

YEGOR BUGAYENKO

Lecture #6 out of 24 80 minutes

The slidedeck was presented by the author in this YouTube Video

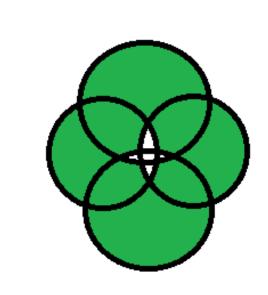
All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.



LARRY L. CONSTANTINE

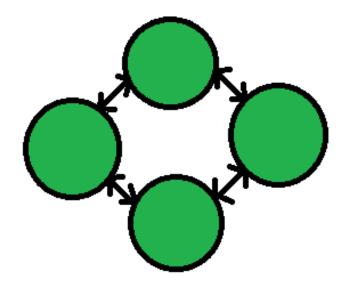
"The fewer and simpler the connections between modules, the easier it is to <u>understand</u> each module without reference to other modules."

— 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



### Tight coupling:

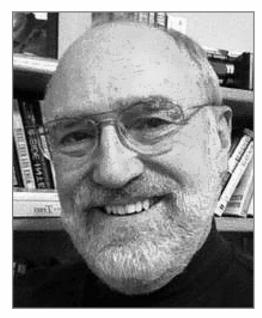
- 1. More Interdependency
- 2. More coordination
- 3. More information flow



### Loose coupling:

- 1. Less Interdependency
- 2. Less coordination
- 3. Less information flow

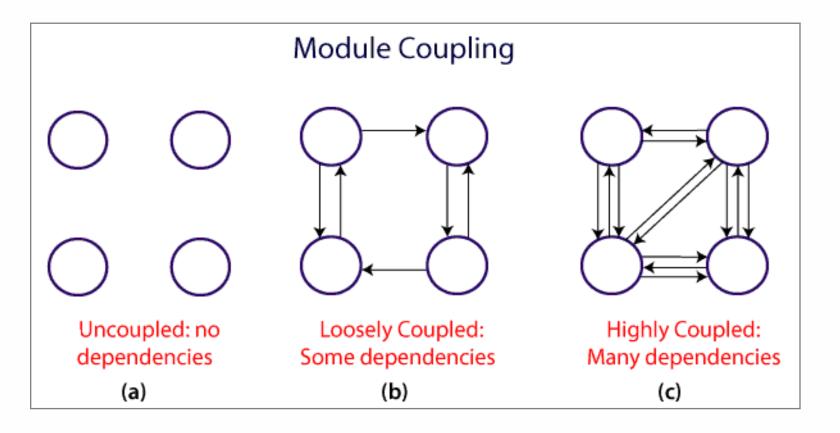
Source: https://www.geeksforgeeks.org/coupling-in-java/



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 difficult to establish 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/





**GRADY BOOCH** 

"However, there is tension between the concepts of coupling and inheritance because inheritance introduces significant coupling. On the one hand, weakly coupled classes are desirable; on the other hand, inheritance—which tightly couples superclasses and their subclasses—helps us to exploit the commonality among abstractions."

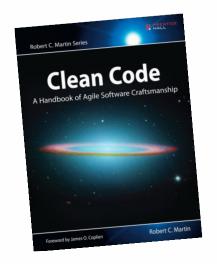
— Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen, and Kelli A. Houston. *Object-Oriented Analysis and Design With Applications*. Addison-Wesley, 1994. doi:10.5555/1407387

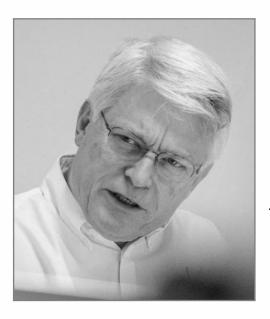




"Collaborations always involve some degree of coupling between both parties of the collaboration, so the <u>number of collaborations</u> should be minimized to the greatest extent possible."

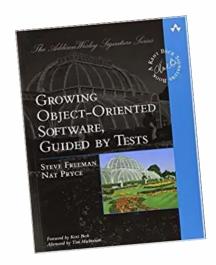
— David West. *Object Thinking*. Pearson Education, 2004. doi:10.5555/984130





"The lack of coupling means that the elements of our system are <u>better isolated</u> from each other and from change. This isolation makes it easier to understand each element of the system."

— Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008. doi:10.5555/1388398





STEVE FREEMAN

"Elements are coupled if a change in one <u>forces</u> a change in the other. Loosely coupled features are easier to maintain."

— Steve Freeman and Nat Pryce. *Growing Object-Oriented Software, Guided by Tests.* Pearson Education, 2009. doi:10.5555/1655852

\*\*Coupling Between Objects (CBO) — for a class is a \*\*Coupling Between Objects (CBO) — for a \*\*Coupling Between Objects (CBO) — for a class is a \*\*Coupling Between Objects (CBO) — for a \*\*Coupling Betwe

Coupling @yegor256

1994. doi:10.1109/32.295895

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

"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."

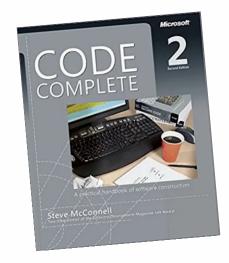
— J. Bansiya and C. G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering, 2002. doi:10.1109/32.979986



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 <u>dependency relationship</u> between the modules."

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



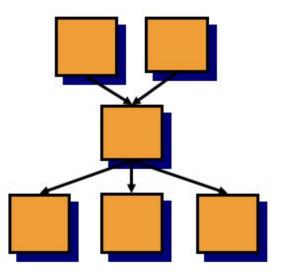


STEVE McConnell

"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 partment of Information Systems and Computing, Brunel University Uxbridge, UK. Email: steve.counsell@brunel.ac.uk

Abstrace Exercise coupling between object-oriented classes in which packnowledge an animetenesy problem that care read it. A light any proposed for the last level as early a transcription of the last level as early as the early

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

specially for failts in software [5], it is widely believed in special properties of the special properties of the special special properties of the special properties a "street in ministensies problem. In practice, a close that is highly special properties of the special properties of the special placetring or removal from the system to mitigate both placetring or removal from the system to mitigate both mendiately arises however for the developer when mediately arises however for the developer when the special properties of the special properties of the other control properties of the special properties of the then are those coupling dependencies? If the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special properties of the special properties of the special special special properties of the special properties of the special special properties of the special properties of the special special special properties of the special properties of the special special properties of the special special properties of the special s In this paper, we investigate versions of five Open Source Systems (OSS) focusing on two well-known coupling metrics "alma" (i.e., incoming coupling and "alma" (i.e., outgoing coupling, "alma" (i.e., incoming coupling and "alma" (i.e., outgoing coupling, We used an automated tool to extract each of the coupling metrics from those five systems. The research questions we explore are first, is it the case that classes with large innoming coupling naturally have been outgoing coupling anaturally have been outgoing coupling anatural year bow outgoing coupling anatural years over time? In other words, does the potential naturalense persobbe theorems were such constraint an automate problem becomes were a constraint an automate problem becomes were an automated and a constraint an automate problem becomes were an automated and a constraint an automate problem becomes were an automated and a constraint and a con

II. MOTIVATION AND RELATED WORK

The reasent in this paper is motivated by a number of lactor of the control of th

In terms of related work, the research presented relates to areas of software evolution, coupling metrics and the use of OSS [8]. In terms of software evolution, the laws of Lehman [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 a contrasting way to that empirically. The research presented in this paper delves into specific evolutionary coupling features

"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 4th International Conference on Research Challenges in Information Science (RCIS)*, 2010. doi:10.1109/rcis.2010.5507329

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

- Checkstyle for Java
- CCCC for C++, C, and Java
- module-coupling-metrics for Python
- effrit for Go
- lizard for JavaScript, C#, TypeScript, Lua, Rust, etc.



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.
- Temporal Coupling 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 robust!"

Source: Yegor Bugayenko. Fear of Decoupling. https://www.yegor256.com/180918.html, sep 2018. [Online; accessed 22-09-2024]

# 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"
);
```

Source: Yegor Bugayenko. Temporal Coupling Between Method Calls. https://www.yegor256.com/151208.html, dec 2015. [Online; accessed 22-09-2024]

# 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, 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 how many times, and by how much, we violated the principle of loose coupling."

Source: Yegor Bugayenko. New Metric: The Distance of Coupling.

https://www.yegor256.com/201027.html, oct 2020. [Online; accessed 22-09-2024]

## References

- J. Bansiya and C. G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Transactions on Software Engineering*, 2002. doi:10.1109/32.979986.
- Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen, and Kelli A. Houston. *Object-Oriented Analysis and Design With Applications*. Addison-Wesley, 1994. doi:10.5555/1407387.
- Yegor Bugayenko. Temporal Coupling Between Method Calls.
  - https://www.yegor256.com/151208.html, dec 2015. [Online; accessed 22-09-2024].
- Yegor Bugayenko. Fear of Decoupling. https://www.yegor256.com/180918.html, sep 2018. [Online; accessed 22-09-2024].
- Yegor Bugayenko. New Metric: The Distance of Coupling.
  - https://www.yegor256.com/201027.html, oct

- 2020. [Online; accessed 22-09-2024].
- 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*, 2001. doi:10.1109/ms.2001.936226.
- Steve Freeman and Nat Pryce. *Growing Object-Oriented Software, Guided by Tests.*Pearson Education, 2009. doi:10.5555/1655852.
- Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008. doi:10.5555/1388398.
- 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 4th 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.

David West. *Object Thinking*. Pearson Education, 2004. doi:10.5555/984130.