A. Strooper, Software Design, Automated Testing, and Maintenance A Practical Approach, International Thomson Computer Press, 1995, <http://citeseer.ist.psu.edu/428727.html>

References Continued

- Dahl, Dijkstra and Hoare, Structured Programming, Academic Press, 1972 (modular decomposition)
- ElSheikh, Ahmed, W. Spencer Smith, and Samir E. Chidiac. (2004) Semi-formal design of reliable mesh generation systems. Advances in Engineering Software, Vol 35, Issue 12, pp 827-841.
- Carlo Ghezzi, Mehdi Jazayeri and Dino Mandrioli, Fundamentals of Software Engineering, 2nd Ed., Prentice Hall, 2003

References Continued

- Dijkstra, The structure of THE multiprogramming system. Communications of the ACM, 11(5): 341-346, May 1968.
- Linger, Mills and Witt. Structured Programming: Theory and Practice, Addison-Wesley, 1979 (step-wise refinement)
- Wirth, Program development by stepwise refinement, Communications of the ACM, 14(4):221-227, April 1971.