Chapter 8

Design Characteristics and Metrics

Software Engineering

“Good” design

What makes a design “good”?

- Easy to understand
- Easy to change
- Easy to reuse
- Easy to test
- Easy to integrate
- East to code

Metrics

Software measurement is a means to an end, not an end in itself.

Goal of measurement:

- What is causing poor quality?
- Where are we spending all our time on software development?
- How accurate are our estimates?
- What is the most cost-effective way to improve our quality?

Halstead's Complexity Metric

Halstead Complexity Metric (early 1970s)

n1 = number of distinct operators
n2 = number of distinct operands
N1 = sum of all occurrences of n1
N2 = sum of all occurrences of n2

Program vocabulary: $n = n1+n2$
Program length: $N = N1+N2$

Volume: $V=N*(\log_2 n)$

Halstead's Complexity Metric

Potential volume

$$V^{@} = (2 + n2^{@}) \log_2(2+n2^{@})$$

Program implementation level

$$L=V^{@}/V$$

Effort

$$E=V/L$$

Only takes into account the lexical complexity, not the structure or the logic.

Halstead's Complexity Metric

Potential volume

$$V^{@} = (2 + n2^{@}) \log_2(2+n2^{@})$$

Program implementation level

$$L=V^{@}/V$$

Effort

$$E=V/L$$

Only takes into account the lexical complexity, not the structure or the logic.

Example models\user.go

```go
// AddUser adds a user to the database with the provided attributes.
func (UserRepo) AddUser(uName string, usernameCAS string, activeFlag bool) error {
    user := User{UName: uName, UsernameCAS: usernameCAS, ActiveFlag: activeFlag}
    if err := db.FirstOrCreate(&user).Error; err != nil {
        return fmt.Errorf("failed to create user: %v", err)
    }
    return nil
}
```

McCabe's Cyclomatic Complexity

Cyclomatic complexity = E – N + 2p

E – number of edges of the graph
N – number of nodes of the graph, and
p – number of connected components
(usually p=1)

Equivalently:
Cyclomatic complexity = number of binary decisions + 1
Cyclomatic complexity = number of closed regions + 1

McCabe's Cyclomatic Complexity

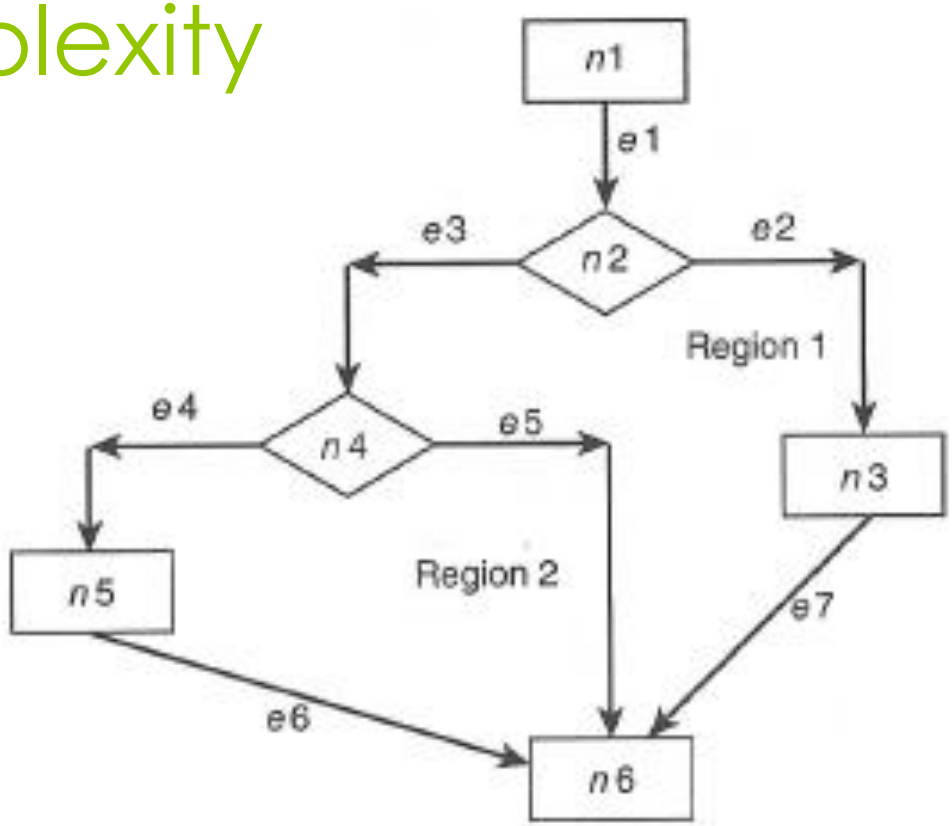


Figure 8.1 A simple flow diagram for cyclomatic complexity.

Henry-Kafura Information Flow

Fan-in – "Flow in", number of modules that call this module

Fan-out – "Flow out", number of modules called by this module

Complexity of a module p:

$$C_p = (\text{fan-in} * \text{fan-out})^2$$

Total complexity is the sum of the complexity of each module

Card & Glass

Improvement:

Structural complexity: $s_x = (\text{fan-out}_x)^2$

Data complexity: $D_x = P_x / (\text{fan-out}_x + 1)$

P_x - number of parameters passed

System complexity: is the sum of the above

Cohesion and Coupling

Cohesion - degree to which the elements within that unit belong or are related together (intramodule)

Coupling - degree of interaction and interdependence between two software units (intermodule)

Variety of Levels

Cohesion and coupling apply to many levels:

- Client, Server, Database system breakdown
- Model, View, Controller design pattern
- backend, frontend, docs , db_docs
- Within backend :
  endpoints, models, server
- Within models/semesters.go
  SemesterRepository interface
  structs
  functions

Cohesion

Categories of cohesion (low to high):

1. Coincidental – multiple unrelated tasks
2. Logical – series of similar tasks which are weakly related, for example an I/O unit design to perform all the different reads and writes
3. Temporal – elements that are related by time
4. Procedural – procedurally related items related in terms of some control sequence
5. Communicational – like procedural cohesion but the activities are targeted on the same data or the same sets of data
6. Sequential – one main activity , achieves one goal, the "single" activity is not as clear as the functional cohesion level
7. Functional – one main activity

Levels are not always clear cut.

Good source of descriptions and examples:

https://weeklyitblogs.wordpress.com/tag/example-of-cohesion/

Cohesion

Another measure of cohesion:

Data token – any instance of a variable or constant

Slice – statements within a program or procedure that can affect the value of a variable of interest

Data slice – data tokens in a slice that will affect the value of a specific variable of interest

Glue tokens – data tokens in a program or procedure that lie in more than one data slice

Superglue tokens – data tokens in a program or procedure that lie in every data slice in the program

Weak functional cohesion

Number of glue tokens / total number of data tokens

Strong functional cohesion

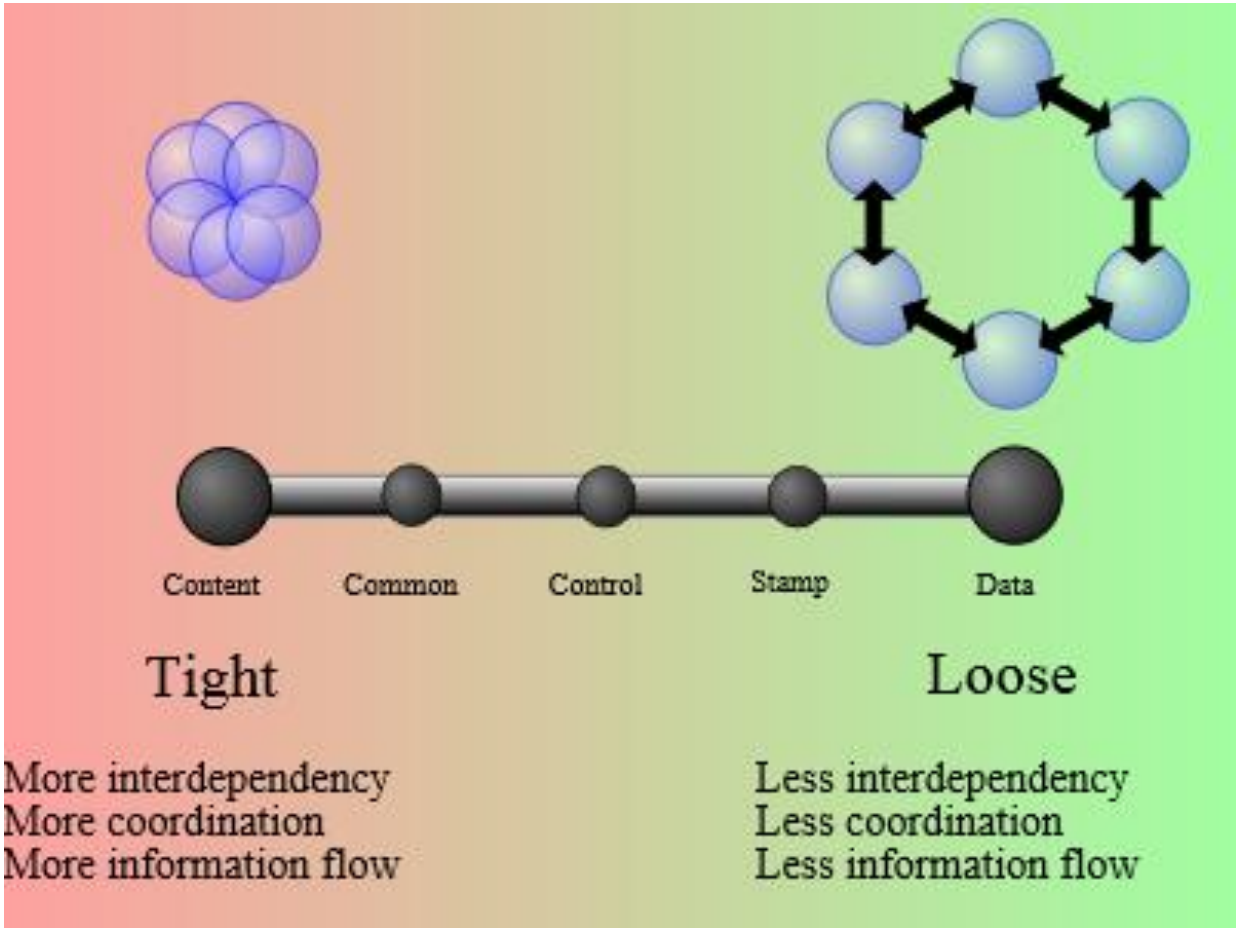
Number of superglue tokens / total number of data tokens

Coupling

Coupling (tight to loose):

1. Content coupling – units access each other's internal data or procedural information
2. Common coupling – units refer to the same global variables
3. Control coupling – one unit passes control information and explicitly influences the login of the other unit
4. Stamp coupling – one unit passes a group of data to another unit
5. Data coupling – like stamp coupling except only the needed data is passed.

Coupling


Content
Common
Control
Stamp
Data
Tight
Loose
More interdependency
More coordination
More information flow
Less interdependency
Less coordination
Less information flow

Object-Orientation

Possible metrics :

- Weighted Methods Per Class (WMC) – can weight these via lines of code, number of method calls, cyclomatic complexity
- Depth of Inheritance Tree (DIT)
- Number of Children (NOC)
- Coupling between Object Classes (CBO) – Coupling between classes
- Response for a Class (RFC) – Number of methods in a class or called by a class
- Lack of Cohesion in Methods (LCOM) – number of pairs of methods that don't share instance variable minus the number of pairs of methods that share instance variable

Law of Demeter

An object should send messages to the following kinds of objects:

- The object itself
- The object's attributes (instance variables)
- The parameters of methods in this object
- Any object created by the methods of this object
- Any object returned from a call to one of this object's methods
- Any object in any collection that is one of the preceding categories

Each unit should have only limited knowledge about other units. Only send messages to units "closely" related to the current unit.

User-Interface Design

Three “golden rules” of user-interface design:

- Place user in control
- Reduce the user’s memory load
- Design consistent user interface

User-Interface Design

Rules of User-Interface Design:

1. Strive for consistency (third “golden rule”)
2. Enable frequent users to use short cuts
3. Offer information feedback
4. Design dialogues to yield closure
5. Offer error prevention and simple error handling
6. Permit easy reversal of actions
7. Support internal focus of control (first “golden rule”)
8. Reduce short-term memory (second “golden rule”)

Usability Evaluation and Testing

Don’t wait until the end