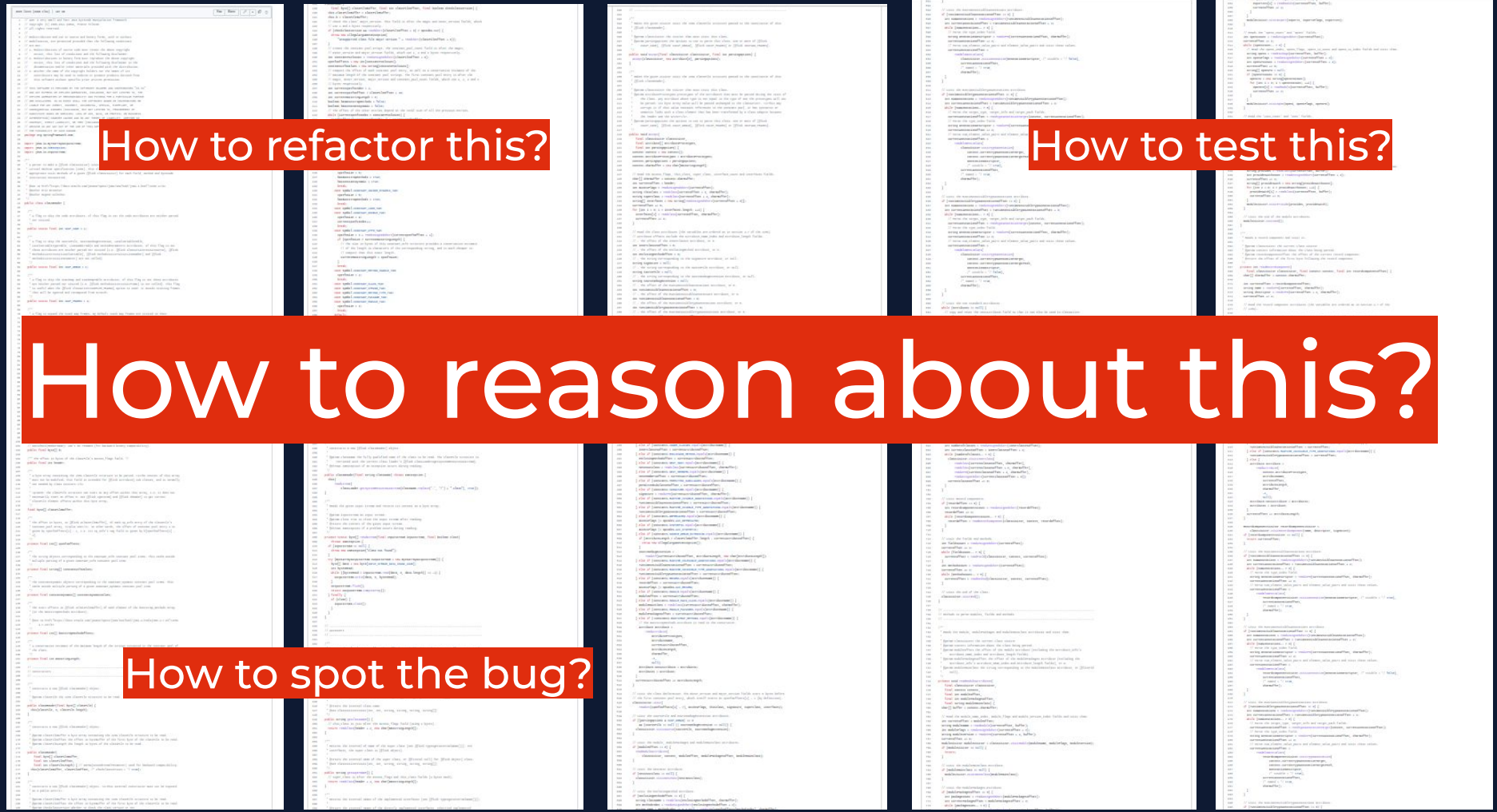
Cognitive-Driven Development Helps Software Teams to Keep Code Units Under the Limit!

Gustavo Pinto

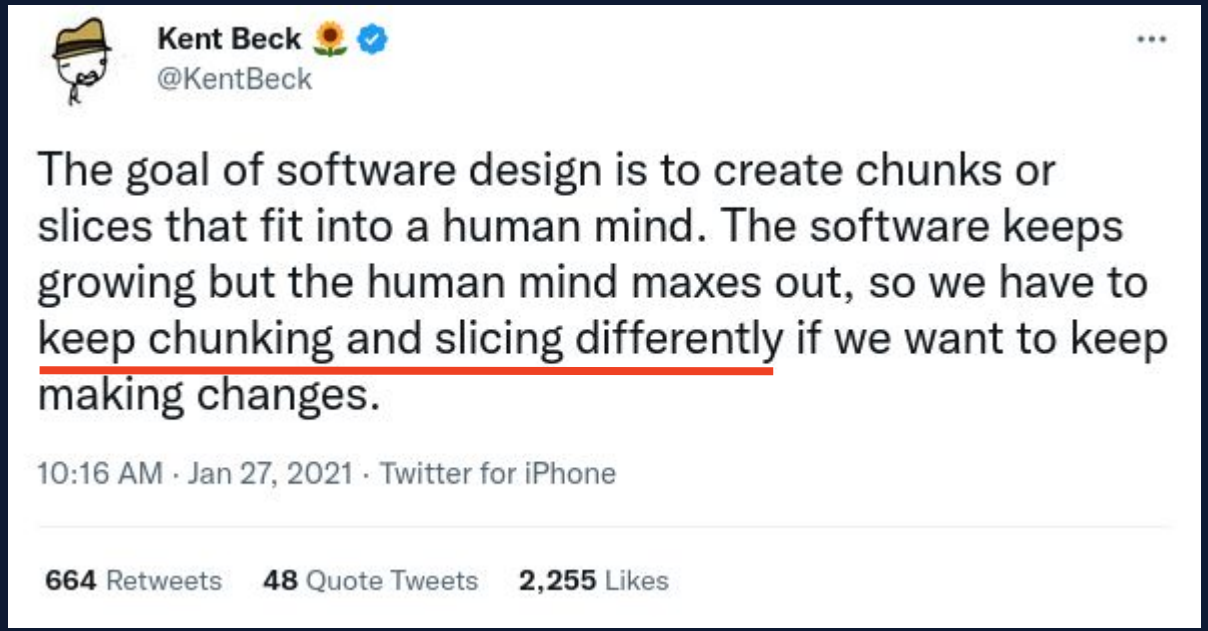
Assistant professor at UFPA.br
Head of research at ZUP.com.br

@gustavopinto



https://github.com/spring-projects/spring-framework/blob/b595dc1dfad9db534ca7b9e8f46bb9926b88ab5a/spring-core/src/main/java/org/springframework/asm/ClassReader.java (~4k LOC)

If we want to keep making changes ...



Kent Beck
@KentBeck

The goal of software design is to create chunks or slices that fit into a human mind. The software keeps growing but the human mind maxes out, so we have to keep chunking and slicing differently if we want to keep making changes.

10:16 AM · Jan 27, 2021 · Twitter for iPhone

664 Retweets **48** Quote Tweets **2,255** Likes

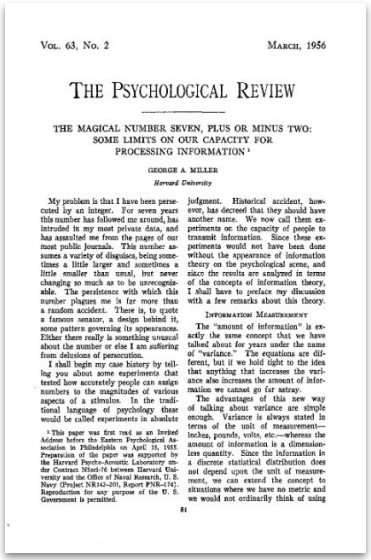
https://twitter.com/kentbeck/status/1354418068869398538


CDD is a **simple** approach
for chunking and slicing

< > Cognitive-Driven Development (CDD)

- CDD aims to reduce the developers' cognitive load during coding activities

- CDD does so by posing a limit on the number of items developers could use at once (at a class or file)



Vol. 63, No. 2 March, 1956

THE PSYCHOLOGICAL REVIEW

THE MAGICAL NUMBER SEVEN, PLUS OR MINUS TWO: SOME LIMITS ON OUR CAPACITY FOR PROCESSING INFORMATION[1]

GEORGE A. MILLER
Harvard University

My problem is that I have been persecuted by an integer. For seven years this number has followed me around, has intruded in my most private data, and has assaulted me from the pages of our most public journals. This number assumes a variety of disguises, being sometimes a little larger and sometimes a little smaller than usual, but never changing so much as to be unrecognizable. The persistence with which this number plagues me is far more than a random accident. There is, to quote a famous senator, a design behind it, some pattern governing its appearances. Either there really is something unusual about the number or else I am suffering from delusions of persecution.

I shall begin my case history by telling you about some experiments that tested how accurately people can assign numbers to the magnitudes of various aspects of a stimulus. In the traditional language of psychology these would be called experiments in absolute judgment. Historical accident, however, has decreed that they should have another name. We now call them experiments on the capacity of people to transmit information. Since these experiments would not have been done without the appearance of information theory on the psychological scene, and since the results are analyzed in terms of the concepts of information theory, I shall have to preface my discussion with a few remarks about this theory.

INFORMATION MEASUREMENT

The "amount of information" is exactly the same concept that we have talked about for years under the name of "variance." The equations are different, but if we hold tight to the idea that anything that increases the variance also increases the amount of information we cannot go far astray.

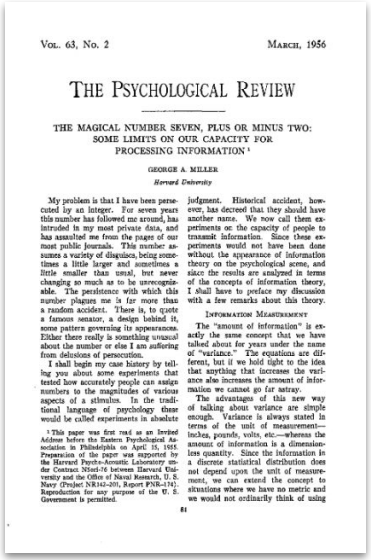
The advantages of this new way of talking about variance are simple enough. Variance is always stated in terms of the unit of measurement—inches, pounds, volts, etc.—whereas the amount of information is a dimensionless quantity. Since the information is a discrete statistical distribution does not depend upon the unit of measurement, we can extend the concept to situations where we have no metric and we would not ordinarily think of using

[1] This paper was first read as an Invited Address before the Eastern Psychological Association in Philadelphia on April 15, 1955. Preparation of the paper was supported by the Harvard Psycho-Acoustic Laboratory under Contract N6ori-76 between Harvard University and the Office of Naval Research, U. S. Navy (Project NR142-201, Report PNR-134). Reproduction for any purpose of the U. S. Government is permitted.

81

The Magical Number 7

- We are only able to process 7 (+/- 2) units of information in short-term memory.
- As we receive more information simultaneously, we lose the ability to process it (and we tend to make mistakes).



Vol. 63, No. 2 MARCH, 1956

THE PSYCHOLOGICAL REVIEW

THE MAGICAL NUMBER SEVEN, PLUS OR MINUS TWO: SOME LIMITS ON OUR CAPACITY FOR PROCESSING INFORMATION[1]

GEORGE A. MILLER

Harvard University

My problem is that I have been persecuted by an integer. For seven years this number has followed me around, has intruded in my most private data, and has assaulted me from the pages of our most public journals. This number assumes a variety of disguises, being sometimes a little larger and sometimes a little smaller than usual, but never changing so much as to be unrecognizable. The persistence with which this number plagues me is far more than a random accident. There is, to quote a famous senator, a design behind it, some pattern governing its appearances. Either there really is something unusual about the number or else I am suffering from delusions of persecution.

I shall begin my case history by telling you about some experiments that tested how accurately people can assign numbers to the magnitudes of various aspects of a stimulus. In the traditional language of psychology these would be called experiments in absolute judgment. Historical accident, however, has decreed that they should have another name. We now call them experiments on the capacity of people to transmit information. Since these experiments would not have been done without the appearance of information theory on the psychological scene, and since the results are analyzed in terms of the concepts of information theory, I shall have to preface my discussion with a few remarks about this theory.

INFORMATION MEASUREMENT

The "amount of information" is exactly the same concept that we have talked about for years under the name of "variance." The equations are different, but if we hold tight to the idea that anything that increases the variance also increases the amount of information we cannot go far astray.

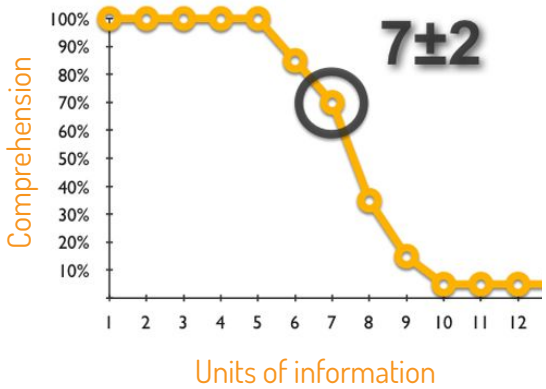
The advantages of this new way of talking about variance are simple enough. Variance is always stated in terms of the unit of measurement—inches, pounds, volts, etc.—whereas the amount of information is a dimensionless quantity. Since the information is a discrete statistical distribution does not depend upon the unit of measurement, we can extend the concept to situations where we have no metric and we would not ordinarily think of using

[1] This paper was first read as an Invited Address before the Eastern Psychological Association in Philadelphia on April 15, 1955. Preparation of the paper was supported by the Harvard Psycho-Acoustic Laboratory under Contract N6ori-76 between Harvard University and the Office of Naval Research, U. S. Navy (Project NR142-201, Report PNR-174). Reproduction for any purpose of the U. S. Government is permitted.

81

The Magical Number 7

- We are only able to process 7 (+/- 2) units of information in short-term memory.
- As we receive more information simultaneously, we lose the ability to process it (and we tend to make mistakes).



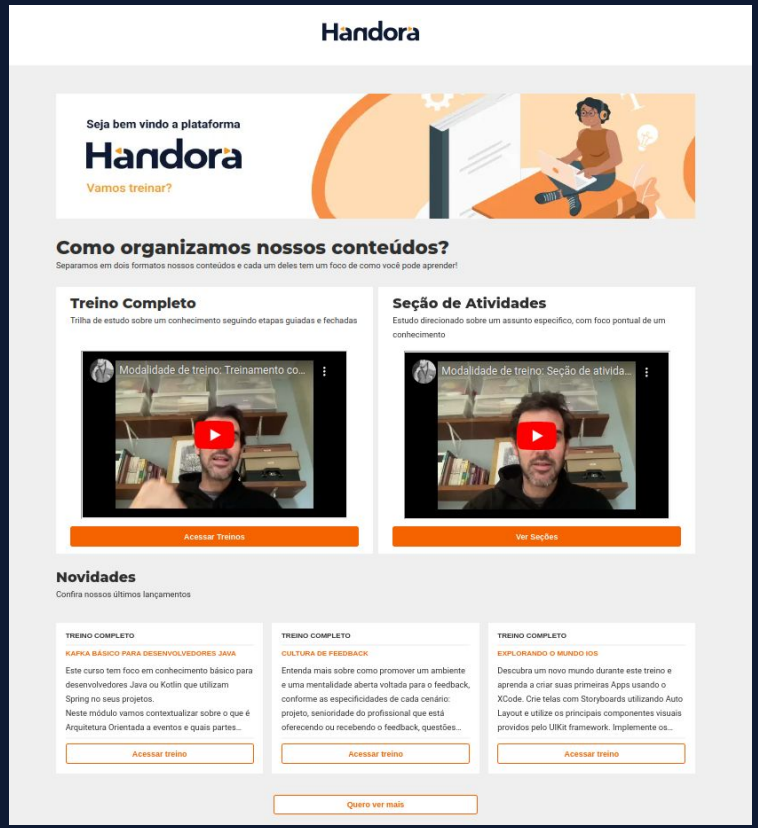
CDD in a nutshell

- CDD provides a **clear limit** indicating how much a code unit could grow
- Every class over the limit **must be** refactored

```
@RestController
@RequestMapping("/certificates")
@ICP(7)
public class CertificateDetailsController {

  @ICP(2)
  private CertificateRepository repo;
  private TrainingCompleted trainingCompleted;

  @ICP(1)
  @GetMapping("/{certificateId}")
  public CertificateResponse execute(
                    Long id, Student student) {
    @ICP(1)
    var potentialCertificate = repo.findById(id);

    var certificate = potentialCertificate.get();

    @ICP(1)
    if (certificate.doesNotBelongTo(student)) {
      throw new ResponseStatusException(NOTFOUND);
    }
    @ICP(1)
    var training = certificate.getTraining();
    @ICP(1)
    return trainingCompleted.check(
        training, student,
        () -> new CertificateResponse(certificate));
  }
}
```

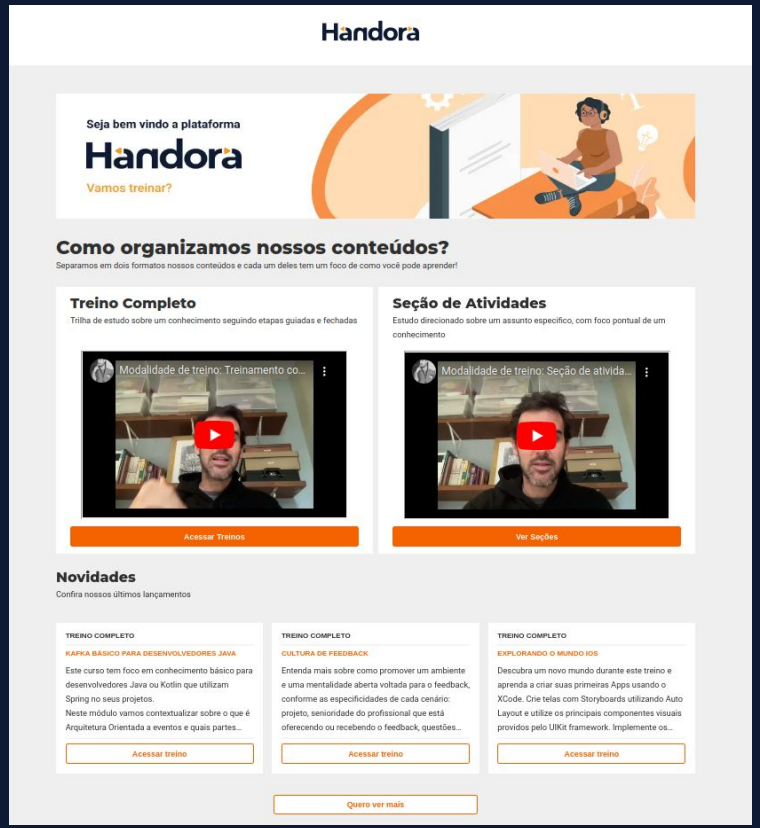
handora.zup.com.br

- Online training platform by Zup
- Composed by 5 services:
  - Core service (Java)
  - Search service (Java)
  - Menu service (Java)
  - Frontend Service (TypeScript)
  - ML service (Python)



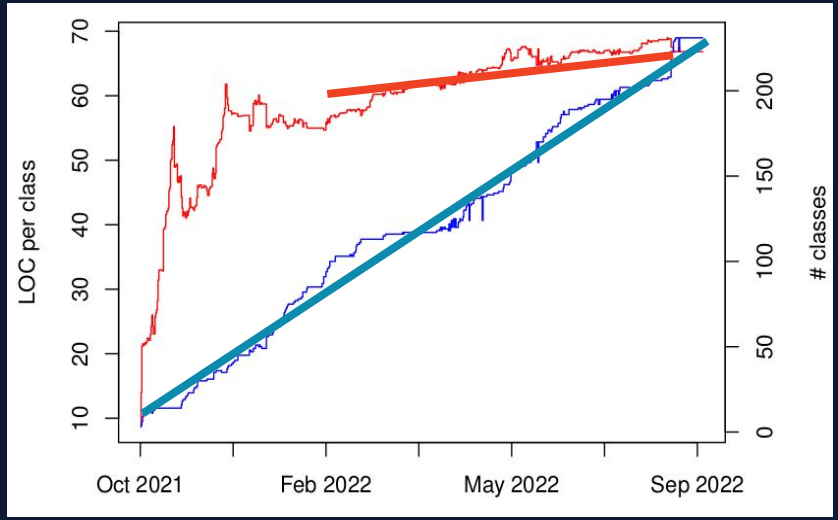
handora.zup.com.br

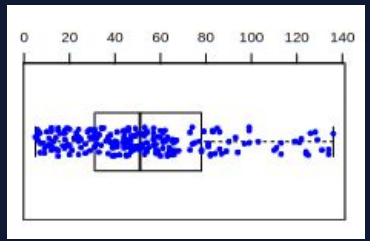
- Online training platform by Zup
- Composed by 5 services:
  - Core service (Java)
  - Search service (Java)
  - Menu service (Java)
  - Frontend Service (TypeScript)
  - ML service (Python)



CDD impacts the **size of the classes**

- CDD seems to help keep the classes small, even as the product evolves (almost linearly)
- Developers concur that the size of the classes are probably due to CDD





ZUP

CDD impacts the **size of the methods**

- Maintenance effort is positively correlated with method length.
- "*Developers should strive to keep their methods within 24 SLOC*"

An Empirical Study on Maintainable Method Size in Java

Shaiful Alam Chowdhury
University of British Columbia
Vancouver, BC, Canada
shaifulc@cs.ubc.ca

Gias Uddin
University of Calgary
Calgary, AB, Canada
gias.uddin@ucalgary.ca

Reid Holmes
University of British Columbia
Vancouver, BC, Canada
rtholmes@cs.ubc.ca

Abstract

Code metrics have been widely used to estimate software maintenance effort. Metrics have generally been used to guide developer effort to reduce or avoid future maintenance burdens. Size is the simplest and most widely deployed metric. The size metric is pervasive because size correlates with many other common metrics (e.g., McCabe complexity, readability, etc.). Given the ease of computing a method's size, and the ubiquity of these metrics in industrial settings, it is surprising that no systematic study has been performed to provide developers with meaningful method size guidelines with respect to future maintenance effort. In this paper we examine the evolution of ~785K Java methods and show that developers should strive to keep their Java methods under 24 lines in length. Additionally, we show that decomposing larger methods to smaller methods also decreases overall maintenance efforts. Taken together, these findings provide empirical guidelines to help developers design their systems in a way that can reduce future maintenance.

CCS Concepts

• **Software engineering → Code metrics.**

Keywords

SLOC, code metrics, maintenance, McCabe, Readability

1 Introduction

Software maintenance has long been identified as challenge for software engineers [43] and maintenance costs often exceed initial development costs [13]. Consequently, both researchers and practitioners are keen to model future maintenance effort from the current state of a software project—to facilitate risk planning and to assist project optimization [28, 45, 53, 76, 79, 85, 95]. These maintenance models rely on various metrics to provide the data necessary to predict future maintenance challenges.

Two of the most widely adopted maintenance indicators are change-proneness [26, 69] and bug-proneness [38, 54, 62, 85, 96]. These metrics suggest that code components (e.g., methods, classes, or modules) that are bug- or change-prone are generally expensive to maintain. Early identification and optimization of such components could thus help reduce future maintenance effort. Unfortunately, change- and bug-proneness of code components are not generally available until after one or more system releases [78].

MSR '22, May 23–24, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9303-4/22/05…$15.00
https://doi.org/10.1145/3524842.3527975

To address this shortcoming, researchers have investigated static code metrics to learn how these metrics alias with change- and bug-proneness (e.g., [26, 43, 62, 93]). These metrics include McCabe [52], McClure [53], nested block depth [42], CBK [18], and size, just to name a few. The utility of these metrics, however, has long been debated (e.g., [25, 26, 74]). While some research has shown that these static metrics correlate with bug- and change-proneness (e.g.,[6, 10, 42, 47, 78]), other research has criticized these findings (e.g., [26, 71, 74]). One underlying concern with these metrics is how they alias with the size of the component being measured. It has been shown that most code metrics are highly influenced by their correlation with size [23, 26, 77]. This suggests that component size, which is easy for software engineers to both measure and reason about, could be the primary maintenance indicator [26].

Ultimately, developers want to be able to design their systems in a way that reduces future maintenance effort. Unfortunately, there is no evidence-based guidelines that developers can follow to optimize the size of their components. In this paper we focus on method level granularity to measure size (e.g., in contrast to class-, or module-level granularities), because method level granularity is desired by both developers and researchers [28, 62]. There is not currently broad consensus for how long a method should be; for example, consider *'What is the ideal length of a method for you?'* [80]. The suggestions vary greatly: between 5-15 lines, 50 lines, and 100-200 lines. We propose that having evidence-based guidance to help developers understand the relationship between method size and software maintenance would help them better structure their systems to reduce future maintenance effort. Therefore, this paper describes an empirical study to investigate optimal method sizes with respect to software maintenance.

To do this, we examined the evolution of ~785K Java methods from 49 open-source software projects to investigate the relationship between method length and change- and bug-proneness. Based on our study, we make the following four primary contributions to the field of software maintenance:

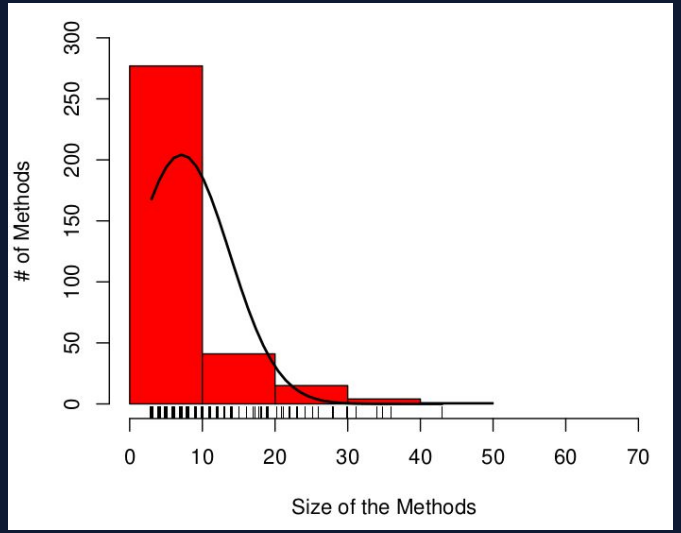
Maintenance effort correlates with method length (RQ1). Previous work has shown that both file-length [26] and module-length [27] negatively impact maintenance; we extend these findings to method-length. We also investigate the best method length measure to use for maintenance prediction.

Methods should be 24 SLOC or fewer (RQ2). Suggesting shorter methods are better is insufficient guidance for practical application as there are challenges to having arbitrarily small methods. We show that methods that are 24 or fewer SLOC are less maintenance-prone than longer methods. We also show that in practice this threshold is widely achievable.

Decomposed large methods undergo less maintenance (RQ3). Complex problems often require complex solutions, which may exceed the 24 SLOC threshold. A natural question is whether these

CDD impacts the **size of the methods**

- 92% of the handora's methods are under 24 SLOC (6.8, on average)
- "*Every unit of code is impacted, because we know what the limit is and what goes into that limit.*"



CDD impacts **testing code**

#	SLOC	Methods	Coverage
S1	7.6k	215	71%
S2	1.3k	41	61%
S3	5.2k	128	64%

- On average, a testing method has ~8 SLOC
- *"I think there is a relationship because the complexity of the test can be seen as a proxy of the complexity of the code under test".*
- There are ~1.3 assertions per method (no method without assertion)

Cognitive Driven Development

Findings

- CDD seems to help designing small classes
- CDD seems to help designing small methods
- CDD seems to help designing small testing methods

Cognitive Driven Development

Findings

- CDD seems to help designing small classes
- CDD seems to help designing small methods
- CDD seems to help designing small testing methods

Challenges

- Still requires manual effort
- We need better tools
- How to ease CDD adoption?