Clustering Software Systems to Identify Subsystem Structures

Understanding the Structure of Programs is Difficult

- Developers create sophisticated applications that are complex and involve a large number of interconnected components.
- **Result:** Program understanding is difficult
- **Goal:** Use automated techniques to help developers understand the structure of software systems.

Common Problems

- Creating a good mental model of the structure of a complex system.
- Keeping a mental model consistent with changes that occur as the system evolves.
- These problems are exacerbated by:
  - non-existent or inconsistent design documentation
  - high rate of turnover among IT professionals
- **Assumption:** Understanding the structure of a systems software is valuable for maintainers.

Solutions

- **Automatic:** Use software clustering techniques to decompose the structure of software systems into meaningful subsystems.
  - Subsystems help developers navigate through the numerous software components and their interconnections.
- **Manual:** Use notations such as UML to specify the software structure.

A Software Clustering Primer

- Directed graphs are commonly used to represent the structure of software.
- Assume that this graph consists of a finite set of **components** (nodes):
  - classes, modules, files, packages, etc.
- and **relationships** (edges) between components:
  - inherit, import, include, call, instantiate, etc.
- <u>**Problem:**</u> How do we partition the nodes of the graph into clusters (subsystems)?

Software Clustering Challenges

- There are many ways to partition a graph into clusters.
- How do we create efficient algorithms to find partitions of the graph that are representative of a system’s structure?
- How do we distinguish between **“good”** partitions, and **“bad”** partitions?

How Hard is this Problem?

If every partition of the graph is considered, the number of partitions that will need to be investigated is:

$$S_{n,k} = \begin{cases} 1 & \text{if } k = 1 \text{ or } k = n \\ S_{n-1,k-1} + S_{n-1,k} & \text{otherwise} \end{cases}$$

The above recurrence equation grows exponentially with respect to the number of nodes (n) in the graph
(each partition 1≤k≤n clusters).

$S_{n,k}$ for some values of *n*:

1=1; 5=52; 10=115,975; 15=1,382,958,545;
20=51,724,158,235,372

Some Solutions

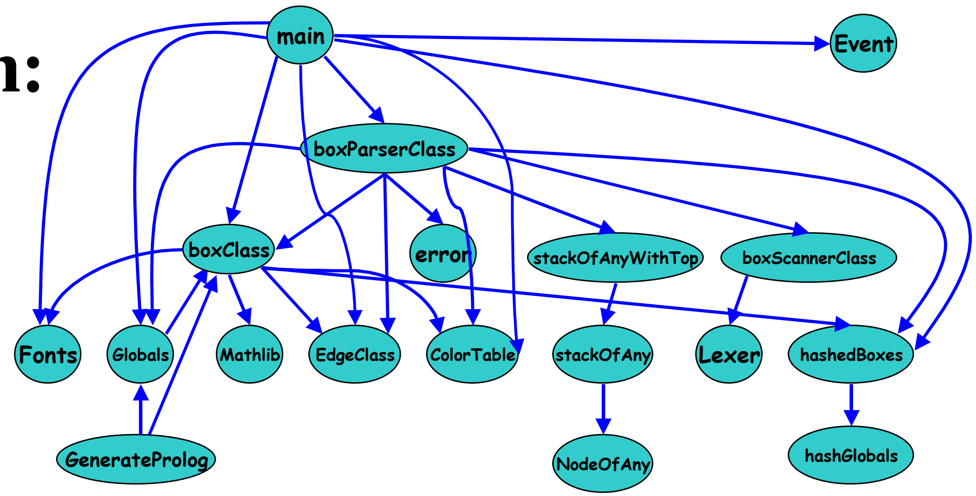
- Enumerating every possible partition of the software structure graph is not practical.
- Heuristics can be used to reduce the number of partitions:
  - Searching algorithms
  - Knowledge about the source code
    - Names (files, directories, method/procedure)
    - Designer input, design documentation
  - Remove entities that provide little structural value
    - libraries
- Result are sub-optimal, but are often adequate.

Why is Software Clustering Useful?

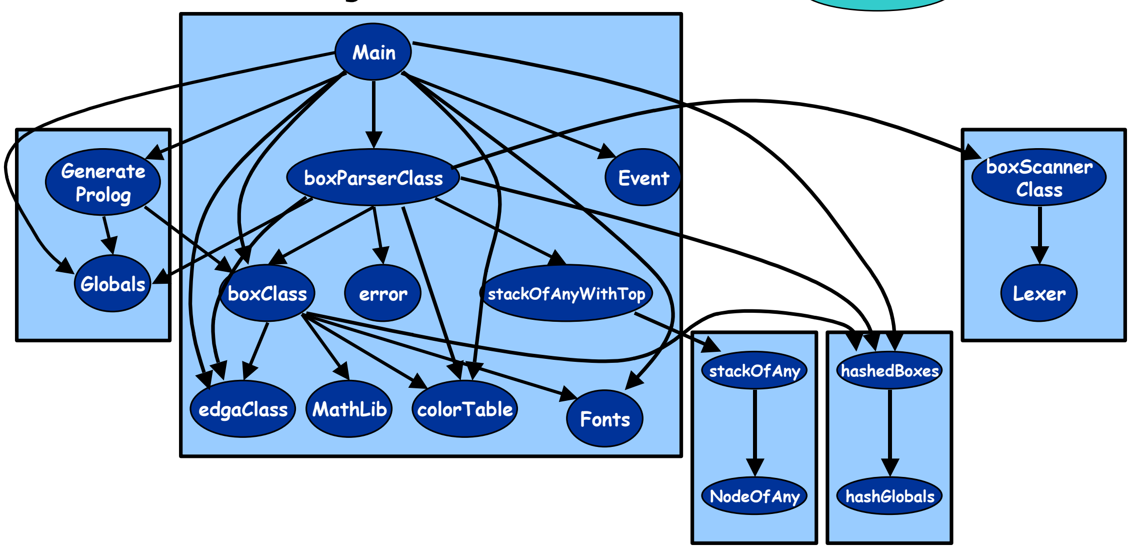
- Helps new developers create a mental model of the software structure.
- Especially useful in the absence of experts or accurate design documentation.
- Helps developers understand the structure of legacy software.
- Enables developers to compare the documented structure with the automatically created (actual) structure.

Example: Clustering Simplifies Program Structure Understanding

Original System:



Clustered System:



Modern Relevance of Software Clustering

- Clustering has been studied for many years in the fields of mathematics, science and engineering.
- Clustering research in software engineering increased because of Y2K and the ‘webifying’ of legacy systems.
- New clustering approaches have been developed, and classical clustering techniques have been modified to work with software structures.

Creating Clusters at Design Time

- Parnas (1972) – Information Hiding
  - Hide program “secrets” behind interfaces
  - A manual form of clustering
- Object Oriented Design (Booch, 1994)
  - Objects group (cluster) related data and operations that act upon the data.
  - Booch suggests principles that are commonly used in clustering research:
    - Abstraction
    - Encapsulation
    - Hierarchies & Modularity

Classification of Software Clustering Research

- **Clustering Procedures/Functions into Modules**
  - Hutchens & Basili, Schwanke, Lindig & Snelting, Montes de Oca & Carver
- **Clustering Modules/Classes into Subsystems**
  - Müller et. al., Mancoridis, Mitchell et. al., Anqetil, Fourrier & Lethbridge, Choi & Scacchi
- **Measuring Differences between Clustered Systems, Incremental Maintenance & Metrics/Measurements**
  - Murphy, Tzerpos & Holt, Mitchell & Mancoridis, Anquetil, Fourrier & Lethbridge

Clustering Techniques

- There are many different clustering techniques, but clustering techniques in general must consider (Wiggerts, 1997):
  - **Representation:** The entities and relationships to be clustered
  - **Similarity:** The degree of similarity between the software entities
  - **Algorithms:** Algorithms that use the similarity measurement to make clustering decisions

Representation

- There are many choices based on the desired granularity of recovered system design
  - Entities may be variables/procedures or modules/classes.
  - What types of relationships will be considered?
  - Will the relationships be weighted?

Representation Examples

- MDG (Bunch – Mancoridis, Mitchell, et. al.)
  - Directed graph, edges are weighted based on the number of dependencies between the nodes
- Resource Flow Graph – RFG (Choi and Scacchi)
  - Directed graph, edges represent resources provided to a node from another node
- Resource Flow Graph – RFG (Müller, et. al.)
  - Directed graph, edges are labeled with the actual set of resource names that are exchanged between the nodes (modules)
- Hutchens & Basili
  - Dissimilarity matrix formed from data bindings.

Similarity

- Similarity measurements are used to determine the degree of “similarity” between a pair of entities
- Different types:
  - **Association coefficients:** Based on common features that exist (or do not exist) between a pair of entities
    - Most common type of similarity measurement
  - **Distance measures:** Measure of the degree of dissimilarity between entities.

Example Similarity Measurement

Classical similarity measurements:

		Entity j	
		1	**0**
Entity i	**1**	***a***	***b***
	0	***c***	***d***

a: Number of common features in ***entity i*** and ***entity j***
b: Number of features unique to ***entity j***
c: Number of features unique to ***entity i***
d: Number of features absent in both ***entity i*** and ***entity j***

$$simple(i, j) = \frac{a+d}{a+b+c+d} \quad \text{and} \quad Jaccard\,(i, j) = \frac{a}{a+b+c}$$

Antquetil et. al. (1999) compared the Simple and Jaccard algorithms and found that overall the Jacacard algorithm produced better results.

Hutchens & Basili (1985)
Data Bindings

A data binding classifies the similarity between two procedures based on the common variables that are within the static scope of the two procedures.

- Useful for clustering procedures and variables into modules.
- Uses hierarchical clustering algorithms to form clusters from the data bindings.
- Addressed several aspects of clustering
  - Use of hierarchies, stability (also examined by Tzerpos and Holt), consistency between a clustered view and a designers view (Anquetil et. al.).

Schwanke (1991)
Machine Learning

- Arch is a semi-automatic clustering technique that is based on using machine learning to maximize cohesion and minimize coupling between software components.
- Maverick analysis is a unique feature of Arch where misplaced procedures are relocated to more appropriate modules.
  - Maverick procedures share many features with procedures in other modules.

Schwanke (1991) Arch Algorithm

Place each entity into a subsystem by itself

Repeat

    Identify the two most similar entities

    Combine them into a common subsystem

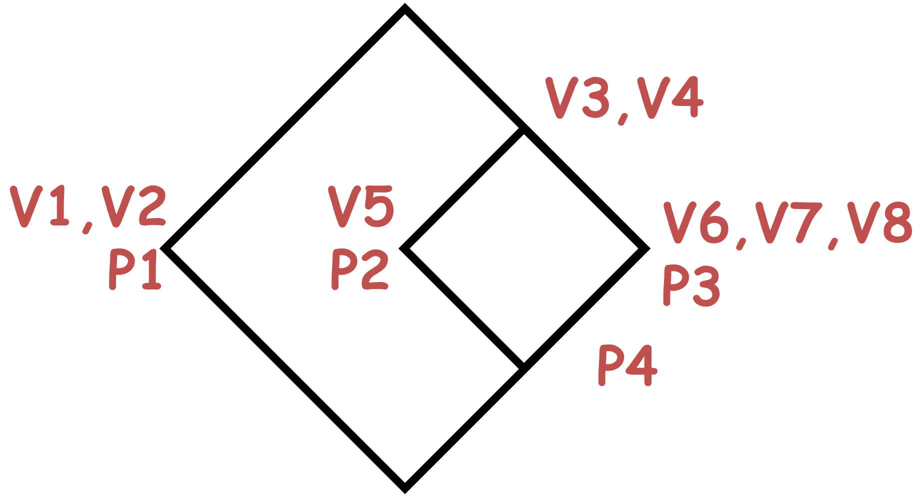
Until the results are "satisfactory"

Lindig & Snelting (1997) Mathematical Concept Analysis

- Used for clustering procedures and variables into modules.
- A concept is defined as *C=(P,V)*
  - Given a set of variables, *V*, *P = cp(V)* is a set of common procedures
  - Given a set of procedures, *P*, *V=cv(P)* is a set of common variables
- A context can be represented as a lattice.
- Lattice can be transformed into a "tree-like" structure to form the modules.

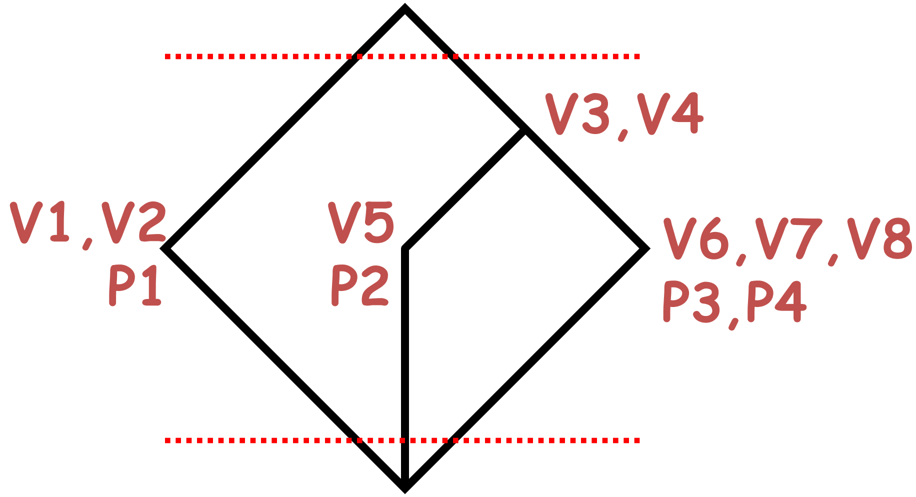
Lindig & Snelting (1997)
Mathematical Concept Analysis

	V1	V2	V3	V4	V5	V6	V7	V8
P1	X	X						
P2			X	X	X			
P3			X	X		X	X	X
P4			X	X	X	X	X	X



Have P2 Pass V5 to P4

	V1	V2	V3	V4	V5	V6	V7	V8
P1	X	X						
P2			X	X	X			
P3			X	X		X	X	X
P4			X	X		X	X	X



Müller et. al. (1992)
The Rigi Tool

- Building block of cluster is a subsystem not a module.
- Rigi – a semiautomatic clustering tool
  - Clustering based on heuristics such as measuring the relative strength between subsystems
    - Interconnection Strength (IS) measurement
- Other interesting research aspects:
  - Omnipresent modules
  - Use of module and directory names to make clustering decisions (further researched by Anquetil et. al.)

Müller et. al. (1992)
Rigi Algorithm

For each pair of entities
measure the Interconnection Strength (IS).

If the IS value exceeds a user-defined threshold
then
place the entities into a common subsystem

Choi & Scacchi (1990) Automatic Clustering

- Goal is to automatically restructure (cluster) legacy systems.
- Build resource flow graph (RFG)
  - Nodes are modules.
  - An edge is placed from node *A* to node *B* if module *A* provides one or more resources to module *B.*
- Clustering approach is based on partitioning the RFG by finding articulation points in the graph.

Montes de Oca & Carver (1994) Data Mining Clustering

- Apply data mining techniques that have been developed for databases to software clustering
- Data mining can find non-trivial relationships between elements in a database.
  - Software Clustering can find non-obvious relationships between source code components.
- Data mining can find interesting relationships in databases without upfront knowledge of the objects being studied
  - Developers who want to cluster are typically not familiar with the structure of the system.

Montes de Oca & Carver (1994) Data Mining Clustering

- Data mining techniques are designed to work with a large amount of information efficiently
  - Most clustering tools are very slow because of the complexity of the software clustering problem.

Mancoridis, Mitchell et. al. (1998) Optimization-based Clustering

"Treat automatic clustering as an optimization problem"

- Automatic clustering technique is implemented as a Java tool called Bunch.
- Bunch is fully automatic, but can exploit designer knowledge when it is available.
- Partitions a Module Dependency Graph into a subsystem hierarchy.
- Like Arch, Bunch attempts to maximize cohesion and minimize coupling.

Mancoridis, Mitchell et. al. (1998) Bunch Algorithm

Create the MDG from the source code structure and generate a random set of partitions of the MDG (the *population*)

For each *p* in the population, Repeat:

  Let partition p' = p

  *L*et *q* be a partition found by applying one of our clustering algorithms to p

  if MQ(q) > MQ(p), let p = q

Until MQ(p') = MQ(q)

Return p

Anquetil, Lethbridge, et al (1999) Comparing Clustering Algorithms

- Anquetil, Fourrier & Lethbridge’s compare various hierarchical clustering algorithms
- Work investigated classical clustering algorithms and similarity measurements.
  - Simple versus Jaccard
- This research defined 3 metrics that can be used to compare different clustering approaches.

Anquetil, Lethbridge, et al (1999) Metrics

- **Precision** – agreement between the clustering method and the expert.
- **Recall** – agreement between the expert and the clustering method.
- **Goal:** High precision and recall, but their experimental results indicate that the classical clustering methods tend to have good precision, but poor recall.

Tzerpos & Holt (1999) Distance Between Partitions

- Mojo is a distance metric that measures the “similarity” between two different partitions of the same system:
  - Good for comparing results between different clustering techniques.
  - Good for validating results with an expert.
  - Good for stability analysis (structural drift over time).

Tzerpos & Holt (1999) Mojo Metric

mno(A,B) = The number of move and join operations to transform A into B

MoJo(A,B) = min(mno(A,B),mno(B,A))

- Given 2 partitions of the same system the goal is to measure the effort to transform the first partition into the other. Based on move and join operations
  - **Move:** move a resource from one cluster to another
  - **Join:** merge two clusters into a single cluster

Anquetil & Lethbrige (1999) Using Names of Source Files

- Anquetil and Lethbridge did research on using the names of source files to determine similarity.
- Technique includes dictionary lookup and substring analysis.
- Using file names produced good results for the systems that were studied.

Mitchell & Mancoridis (2001)

- Developed improved metrics to measure the similarity of two partitions:
  - A distance metric called MeCl
  - A similarity metric called EdgeSim
  - A framework for comparing clustering algorithms called CRAFT.
- More details will follow …