

YEGOR BUGAYENKO

Lecture #6 out of 24 80 minutes

The slidedeck was presented by the author in this YouTube Video

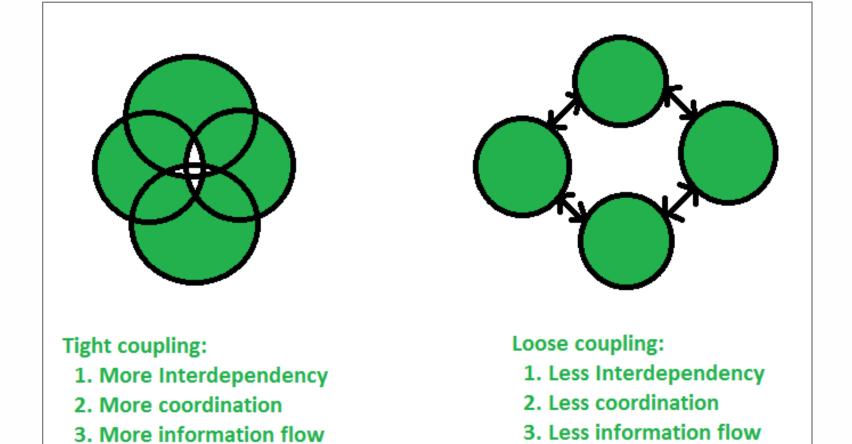
All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.



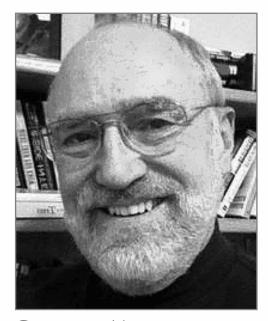
Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine

"The fewer and simpler the connections between modules, the easier it is to understand each module without reference to other modules."

— Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974. doi:10.1147/sj.132.0115



Source: https://www.geeksforgeeks.org/coupling-in-java/



GLENFORD MYERS

"Coupling is the measure of the strength of association established by a connection from one module to another. Strong coupling complicates a system since a module is harder to understand, change, or correct by itself if it is highly interrelated with other modules. Complexity can be reduced by designing systems with the weakest possible coupling between modules."

— Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974. doi:10.1147/sj.132.0115



Source: https://www.javatpoint.com/software-engineering-coupling-and-cohesion



WAYNE P. STEVENS

"The degree of coupling established by a particular connection is a function of several factors, and thus it is <u>difficult to establish</u> a simple index of coupling. Coupling depends (1) on how complicated the connection is, (2) on whether the connection refers to the module itself or something inside it, and (3) on what is being sent or received."

— Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974. doi:10.1147/sj.132.0115



Source: https://nordicapis.com/the-difference-between-tight-coupling-and-loose-coupling/

\*\*Coupling Between Objects (CBO) — for a class is a Metrics Suite for Object Oriented Design Sun A. Metrics Suite for Object Oriented Design Sun A. Class of Suite for Object Oriented Design Sun A. Metrics Suite for Object Suite Suite

#### A Hierarchical Model for Object-Oriented Design Quality Assessment

JAGDISH BANSIYA AND CARL G. Davis

"Direct Class Coupling (DCC) — this metric is a count of the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in methods."

— J. Bansiya and C.G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering, 28, 2002. doi:10.1109/32.979986



Martin Fowler

"The biggest problems come from uncontrolled coupling at the <u>upper levels</u>. I don't worry about the number of modules coupled together, but I look at the pattern of dependency relationship between the modules."

— M. Fowler. Reducing Coupling. *IEEE Software*, 18, 2001. doi:10.1109/ms.2001.936226

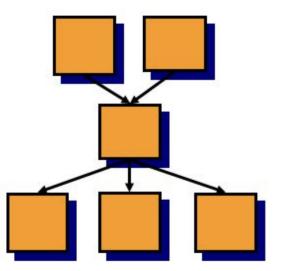


STEVE McConnell

"Low-to-medium fan-out means having a given class use a low-to-medium number of other classes. High fan-out (more than about seven) indicates that a class uses a large number of other classes and may therefore be overly complex. High fan-in refers to having a high number of classes that use a given class. High fan-in implies that a system has been designed to make good use of utility classes at the lower levels in the system."

— Steve McConnell. *Code Complete*. Pearson Education, 2004. doi:10.5555/1096143

Fan-in = number of ingoing dependencies Fan-out = number of outgoing dependencies



Heuristic: a high fan-in/fan-out indicates a high complexity

(c) Natalia Kokash, Leiden Institute of Advanced Computer Science

#### An Evolutionary Study of Fan-in and Fan-out Metrics in OSS

A. Mubarak, S. Counsell and R.M. Hierons Department of Information Systems and Computing, Brunel University

Absorbe. Exercise coupling between object-circuited clauses, a sligher propassing for fault in systems and 'stored say' find a sligher propassing for faults in systems and 'stored say' find seven Tasa's and 'fine-sai' complies mixture over minight seven that of the seven the say of the seven that the relationship between the two metrics is determine patterns when the same of the seven of first. The III like is tool was no seven the seven of first the III like is tool was not seven the seven of first the III like is tool was not seven the seven of first the III like is tool was not seven the seven of first the III like is tool was not seven the seven of first the III like is too was not seven the seven of first the III like is too was not seven the III like is the clauses to explain the big has not be tweed first in the clauses to explain the big has not severely first the seven of first in the clauses to explain the big has not sell only the seven of first in the clause to explain the big has not sell only the seven of first in the clause to explain the big has not sell only the seven of the seven the seven of the seven the seven of the seven that the seven of the seven the seven that the seven th

ywords-coupling, Java, fan-in, fan-out, package.

I. INTRODUCTION

the Object-Oriented (OO) community that executive couple between clauses receives a level of complexity that complicate subsequent maintenance and respected in a complexity subsequent to emplexity and complexity of the clause is in itself candidate for emplexity of the complexity o this paper, we investigate versions of five Open Source votation (OSS) Gooding on two well-shown coupling metrics (annive (i.e., incoming coupling) and 'fine-out' (i.e., capping, spelling). We used in autoentated dool to extract each of the spelling, are seen autoentated tool to extract each of the spelling metrics from those five systems. The research spelling metrics from those five systems. The research repetition we explore are first, is it the central cases with type incoming coupling naturally have low outgoing coupling, as a second, does this relationship women over time!' In other ords, does the potential maintenance problem become werse terms of fan-in and fan-out values?

I. MOTIVATION AND RELATED WORK

Findly, previous resized [15] has shown that there is a tradeoff between coughing types: — By particular, that between coughing through imported packages and the introduction of the potential characteristic and trade-off-between fine-in and flavous metrics over time. Second, we would always expecded evolupes: frough the characteristic and trade-off-between fine-in and flavous metrics over time. Second, we would always expecded evolupes: through the characteristic and trade-off-between previous present and the contraction of the previous research [16] which showed that the fancin and flavsure and the contraction of the previous research [16] which showed that the fancin and flavment of the contraction of the previous research [16] which showed that the fancin and flavshows a previous contraction of the contraction of the previous research of the contraction of the previous research of the contraction of the contraction of the previous research of the contraction of the contraction of the second of the contraction of the contraction

In terms of related work, the research presented relates to areas of software evolution, coupling metrics and the use of OSS B[]. In terms of software evolution, the laws of Lethman [2] provide the backdrop for many past evolutionary studies. Evolution has also been the subject of simulation studies [18] and this has allowed OSS evolution to be studied in as contrasting way to that empirically. The research presented in

"We also found evidence of certain 'key' classes (with both high fan-in and fan-out) and 'client' and 'server'-type classes with just high fan-out and fan-in, respectively."

— A. Mubarak, S. Counsell, and R.M. Hierons. An Evolutionary Study of Fan-In and Fan-Out Metrics in OSS. In *Proceedings of the Fourth International Conference on Research Challenges in Information Science (RCIS)*, 2010. doi:10.1109/rcis.2010.5507329

A. Mubarak et al.

Fan-out, as a metric, is supported by a few tools:

- Checkstyle for Java
- $\bullet$  <u>CCCC</u> for C++, C, and Java
- module-coupling-metrics for Python



DEREK COMARTIN

"Afferent coupling (denoted by  $\mathbf{Ca}$ ) is a metric that indicates the total number of other projects/boundaries that are dependent upon it. Efferent coupling (denoted by  $\mathbf{Ce}$ ) is another metric that is the verse of Afferent Coupling. It is the total number of projects that a given project depends on. Instability another metric that is a ratio:  $\mathbf{I} = \mathbf{Ce}/(\mathbf{Ce} + \mathbf{Ca})$ . This metric is a ratio between 0 and 1. With 0 meaning it's totally stable and 1 meaning it's unstable."

— Derek Comartin. Write Stable Code Using Coupling Metrics. https://codeopinion.com/write-stable-code-using-coupling-metrics/, 2021. [Online; accessed 15-03-2024]

## Types of Coupling (some of them)

- Content Coupling is when one module modifies or relies on the internal workings of another module (e.g., accessing local data of another module).
- Global Coupling is when two modules share the same global data (e.g., a global variable).
- External Coupling occurs when two modules share an externally imposed data format, communication protocol, or device interface.
- <u>Control Coupling</u> is one module controlling the flow of another, by passing it information on what to do (e.g., passing a what-to-do flag).
- Stamp Coupling is when modules share a composite data structure and use only a part of it, possibly a different part (e.g., passing a whole record to a function that only needs one field of it).

- <u>Data Coupling</u> is when modules share data through, for example, parameters. Each datum is an elementary piece, and these are the only data shared (e.g., passing an integer to a function that computes a square root).
- Message Coupling can be achieved by state decentralization (as in objects) and component communication is done via parameters or message passing (see Message passing).
- <u>Subclass Coupling</u> describes the relationship between a child and its parent. The child is connected to its parent, but the parent isn't connected to the child.
- <u>Temporal Coupling</u> is when two actions are bundled together into one module just because they happen to occur at the same time.

#### Source:

https://wiki.edunitas.com/IT/en/114-10/Coupling-(computer-programming)\_1430\_eduNitas.html

## Fear of Decoupling

```
interface Money {
  double cents();
}

void send(Money m) {
  double c = m.cents();
  // Send them over via the API...
}

class OneDollar implements Money {
  @Override
  double cents() {
  return 100.0d;
  }
}
```

```
class EmployeeHourlyRate
implements Money {
    @Override
    double cents() {
        // Fetch the exchange rate;
        // Update the database;
        // Calculate the hourly rate;
        // Return the value.
}
```

"Polymorphism makes sofware more fragile ... to make it more robust!"

## Temporal Coupling

### Tight coupling (not good):

```
List<String> list =
new LinkedList<>();
Foo.append(list, "Jeff");
Foo.append(list, "Walter");
return list;
```

#### Loose coupling (good):

```
return Foo.with(
Foo.with(
new LinkedList<>(),
"Jeff"
),
"Walter"
);
```

### Distance of Coupling

```
class Temperature {
  private int t;
  public String toString() {
    return String.format("%d F", this.t);
  }
}

Temperature x = new Temperature();
String txt = x.toString();
String[] parts = txt.split(" ");
int t = Integer.parseInt(parts[0]);
```

"The larger the number (or the mean of all numbers), the worse the design: in good design we are not supposed to take something out of a method and then do some complex processing. The distance metric will tell us exactly that: how many times, and by how much, we violated the principle of loose coupling."

https://www.yegor256.com/2020/10/27/distance-of-coupling.html

# Read this:

New Metric: the Distance of Coupling (2020)

Fear of Decoupling (2018)

Reflection Means Hidden Coupling (2022)

### References

- J. Bansiya and C.G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Transactions on Software Engineering*, 28, 2002. doi:10.1109/32.979986.
- Shyam R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6): 476–493, 1994. doi:10.1109/32.295895.
- Derek Comartin. Write Stable Code Using Coupling Metrics. https://codeopinion.com/write-stable-code-using-coupling-metrics/, 2021. [Online; accessed 15-03-2024].

- M. Fowler. Reducing Coupling. *IEEE Software*, 18, 2001. doi:10.1109/ms.2001.936226.
- Steve McConnell. *Code Complete*. Pearson Education, 2004. doi:10.5555/1096143.
- A. Mubarak, S. Counsell, and R.M. Hierons. An Evolutionary Study of Fan-In and Fan-Out Metrics in OSS. In *Proceedings of the Fourth International Conference on Research Challenges in Information Science (RCIS)*, 2010. doi:10.1109/rcis.2010.5507329.
- Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974. doi:10.1147/sj.132.0115.