Coupling

Yegor Bugayenko

Lecture #6 out of 24
80 minutes

The slidedeck was presented by the author in this YouTube Video

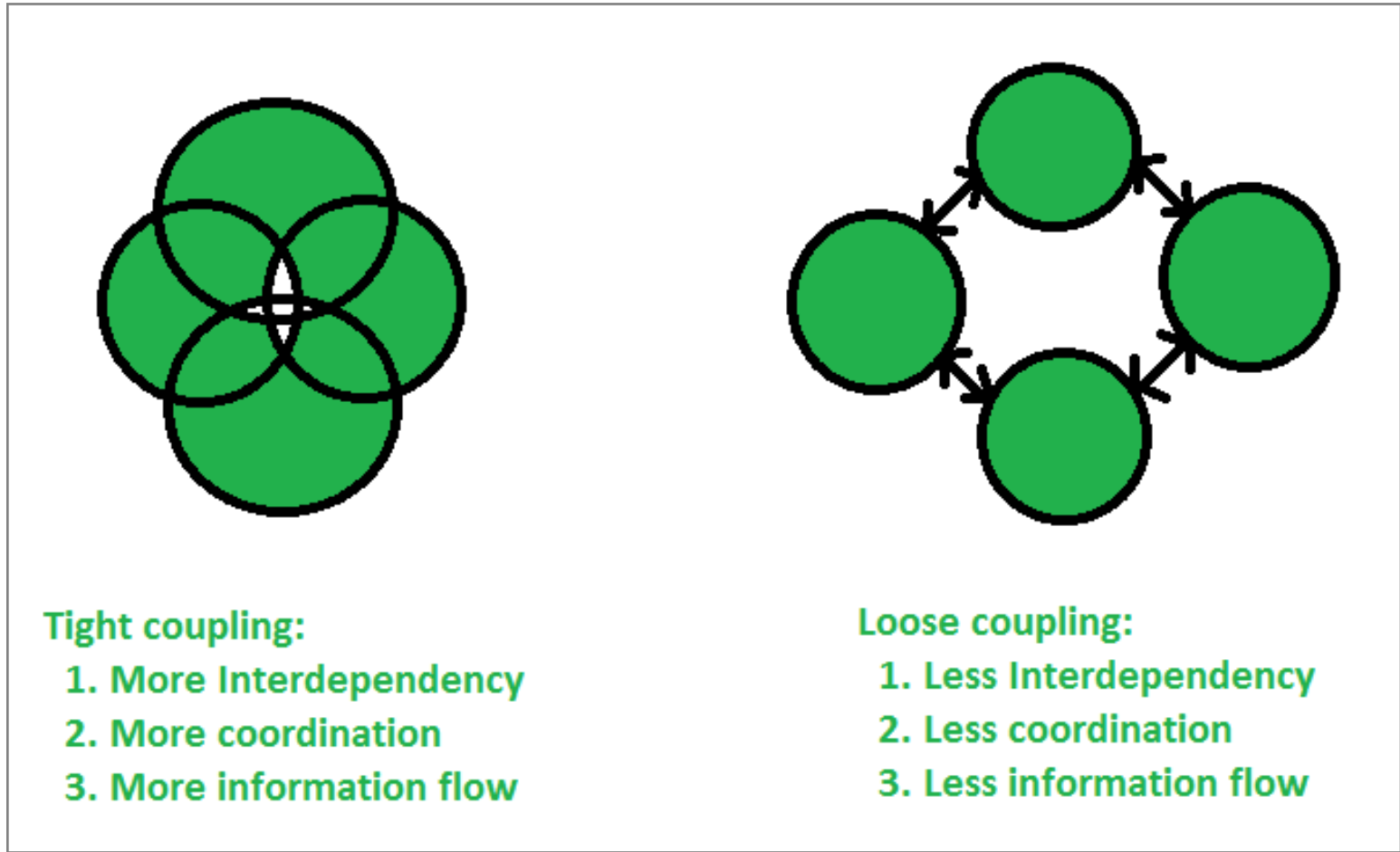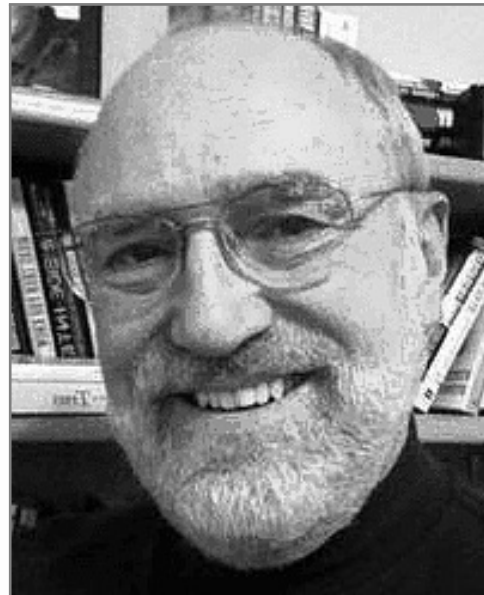"The fewer and simpler the connections between modules, the easier it is to <u>understand</u> each module without reference to other modules."

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

LARRY L. CONSTANTINE

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/
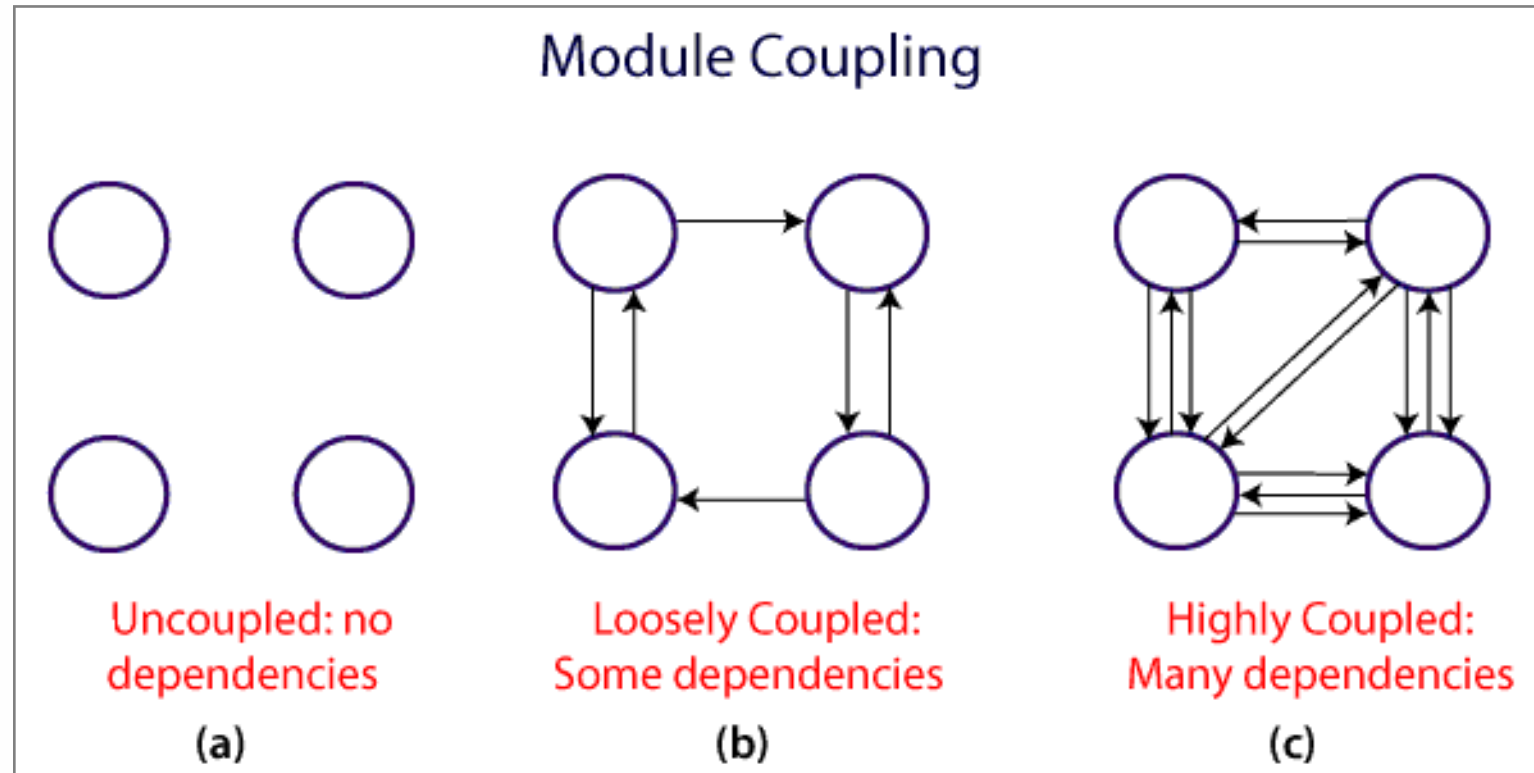
Glenford Myers

"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

Module Coupling

Uncoupled: no dependencies
(a)

Loosely Coupled: Some dependencies
(b)

Highly Coupled: Many dependencies
(c)

Source: https://www.javatpoint.com/software-engineering-coupling-and-cohesion
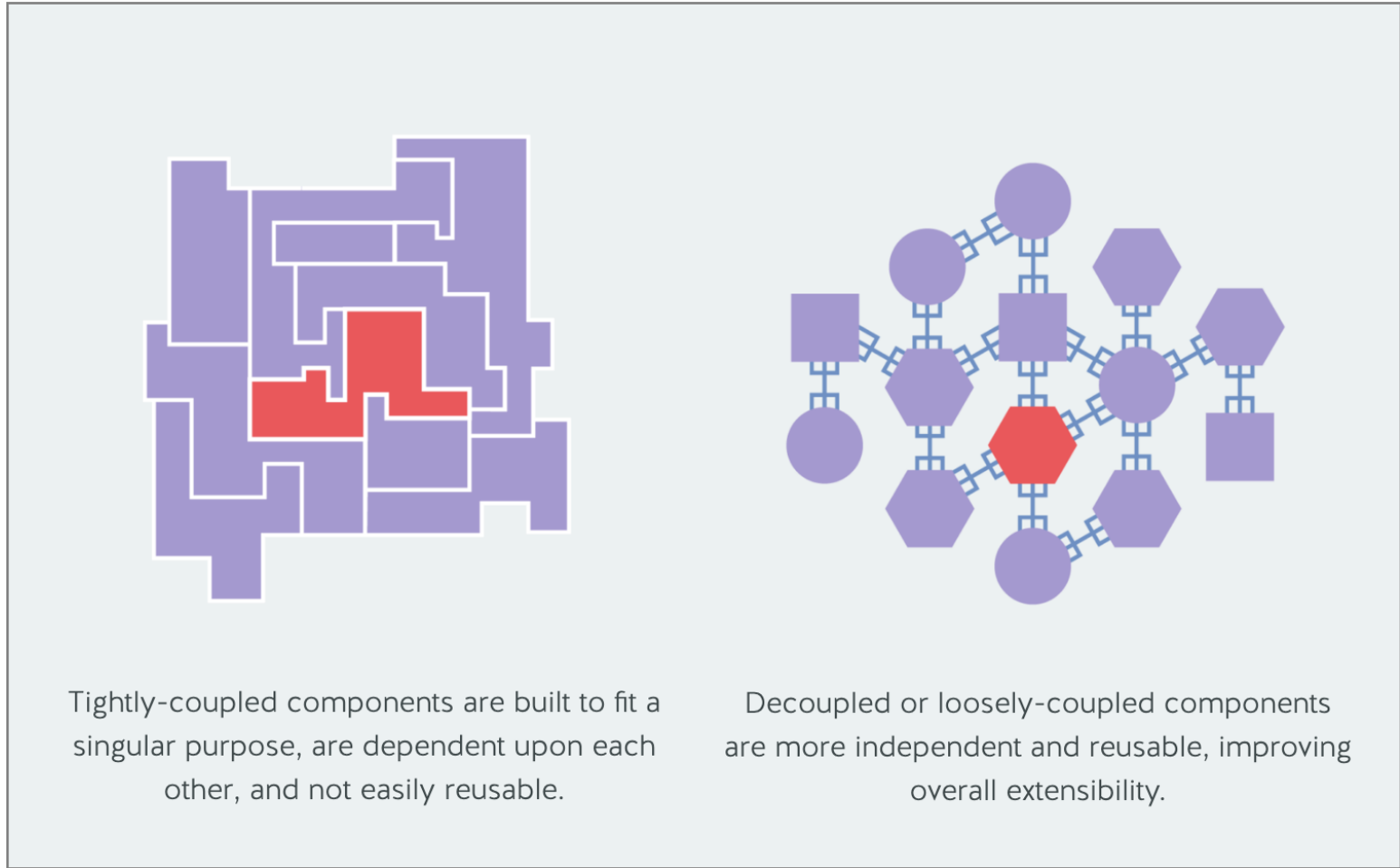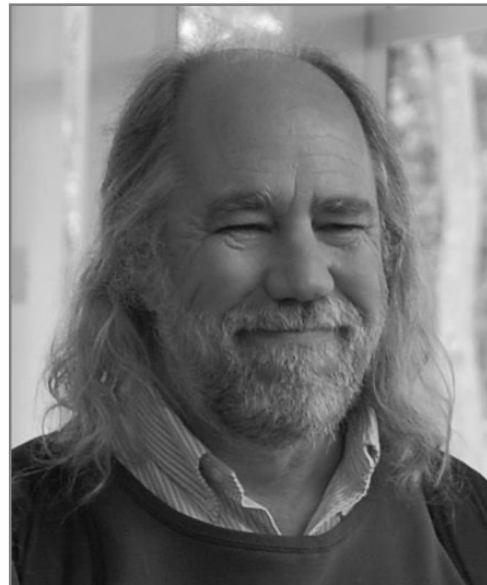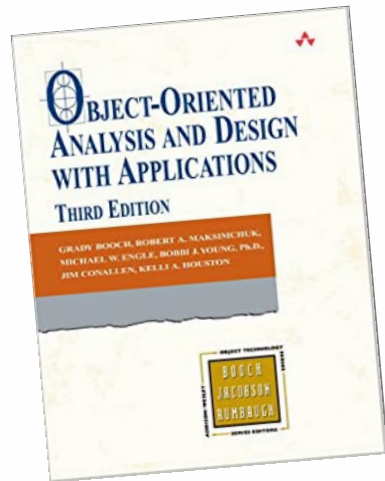
"The degree of coupling established by a particular connection is a function of several factors, and thus it is difficult to establish a simple index of coupling. Coupling depends (1) on how complicated the connection is, (2) on whether the connection refers to the module itself or something inside it, 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

WAYNE P. STEVENS

Tightly-coupled components are built to fit a singular purpose, are dependent upon each other, and not easily reusable.

Decoupled or loosely-coupled components are more independent and reusable, improving overall extensibility.

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

Object-Oriented Analysis and Design with Applications, Third Edition

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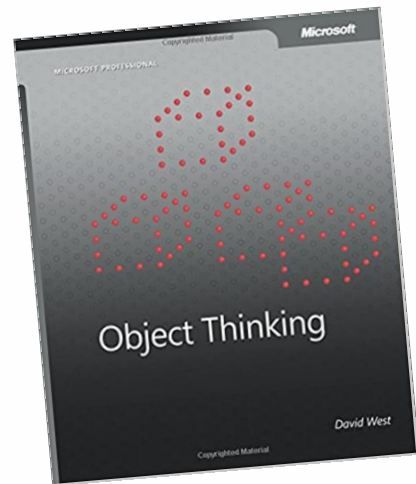
"Collaborations always involve some degree of coupling between both parties of the collaboration, so the number of collaborations should be minimized to the greatest extent possible."

— David West. *Object Thinking*. Pearson Education, 2004. doi:10.5555/984130
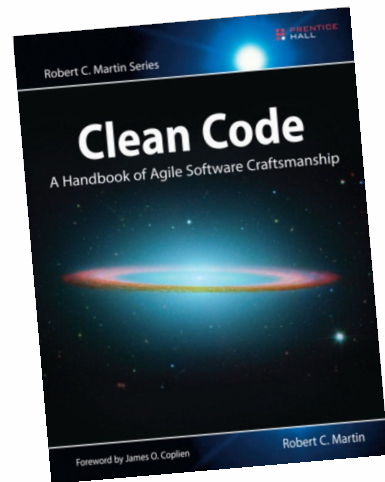
"The lack of coupling means that the elements of our system are <u>better isolated</u> from each other and from change. This isolation makes it easier to <u>understand</u> each element of the system."

— Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship.* Pearson Education, 2008. doi:<u>10.5555/1388398</u>
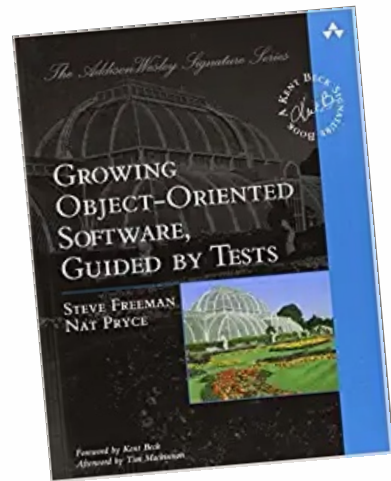
"Elements are coupled if a change in one <u>forces</u> a change in the other. Loosely coupled features are easier to <u>maintain</u>."

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

Steve Freeman

"Coupling Between Objects (CBO) — for a class is a count of the number of other classes to which it is coupled."

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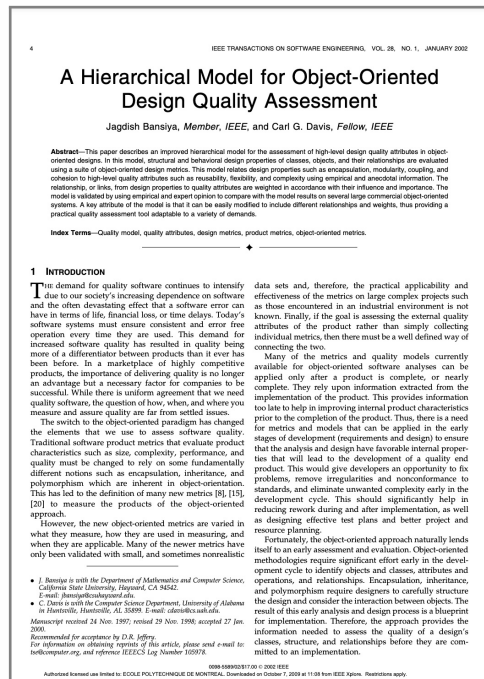
"Direct Class Coupling (DCC) — this metric is a count of the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in methods."

— 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

"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.
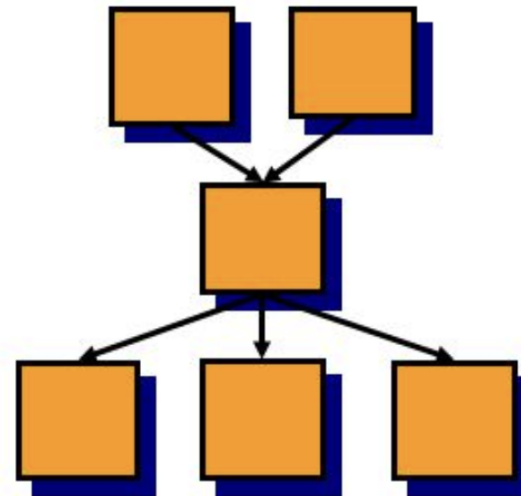doi:<u>10.1109/ms.2001.936226</u>

Martin Fowler
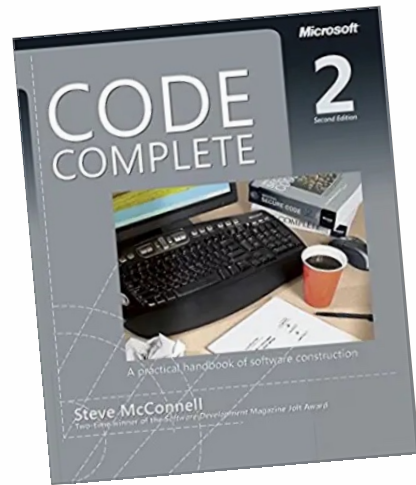
Steve McConnell

"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

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

"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 **Ca**) is a metric that indicates the total number of other projects/boundaries that are dependent upon it. Efferent coupling (denoted by **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: $I = Ce/(Ce + Ca)$. 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.

- Control Coupling 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).

- Data Coupling 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).

- Subclass Coupling 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

```
1  interface Money {
2    double cents();
3  }
4
5  void send(Money m) {
6    double c = m.cents();
7    // Send them over via the API...
8  }
9
10 class OneDollar implements Money {
11   @Override
12   double cents() {
13     return 100.0d;
14   }
15 }
```

```
1  class EmployeeHourlyRate
2    implements Money {
3    @Override
4    double cents() {
5      // Fetch the exchange rate;
6      // Update the database;
7      // Calculate the hourly rate;
8      // 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):

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

Loose coupling (good):

```
1  return Foo.with(
2    Foo.with(
3      new LinkedList<>(),
4      "Jeff"
5    ),
6    "Walter"
7  );
```

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

Distance of Coupling

```
1  class Temperature {
2    private int t;
3    public String toString() {
4      return String.format("%d F", this.t);
5    }
6  }
7
8  Temperature x = new Temperature();
9  String txt = x.toString();
10 String[] parts = txt.split(" ");
11 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.