

YEGOR BUGAYENKO

Lecture #6 out of 24 80 minutes

The slidedeck was presented by the author in this YouTube Video

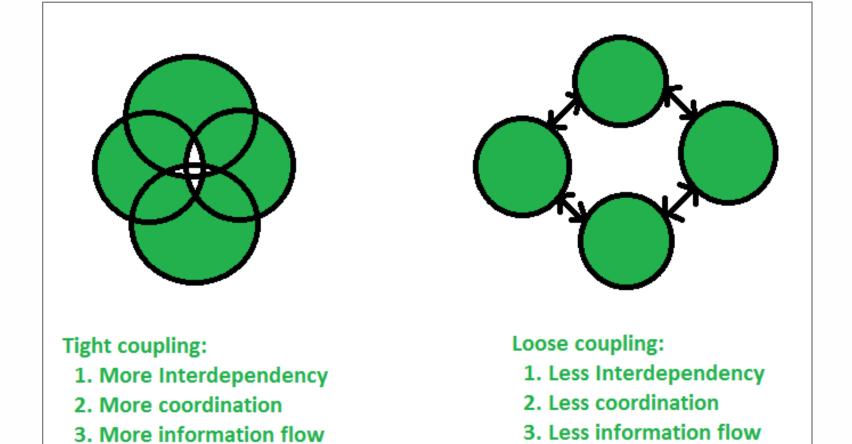
All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.



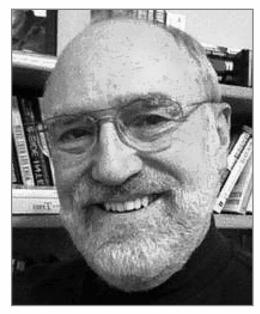
LARRY L. CONSTANTINE

"The fewer and simpler the connections between modules, the easier it is to 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 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/

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

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 dependency relationship between the modules."

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

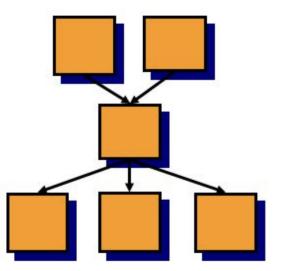


STEVE McConnell

"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

Admore. Exemple capting between object-oriented clauses is a kingle preparation from the system and a trivent of friend and subject proparation from its system and a strong of friend and exemple control of the system and the system and a strong of the system of the system and system and the system and a control of system and system and the system and the system and provide in each were the course of time. The Illiest knowl was not a carried the two antices from the system control of the course of the system and the system and the system and the characteristics of clause subhiding the highest fine-in values of the system and the system and the system and the system of the system and the dates which spin table shows and finally, respectively. We provide clause with glast high fine-out and finally, respectively. We provide that for dispositional increases in each of the system and not all not for the system and not seen and seed of the system and the syst

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

processing for fault is notwer [5,3]. It is widely believed the Object-Oriented (Oo) community that excessive coupling the Object-Oriented (Oo) community that excessive coupling the Object-Oriented (Oo) community that excessive coupling the Object-Oriented (Oo) community that excessive the Object-Oriented (Oo) coupled to many other classes is an ideal candidate for registering or removal them the system to mitigate both immediately artises however for the developer when considering re-engineering of classes with high coupling is proceeding that the Object-Oriented (Oo) coupled (Oo) c

In this apper, we investigate versions of five Open Source System (CSS) Gootsing in the well-known coupling nettice: -fine in (Le, incoming coupling) and fine-out (a, oraging, coupling). We used an automated tool to extract each of the coupling metrics from those five systems. The research questions we explore our first, is the case that classes with large incoming coupling naturally have low estipolic goodping and second, does this relationshy women over time? In other works, does the potential maniferance problem become worse in terms of fine-in and fine-our values?

II. MOTIVATION AND RELATED WORK

The reason is this paper is motivated by a number of factors. Trails, previous necessical [15] has shown that there is a traderist, previous necessical [15] has shown that there is a tradecoupling through imported packages and the introduction of internal-only-package coupling. In this paper, we explore the potential transcertictic and trade-offs between finest and proportically problematic clauses to be recognitive of the protection of the complete of the proportically problematic clauses to be recognitived by developes through techniques such as reflectoring [17] developes through techniques such as reflectoring [17] and their disposal means that only when clauses calculated and with, in this gray, we explore, one time, whether smalls and with, in this gray, we explore, no exit, whether smalls what proportions. Finally, the research is motivated by what previous research [10] which showed that the fines in and fineous matrix smalled to be relatively small for clauses removed and our matrix smalled to be relatively small for clauses removed and another manufactories of the proportion. Finally, the research is motivated by what proportions. Finally, the research is motivated by what proportions. Finally, the research is motivated by what proportions. Finally, the research is motivated by what proportions of the proportion 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
- \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*, 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*, 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 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.