Dead Code

Yegor Bugayenko

Lecture #12 out of 24
80 minutes

The slidedeck was presented by the author in this YouTube Video

Motivating Example

Before (wrong):

```
class Book
  private int id;
  public Book(int it)
    this.id = i;
  public int getId()
    return this.id;

  private int setId(int i)
    this.id = i;
```

After (better):

```
class Book
  private final int id;
  public Book(int it)
    this.id = i;
  public int getId()
    return this.id;
```

Mika Mäntylä, Jari Vanhanen, Casper Lassenius

“Dead code is code that has been used in the past, but is currently never executed. Dead code hinders code comprehension and makes the current program structure less obvious.”

— M. Mantyla, J. Vanhanen, and C. Lassenius. A taxonomy and an initial empirical study of bad smells in code. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, 2003. doi:10.1109/icsm.2003.1235447. URL `http://dx.doi.org/10.1109/ICSM.2003.1235447`

Dead Code Elimination (Compiler Optimization)

Dead code is here:

```
void main(int x) {
  int a = 42;
  if (x > 0) {
    a = 256;
  }
  a = 7;
  print(a);
}
```

"Dead code refers to computations whose results are never used. Code that is dead can be eliminated without affecting the behavior of the program."

Source: *Compiler Techniques for Code Compaction*, Saumya K. Debray, William Evans, Robert Muth, Bjorn De Sutter, ACM Transactions on Programming languages and Systems (TOPLAS), 22(2), 2000

Simone Romano,
Christopher Vendome,
Giuseppe Scanniello,
Denys Poshyvanyk

"Although there is some consensus on the fact that dead code is a common phenomenon, it could be harmful, and it seems to matter to software professionals; surprisingly, dead code has received very little empirical attention from the software engineering research community."

— Simone Romano, Christopher Vendome, Giuseppe Scanniello, and Denys Poshyvanyk. A Multi-Study Investigation into Dead Code. *IEEE Transactions on Software Engineering*, 2018. doi:10.1109/tse.2018.2842781

Unreachable/Dead Methods in Java

Table 1: Dataset Information

Software	LOCs	#Types	#Meth.	#Un. Meth.	%Un. Meth.
ArtOfIllusion 2.4.1	79,383	600	5,426	545	10%
LaTeXDraw 2.0.8	65,334	252	3,130	212	7%
aTunes 1.10.1	42,357	778	4,067	240	6%
MediaPesata 1.0	1,580	31	162	8	5%

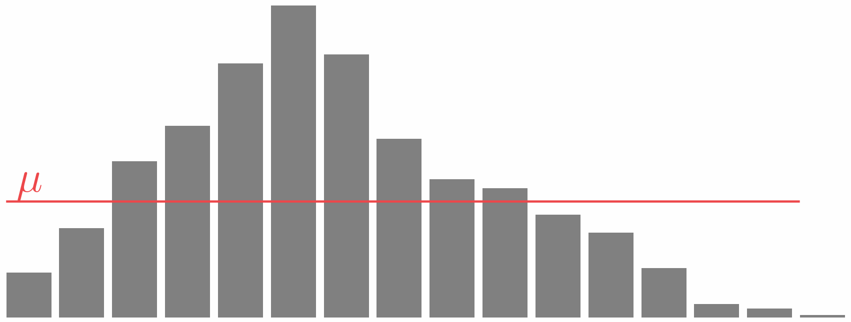
Source: *A Graph-based Approach to Detect Unreachable Methods in Java Software*, Simone Romano, Giuseppe Scanniello, Carlo Sartiani, Michele Risi, Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016

Sebastian Eder, Maximilian Junker, Benedikt Hauptmann, Elmar Juergens, Rudolf Vaas, Karl-Heinz Prommer

"We conducted the study on the level of methods in the sense of object oriented programming. The systems contains 25,390 methods. We found that 25% of all methods were never used during the complete period."

— Sebastian Eder, Maximilian Junker, Elmar Jurgens, Benedikt Hauptmann, Rudolf Vaas, and Karl-Heinz Prommer. How much does unused code matter for maintenance? In *Proceedings of the 34th International Conference on Software Engineering*, 2012

Volatility Metric



"The variance $Var(g)$ is the **Volatility** of the source code. The smaller the Volatility the more *cohesive* is the repository and the smaller the amount of the abandoned code inside it."

Then, the mean μ is calculated as:

$$\mu = \frac{1}{Z} \sum_{j=1}^{Z} g_j \tag{5}$$

Finally, the variance is calculated as:

$$\mathrm{Var}(g) = \frac{1}{Z} \sum_{j=1}^{Z} |g_j - \mu|^2 \tag{6}$$

The variance $\mathrm{Var}(g)$ is the Volatility of the source code.

Volatility vs. Number of Files in a Repo

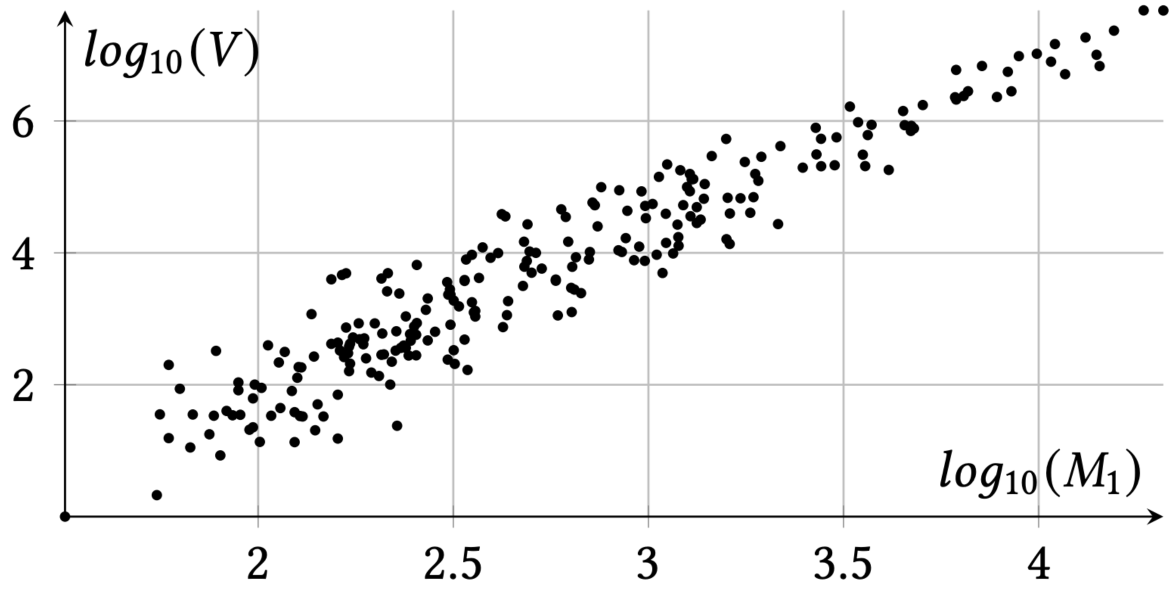


Figure 3. The relationship between the number of files in the repository and its Volatility (Z is set to 32)

Monolithic Repositories

Centralization The codebase is contained in a single repo encompassing multiple projects.

Visibility Code is viewable and searchable by all engineers in the organization.

Synchronization: The development process is trunk-based; engineers commit to the head of the repo.

Completeness Any project in the repo can be built only from dependencies also checked into the repo. Dependencies are unversioned; projects must use whatever version of their dependency is at the repo head.

Standardization A shared set of tooling governs how engineers interact with the code, including building, testing, browsing, and reviewing code.

Source: *Advantages and Disadvantages of a Monolithic Repository: A case study at Google*, Ciera Jaspan et al., ICSE, 2018

Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K. Smith, Collin Winter, Emerson Murphy-Hill

"Our survey results show that engineers at Google strongly prefer our monolithic repo, and that visibility of the codebase and simple dependency management were the primary factors for this preference."

— Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K. Smith, Collin Winter, and Emerson Murphy-Hill. Advantages and disadvantages of a monolithic repository: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018

Rachel Potvin and Josh Levenberg

"The Google codebase includes approximately one billion files and has a history of approximately 35 million commits spanning Google's entire 18-year existence. The repository contains 86TBa of data, including approximately two billion lines of code in nine million unique source files."

— Rachel Potvin and Josh Levenberg. Why Google stores billions of lines of code in a single repository. *Communications of the ACM*, 2016. doi:10.1145/2854146

Durham Goode et al.

"Facebook's main source repository is enormous—many times larger than even the Linux kernel, which checked in at 17 million lines of code and 44,000 files in 2013."

— Durham Goode et al. Scaling Mercurial at Facebook, 2014. URL https://engineering.fb.com/2014/01/07/core-infra/scaling-mercurial-at-facebook/

Tomas Votruba

"Before monorepo, I had to upgrade every package manually, which resulted in dissonance: one package used Symfony\Console 3.2, but other only 2.8 and it got messy <u>for no reason</u>."

— Tomas Votruba. How Monolithic Repository in Open Source saved my Laziness, 2017. URL `https://tomasvotruba.com/blog/2017/01/31/how-monolithic-repository-in-open-source-saved-my-laziness`

Benefits of “Manyrepo” Approach

Encapsulation Each repo encapsulates and hides its details from everybody else.

Fast Builds When a repo is small, the time its automated build takes is small.

Accurate Metrics Calculating LoC for a large repository doesn’t make any sense.

Homogeneous Tasks It’s easier to make tasks similar in size and complexity.

Single Coding Standard Smaller repositories look more beautiful.

Short Names Smaller namespaces mean better maintainability.

Simple Tests More dependencies are difficult to mock and test.

Source: Monolithic Repos Are Evil (2018)

Read this:

Volatility Metric to Detect Anomalies in Source Code Repositories, Yegor Bugayenko, Proceedings of the 1st ACM SIGPLAN International Workshop on Beyond Code: No Code, 2021

Advantages and Disadvantages of a Monolithic Repository: A case study at Google, Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K. Smith, Collin Winter, Emerson Murphy-Hill, Proceedings of the International Conference on Software Engineering, 2018

Monolithic Repos Are Evil (2018)

References

Sebastian Eder, Maximilian Junker, Elmar Jurgens, Benedikt Hauptmann, Rudolf Vaas, and Karl-Heinz Prommer. How much does unused code matter for maintenance? In *Proceedings of the 34th International Conference on Software Engineering*, 2012.

Durham Goode et al. Scaling Mercurial at Facebook, 2014. URL `https://engineering.fb.com/2014/01/07/core-infra/scaling-mercurial-at-facebook/`.

Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K. Smith, Collin Winter, and Emerson Murphy-Hill. Advantages and disadvantages of a monolithic repository: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018.

M. Mantyla, J. Vanhanen, and C. Lassenius. A taxonomy and an initial empirical study of bad smells in code. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, 2003. doi:10.1109/icsm.2003.1235447. URL `http://dx.doi.org/10.1109/ICSM.2003.1235447`.

Rachel Potvin and Josh Levenberg. Why Google stores billions of lines of code in a single repository. *Communications of the ACM*, 2016. doi:10.1145/2854146.

Simone Romano, Christopher Vendome, Giuseppe Scanniello, and Denys Poshyvanyk. A Multi-Study Investigation into Dead Code. *IEEE Transactions on Software Engineering*, 2018. doi:10.1109/tse.2018.2842781.

Tomas Votruba. How Monolithic Repository in Open Source saved my Laziness, 2017. URL `https://tomasvotruba.com/blog/2017/01/31/how-monolithic-repository-in-open-source-saved-my-laziness`