

Software Clustering

Decomposing a large software system into meaningful subsystems

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

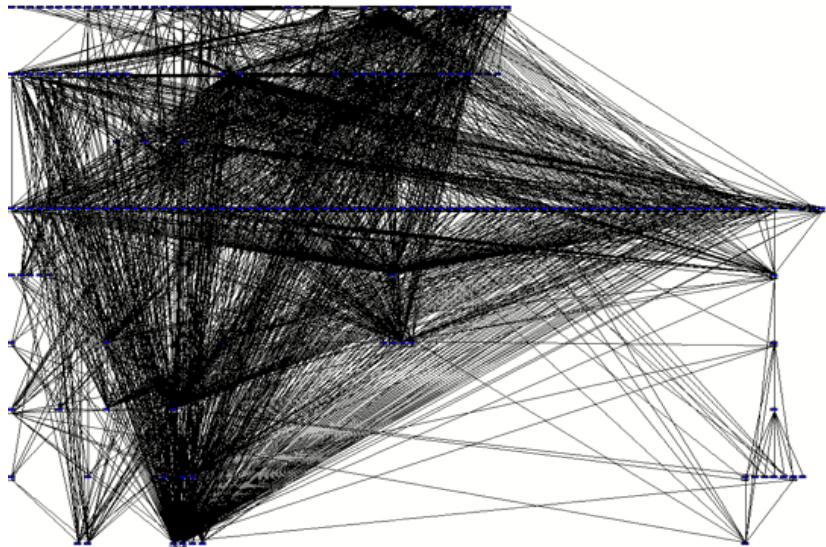
- Creating a good mental model of the structure of a complex system.
- Keeping a mental model consistent with changes that occur as the system evolves.
- These problems are exacerbated by:
 - Non-existent or inconsistent design documentation
 - High rate of turnover among IT professionals
- **Assumption:**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

- **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.

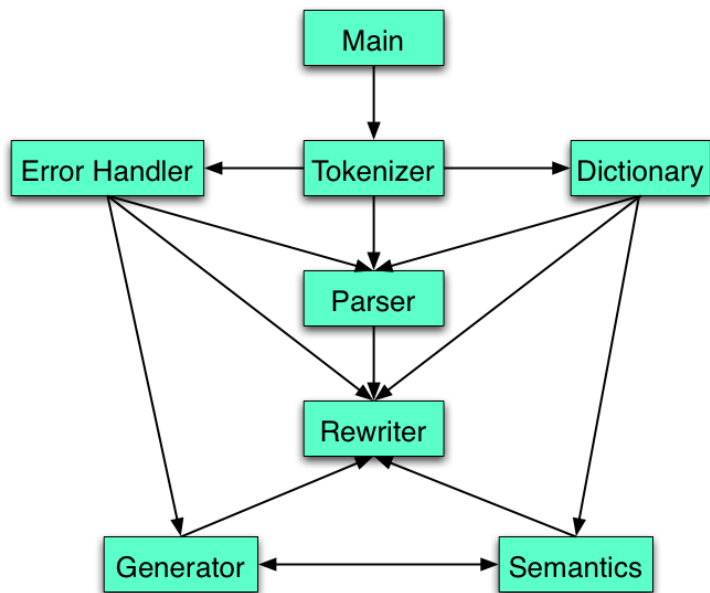
Why is clustering useful?

- 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 (before)



Example (after)



Software Clustering Challenges

- There are many ways to partition a set of entities into clusters.
- How do we create efficient algorithms to find partitions that are representative of a system's structure?
- How do we distinguish between **good** and **bad** partitions?

How Hard is this Problem?

- The number of partitions of n objects into k clusters is:

$$S_{n,k} = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$$

- The number of ways to partition a set of n objects is: $B_n = \sum_{k=1}^n S_{n,k}$
- This function grows exponentially with respect to n . Some values:

1	5	10	15	20
1	52	115,975	1,382,958,545	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, directory structure, designer input
 - Remove entities that provide little structural value
 - Libraries, omnipresent nodes
- Result is sub-optimal, but often adequate.

- Clustering Procedures/Functions into Modules
- Clustering Modules/Classes into Subsystems
- Evaluating clustering algorithms
 - Measuring distance between partitions
 - Algorithm stability

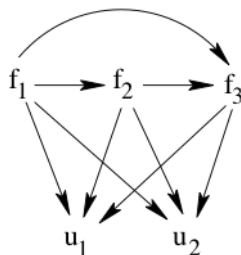
- There are many different clustering techniques, but they all need to consider:
 - **Representation:** The entities and relationships to be clustered
 - **Similarity:** What determines the degree of similarity between the software entities
 - **Algorithms:** Algorithms that use the similarity measurement to make clustering decisions

- 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?

- 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.

Similarity Measurements

- Assume that every entity is expressed in terms of binary features, 1 denoting the existence of a feature, 0 its absence.



	f₁	f₂	f₃	u₁	u₂
<i>f₁</i>	0	1	1	1	1
<i>f₂</i>	1	0	1	1	1
<i>f₃</i>	1	1	0	1	1
<i>u₁</i>	1	1	1	0	0
<i>u₂</i>	1	1	1	0	0

Similarity Measurements

- We can also include information about who developed what file, and where each file is located

	f₁	f₂	f₃	u₁	u₂	Alice	Bob	p₁	p₂	p₃
<i>f₁</i>	0	1	1	1	1	1	0	0	1	0
<i>f₂</i>	1	0	1	1	1	0	1	1	0	0
<i>f₃</i>	1	1	0	1	1	0	1	0	1	0
<i>u₁</i>	1	1	1	0	0	1	0	0	0	1
<i>u₂</i>	1	1	1	0	0	0	1	0	0	1

For two entities i and j , we can define...

- **a**: Number of features present in both entities
- **b**: Number of features unique to entity i
- **c**: Number of features unique to entity j
- **d**: Number of features absent in both entities

Association Coefficients

- Association co-efficients can be defined based on these values:

Simple Matching coefficient $\frac{a+d}{a+b+c+d}$

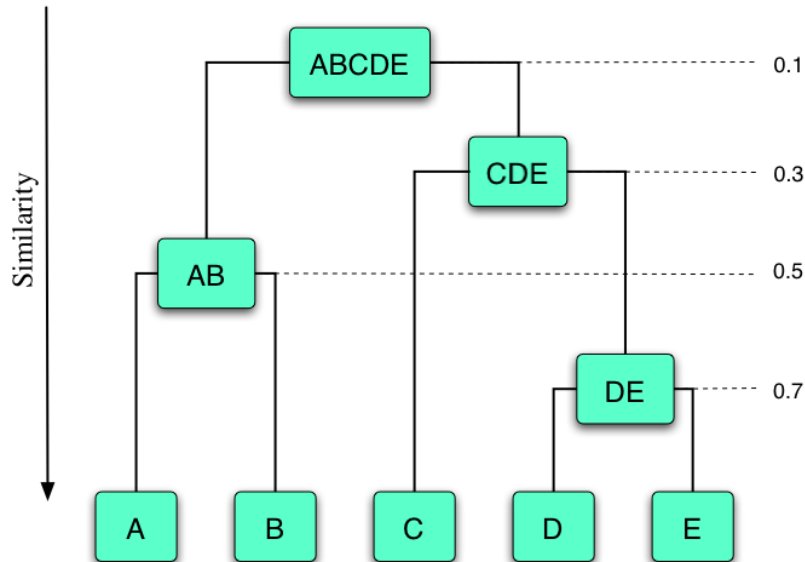
Jaccard coefficient $\frac{a}{a+b+c}$

Sorensen coefficient $\frac{2a}{2a+b+c}$

Agglomerative hierarchical algorithm

- Start by creating one cluster for each object
- Join the two most similar objects into one cluster
- Continue joining the two most similar objects/clusters until everything is in one cluster
- What you get is a dendrogram...

Dendrogram example



Cut height

- By choosing to “cut” the dendrogram at a particular height, we can create a partition of the set of objects, e.g. a cut height of 0.45 in the previous example would give us 3 clusters
- Finding an appropriate cut height is a tough problem
- Heuristics, such as the number of clusters, are usually employed

- How to determine the similarity between two already formed clusters (or an object and a cluster)
- Many possibilities
 - Minimum of all pair-wise similarities
 - Maximum of all pair-wise similarities
 - Weighted or unweighted averages

Assignment tool: aa

- The aa tool allows to run any version of the agglomerative algorithms described before
- It requires input in “market basket data” form. You can transform from RSF to MBD with:

```
unitrans input.rsf output.mbd
```

input.rsf

call	f1	f2
call	f1	f3
call	f2	f3
call	f1	u1
call	f1	u2
call	f2	u1
call	f2	u2
call	f3	u1
call	f3	u2

output.mbd

f1	f2	f3	u1	u2
f2	f1	f3	u1	u2
f3	f1	f2	u1	u2
u1	f1	f2	f3	
u2	f1	f2	f3	

Assignment tool: aa

- **Example:** aa input.mbd contain.rsf
-c0.4 -s1 -a2
 - Cluster the objects in `input.mbd` using a cut-height of 0.4, the Simple Matching Coefficient, and the Weighted Average Algorithm
- **Output:**

```
contain ss5 u1
contain ss5 u2
contain ss3 f3
contain ss3 f1
contain ss3 f2
```

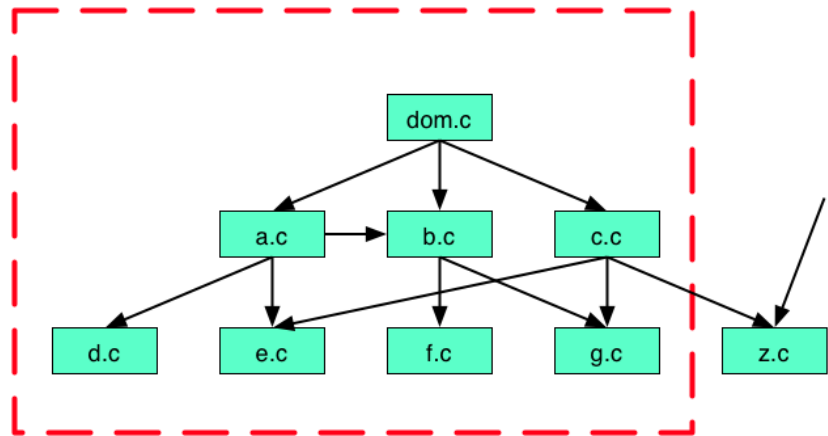
Pattern-based software clustering

- Manual decompositions of large pieces of software often contain certain types of subsystems
- A software clustering algorithm that creates clusters based on these patterns would have a better chance of creating a decomposition that can help system comprehension
- These clusters can also have better names (based on the pattern they were derived from) as well as a more manageable number of contents

The ACDC algorithm

- A skeleton of the decomposition is created based on the identified patterns
- Entities not clustered this way are assigned to the cluster that they exhibit the largest connectivity to
- Experiments with large systems have shown that the skeleton usually contains at least half the system entities

Example pattern: Subgraph Dominator



- The acdc tool is an implementation of this algorithm

- **Example:**

```
acdc input.rsfc output.rsfc -125
```

- Cluster the objects in `input.rsfc` with a maximum size of 25 for the Subgraph Dominator pattern

Optimization-based Clustering

- If one can express the desired properties of a clustering as a formula, then the problem of clustering is reduced to that of finding the decomposition that optimizes the value of the formula
- A typical goal is to maximize cohesion and minimize coupling

- Bunch attempts to maximize the value of the MQ function

$$MQ = \begin{cases} \frac{\sum_{i=1}^k A_i}{k} - \frac{\sum_{i,j=1}^k E_{i,j}}{\frac{k(k-1)}{2}} & k > 1 \\ A_1 & k = 1 \end{cases}$$

where $A_i = \frac{\mu_i}{N_i^2}$ and $E_{i,j} = \begin{cases} 0 & i = j \\ \frac{\epsilon_{i,j}}{2N_i N_j} & i \neq j \end{cases}$

N_i : the number of entities in cluster i

μ_i : the number of intra-edges in cluster i

$\epsilon_{i,j}$: the number of inter-edges between clusters i and j

- Finding the optimal clustering based on this formula is impractical
 - Exhaustive search is not recommended for more than 15 entities
- Bunch employs hill climbing and genetic algorithms to find approximate solutions

Assignment tool: bunch

- Bunch is an interactive tool written in Java
- Input is in a format that is exactly like RSF except that the first token is missing, i.e. only one type of relationship is assumed
- Output is in a format called SIL that can be translated to RSF (see webpage)

- The literature contains many more ideas for clustering algorithms
- Data mining techniques as well as mathematical tools such as concept analysis have been used for clustering purposes
- Using naming or ownership information has also been shown to improve clustering results