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

Winter 2018

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

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 6, 2018

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

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

Administrative Details

- Assignment 2 (Steps and questions need clarification, spec essentially done)
  - Part 1: February 12, 2018
  - Partner Files: February 18, 2018
  - Part 2: March 2, 2018
  - The assignment uses PyTest, covered in this week's tutorials
- Midterm exam
  - Wednesday, February 28, 7:00 pm
  - 90 minute duration
  - Multiple choice - 30–40 questions

Assignment 2

A2

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

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

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

Relations

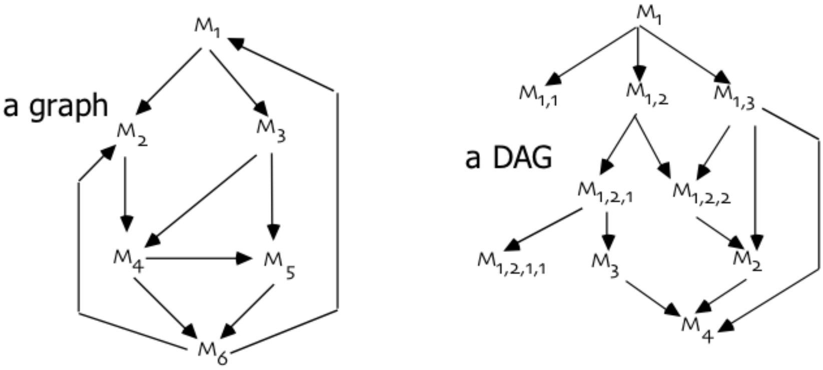
- Transitive closure r^+ of r

  $M_i r^+ M_j$ 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 $M_i r^+ M_j \land M_j r^+ M_i$

Relations Continued

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



a) b)

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

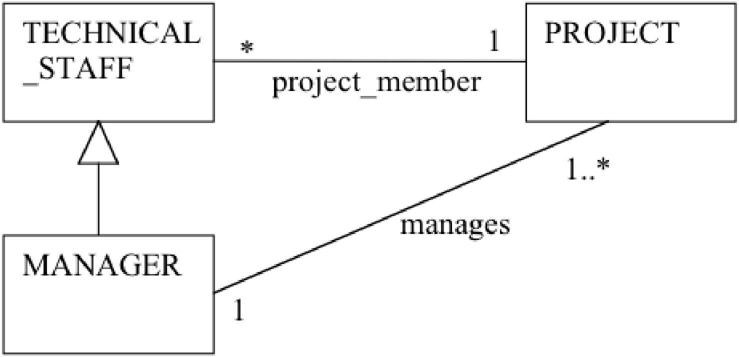
Your program has code like:

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

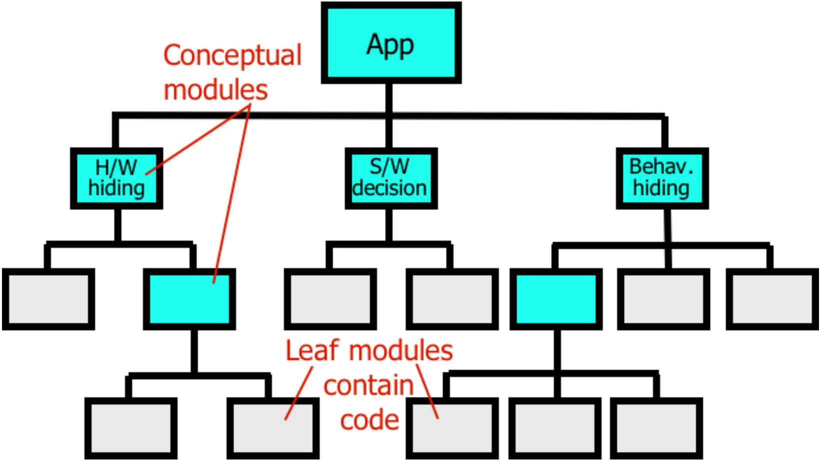
This is the only place where each module is used. Does this mean the uses relation depends on the dynamic execution of the program?

Question about Association and DAG

Is the uses relation here a DAG?



Module Decomposition (Parnas)



Conceptual modules

Leaf modules contain code

Module Decomposition (Parnas)

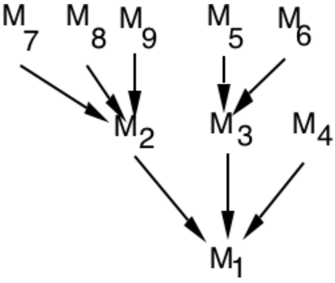
For the module decomposition on the previous slide:

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

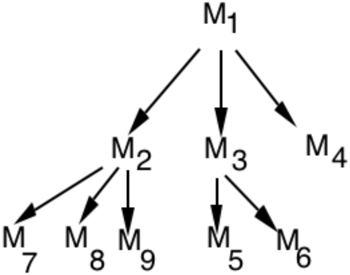
IS_COMPONENT_OF

- 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