Code Duplication:
A Measurable Technical Debt?

Jens Krinke

Centre for Research on Evolution, Search & Testing
Software Systems Engineering Group
Department of Computer Science
University College London

Introduction

Motivation

- It is possible to build a program that works (i.e. meets its requirements) but also has serious flaws such as:
  - It is not extensible
  - It has low maintainability
    - Lacks separation of concerns
    - Uses anti-patterns
    - Has code duplication
- Some become long-lived programs

Eli Brown – CC

ABB

How to calculate the technical debt?

2013

4

We try to use Sonar to manage the software quality, like this page, we can get the technical debt.
http://www.sonarqube.org/sqale-the-ultimate-quality-model-to-assess-technical-debt/

My question is, how to define the debt to fix a violation remove some duplicated code or a new test case. Is there any calculate algorithm?

sonarqube

share improve this question

edited Jan 2 at 1:06
Tiny Giant
10.4k ●6 ●21 ●47

asked Jul 18 '13 at 15:09
Dennys
314 ●1 ●7 ●23

add a comment

1 Answer

active oldest votes

3

Well, you get all the required information in the blog post: the SQALE methodology is perfectly explained on the official SQALE website.

The blog post even mentions that there's a Sonar plugin that implements the methodology: the Sonar SQALE plugin.

share improve this answer

answered Jul 18 '13 at 15:29
Fabrice - SonarSource Team
16k ●2 ●31 ●38

http://stackoverflow.com/questions/17727289/how-to-calculate-the-technical-debt

FOCUS: **TECHNICAL DEBT**

IEEE Software 2012, vol 29, no 6.

Managing Technical Debt with the SQALE Method

Jean-Louis Letouzey and Michel Ilkiewicz, Inspearit

// *The SQALE (software quality assessment based on life-cycle expectations) method provides guidance for managing the technical debt present in an application's source code.* //

2012

Characteristic:
Testability

Requirement:
There are no cloned parts of 100 tokens or more

Remediation microcycle:
Refactor with IDE and write tests

Remediation function:
20 minutes per occurrence

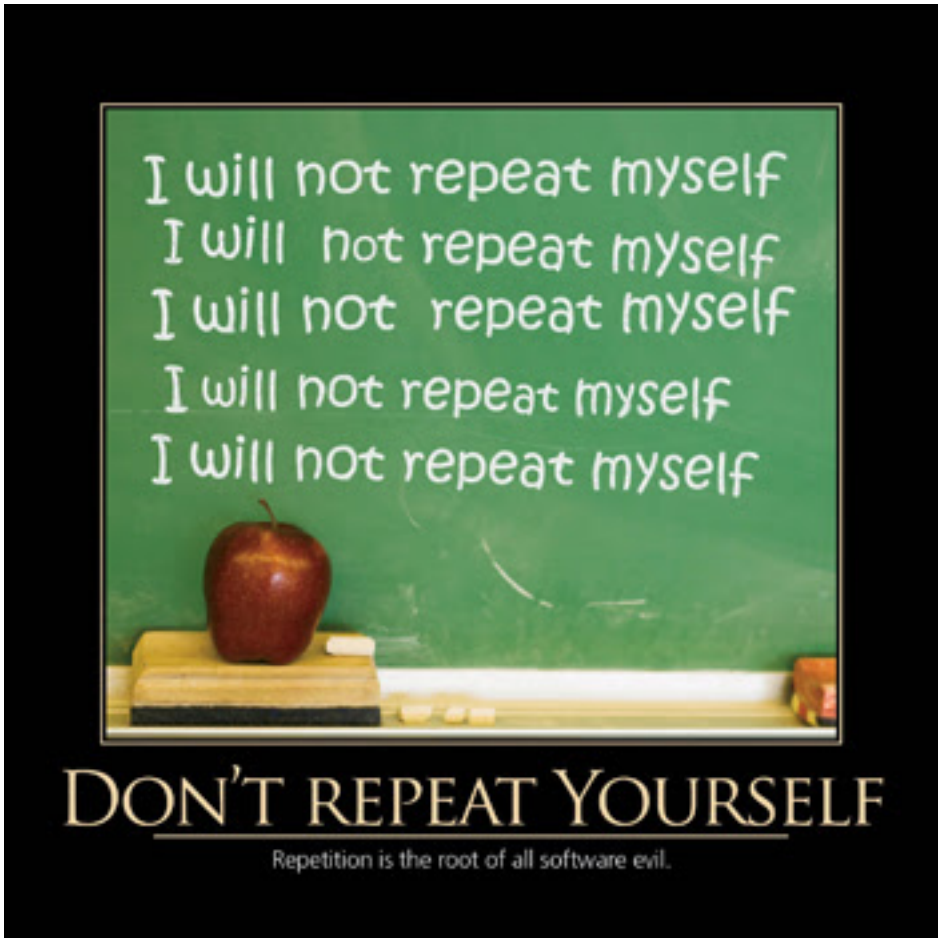
The «SQALE» Analysis Model

2010

Another testability indicator is the presence of duplicated code. Indeed, every piece ~~of duplicated~~ code increases the unit test effort.

Letouzey JL, Coq T. The SQALE analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code.
International Conference on Advances in System Testing and Validation Lifecycle 2010

Don't Repeat Yourself (DRY) Principle



The Don't Repeat Yourself (DRY) principle states that duplication in logic should be eliminated via abstraction; duplication in process should be eliminated via automation.

http://marchoeijmans.blogspot.co.uk/2012/06/dont-repeat-yourself-dry-principle.html

There Is No Agility Without Maintainability

SPM-3—Copy/Paste Programming

Defect Indicators: A largely similar or identical section of code appears in two or more places in the program or set of programs.

Risks: Having to modify identical code in multiple places:

- Increases the likelihood of making slightly different modifications under the mistaken assumption that you have made identical ones,
- Increases the time needed to make changes,
- Increases maintenance costs in direct proportion to the number of times the code has been copied/pasted.

Assessment:

- If you see a block of code that looks familiar, backtrack through the already assessed code looking for similar blocks. If the blocks could be implemented as a (parameterised) subroutine, mark the second and subsequent blocks.

Remedy: Isolate a single copy of the code into a separate program (e.g., method, function, subroutine) and reuse it by calling it from the places in which it was used.

References:

- Why You Should Use Routines, Routinely (Steve McConnell), 1998.

Research Findings:

- Don't Repeat Yourself (DRY):
  The DRY principle: Don't Repeat Yourself,
- Repetitive code indicates poor design:
  Copy and Paste is a design error.

More Defect Categories:

- Work In Progress
- Structured Programming
- Single Point of Maintenance
- Defensive Programming
- Causes for Concern

More Defect Indicators for "Single Point of Maintenance":

- SPM-1 Magic Numbers
- SPM-2 Magic Strings
- SPM-3 Copy/Paste Programming

Famous Quote

1998

David Parnas says that if you use copy and paste while you're coding, you're probably committing a design error.

S. McConnell. 1998. Why you should use routines...routinely. IEEE Softw. 15, 4

I don't remember the occasion but it sounds like something I would say and I think it is true. It is attributed to me on the internet on those note sites so it must be true. You can always trust the internet. 😀😁😂

Dave Parnas

Code Smells as Hints

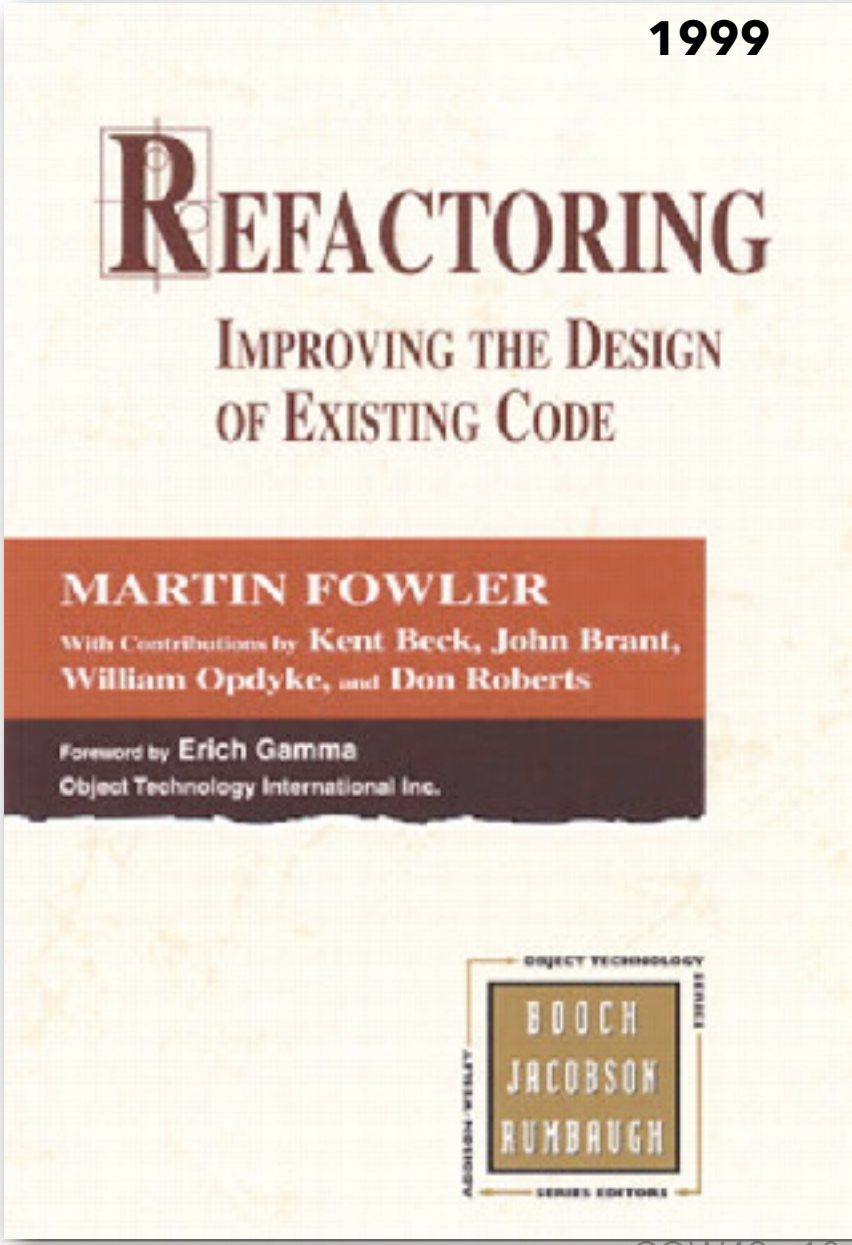
"A code smell is a surface indication of a deeper design problem, observed in a program's low-level structure" [Fowler 1999]

- Long Methods, God Classes, ...

- Design problem is a structure that indicates violations of key design principles or rules
  - Fat Interface, Component Overload, Scattered Concern

Duplicated Code

"Number one in the stink parade is duplicated code. If you see the same code structure in more than one place, you can be sure that your program will be better if you find a way to unify them."



Empir Software Eng
DOI 10.1007/s10664-008-9076-6

2006

“Cloning considered harmful” considered harmful: patterns of cloning in software

Cory J. Kapser · Michael W. Godfrey

We found that as many as 71% of the clones could be considered to have a positive impact on the maintainability of the software system.

Duplicated Code

- increases code size
- complicates software maintenance
  - ✘ errors may have been duplicated, too
  - ✘ changes have to be applied to all clones

✘ ...

Duplicated Code as Technical Debt Examples

2011

For example, if, on average, 3-star and 4-star system snapshots have 10% and 3% duplicated code respectively, then it is inferred that the amount of code that needs to be changed/refactored to improve the level of quality from 3-star to 4-star would be 7% of the total LOC.

Nugroho A, Visser J, Kuipers T. An empirical model of technical debt and interest. International Workshop on Managing Technical Debt 2011

Duplicated Code as Technical Debt Examples

2012

the cost of writing the system from the ground up [5]. The cause of this unfortunate situation is usually less visible: The internal quality of the system design is declining [6], and duplicated code, overly complex methods, noncohesive classes, and long parameter lists are just a few signs of this decline [7]. These issues, and many others, are usually

Marinescu R. Assessing technical debt by identifying design flaws in software systems. IBM Journal of Research and Development. 2012 56(5).

Duplicated Code as Technical Debt Examples

2014

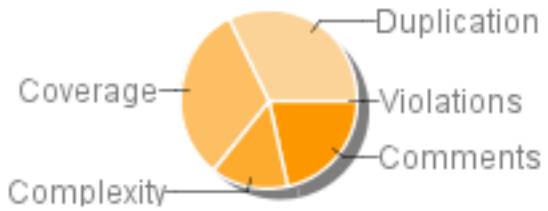
$$Debt = duplication + violations + comments + coverage + complexity + design \quad (1)$$

$$duplication = cost\_to\_fix\_one\_block * duplicated\_blocks \quad (2)$$

Griffith I, Reimanis D, Izurieta C, Codabux Z, Deo A, Williams B.
The correspondence between software quality models and technical debt estimation approaches.
International Workshop on Managing Technical Debt 2014

2009

Technical Debt
$ 14'605
29 man days



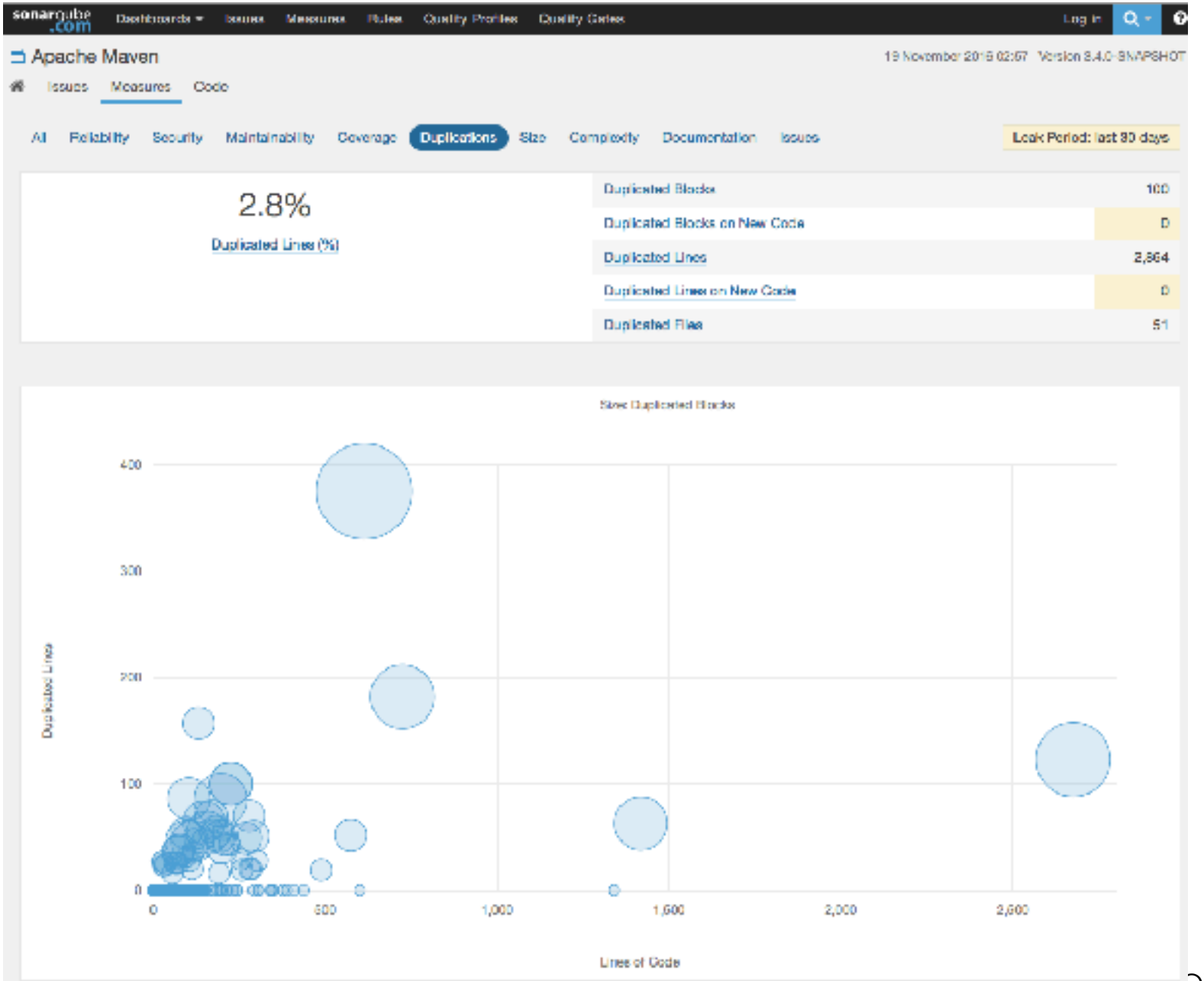
The current version of the plugin is 0.2 and uses the following formula to calculate the debt :

Debt(in man days) = cost_to_fix_duplications + cost_to_fix_violations +
cost_to_comment_public_API + cost_to_fix_uncovered_complexity +
cost_to_bring_complexity_below_threshold

Where :

Duplications = cost_to_fix_one_block * duplicated_blocks

O. Gaudin, "Evaluate your technical debt with Sonar," Sonar, 2009.
http://www.sonarqube.org/evaluate-your-technical-debt-with-sonar/


sonarqube .com
Dashboards
Issues
Measures
Rules
Quality Profiles
Quality Gates
Log in
Apache Maven
19 November 2019 02:07 Version 3.4.0-SNAPSHOT
Issues
Measures
Code
All
Reliability
Security
Maintainability
Coverage
Duplications
Size
Complexity
Documentation
Issues
Leak Period: last 30 days
2.8%
Duplicated Lines (%)
Duplicated Blocks 100
Duplicated Blocks on New Code 0
Duplicated Lines 2,864
Duplicated Lines on New Code 0
Duplicated Files 51
Size: Duplicated Blocks
400
300
200
100
0
Duplicated Lines
0
500
1,000
1,500
2,000
2,500
Lines of Code

Duplicated Code
aka "Cloned Code"

Clones are segments of code that are similar according to some definition of similarity.

Ira Baxter, 2002

Duplicated Code Detectors

JCCD

KClone

CCFinder

ACD

Dup

SDD

SimScan

CloneDr

Scorpio

ConQAT

CPC

PMD

JCD

Bauhaus

NICAD

Duplix

Duplo

Simian

Clone Tracker

CLICS

Shinobi

CPD

Clone Digger

Clone Board

30 Similarity Analysers

Clone detectors

CCFinderX
iClones
Simian, NiCad
Deckard

Plagiarism detectors

JPlag
Plaggie, Sherlock
Sim

7zncd, bzip2ncd
gzipncd, xz-ncd
icd, ncd

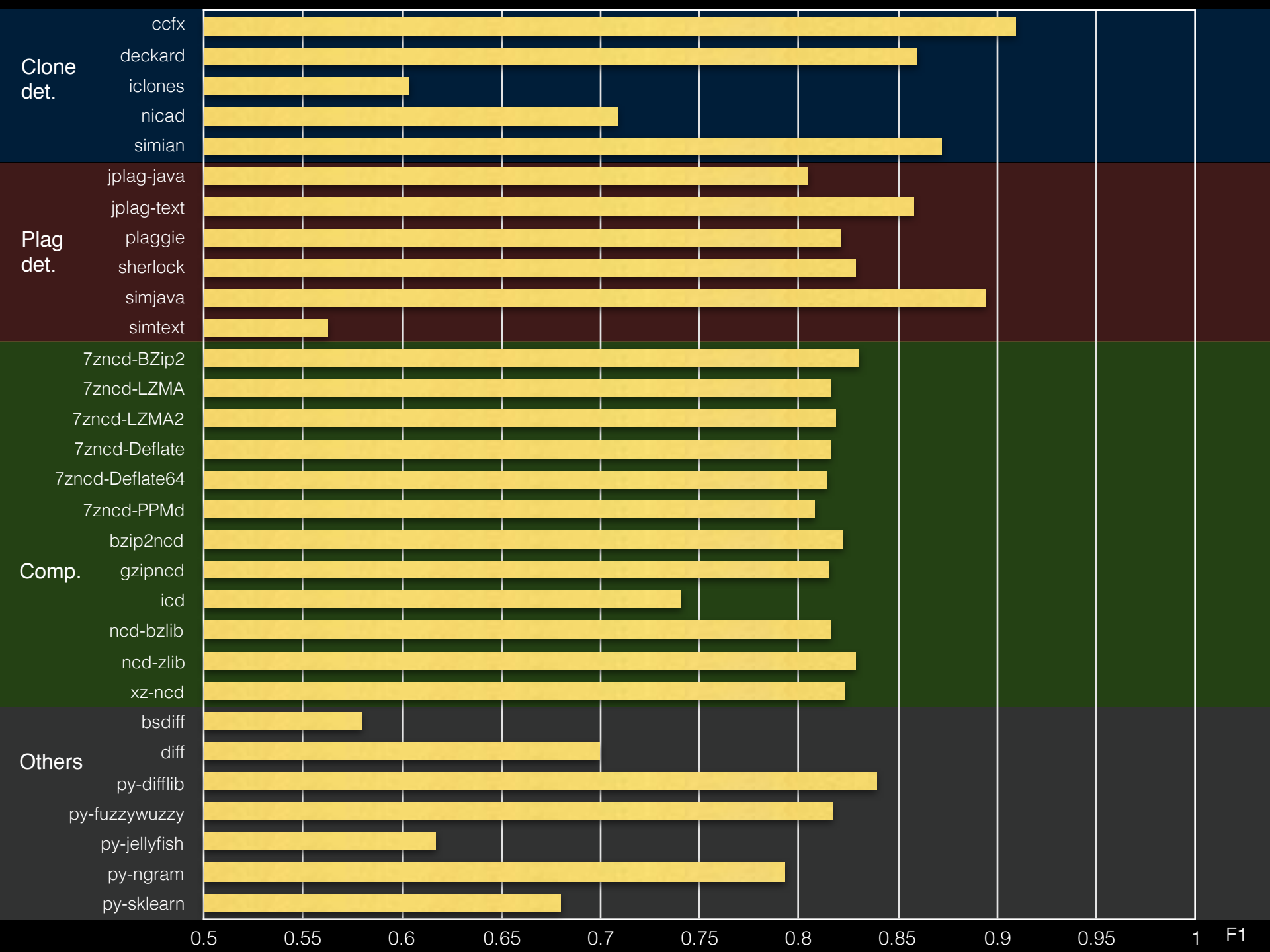
Compression

diff, bsdiff
difflib, fuzzywuzzy
jellyfish, ngram, sklearn

Others

Ragkhitwetsagul et al: Similarity of Source Code in the Presence of Pervasive Modifications. International Working Conference on Source Code Analysis and Manipulation 2016.

Parameter Settings

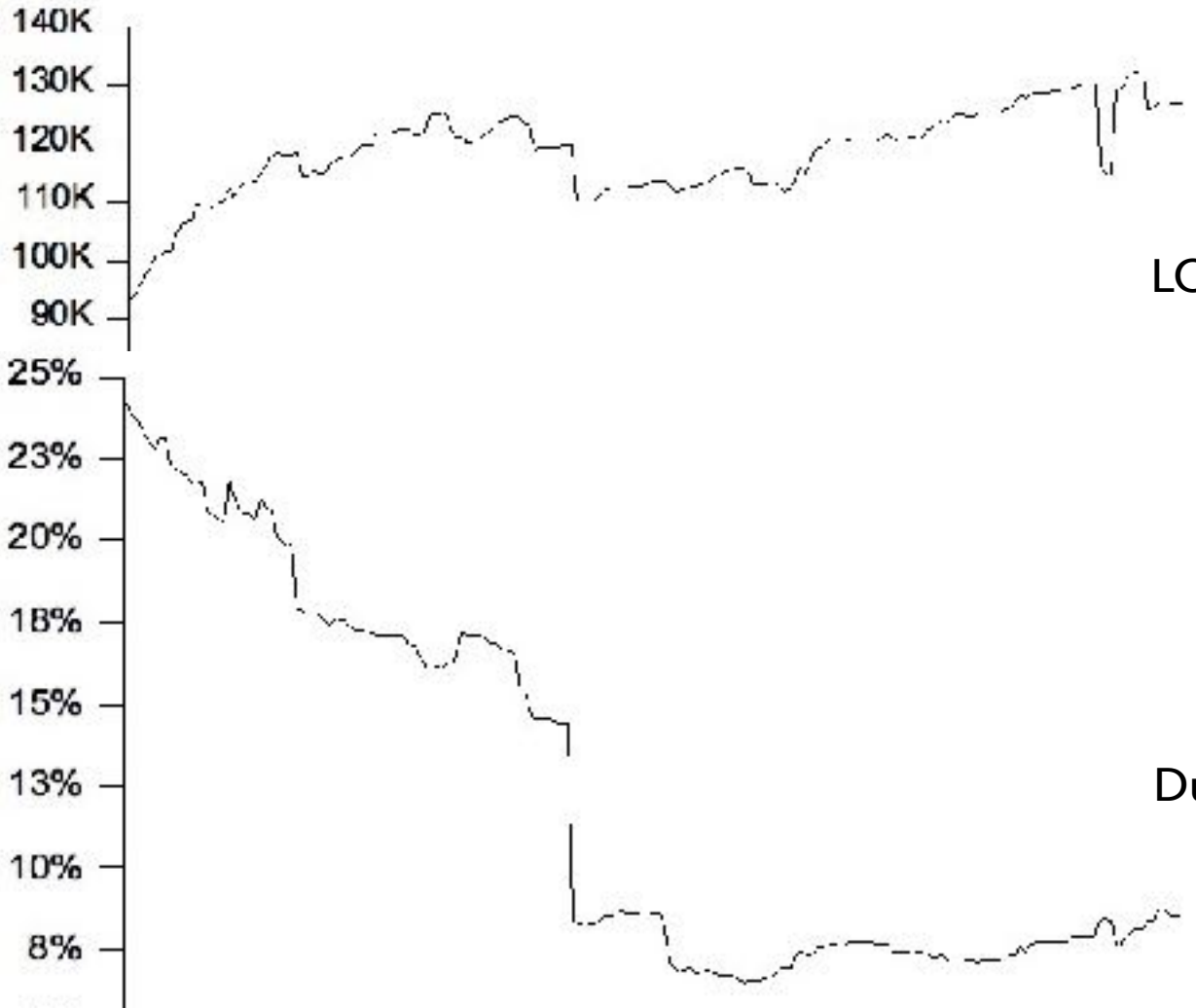

Clone det.
ccfx
deckard
iclones
nicad
simian
Plag det.
jplag-java
jplag-text
plaggie
sherlock
simjava
simtext
Comp.
7zncd-BZip2
7zncd-LZMA
7zncd-LZMA2
7zncd-Deflate
7zncd-Deflate64
7zncd-PPMd
bzip2ncd
gzipncd
icd
ncd-bzlib
ncd-zlib
xz-ncd
Others
bsdiff
diff
py-difflib
py-fuzzywuzzy
py-jellyfish
py-ngram
py-sklearn
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
F1

Citation from an typical clone related paper

"the existence of clones is known to worsen productivity in software maintenance"

Is it really?

How to measure duplication?



Is Clone Code more stable?

- If cloned code is changed often, it requires more attention and is more expensive
- If cloned code is more stable,
  its maintenance costs will be lower

Cloned Code is (not) changed more often

- Lozano et al 2007 and 2010...
- Krinke 2008: Is cloned code more stable than non-cloned code?
- Hotta et al 2010: Is duplicate code more frequently modified than non-duplicate code in software evolution?
- Göde and Harder 2011: Clone Stability
- Harder and Göde 2012: Cloned Code: Stable Code

Cloned code may be more stable than Non-Cloned Code

~~Is Clone Code more stable?~~

Is Clone Code older?

- Cloned code is usually older than non-cloned code.
- Cloned code in a file is usually older than the non-cloned code in the same file.

Cloned code is older, i.e., more stable than Non-Cloned Code

J.Krinke: Is Cloned Code older than Non-Cloned Code?
International Workshop on Detection of Software Clones 2011.

Pitfalls

The different studies agree in general,
so can we make a general observation?

- If so, they must agree for each system!
- Experiment:
  Compare two approaches on 12 systems

Experiment

	Agreement
DNSJava	✘
Ant-Contrib	✘
Carol	✘
Plandora	✘
Ctags	✘
TidyForNet	✘
QMail Admin	✔
Hash Kill	✔
GreenShot	✘
ImgSeqScan	✘
CapitalResourc	✔
MonoOSC	✔

- Major Disagreement
- AA only considers the time after the last change
- MF considers the complete history
- AA and MF do not correlate!
- **Be aware of what you measure!**

Are Clones changed consistently?

- If cloned code is changed consistently, it evolves together
- If cloned code is changed consistently, inconsistent changes may be bugs

Krinke J: A Study of Consistent and Inconsistent Changes to Code Clones. Working Conference on Reverse Engineering 2007

Are Clones changed consistently?

- Clones are inconsistently changed (~50%)
- if a clone group is inconsistently changed, there is an increasing probability that it is consistently changed later (Late Propagation)

Krinke J: A Study of Consistent and Inconsistent Changes to Code Clones. Working Conference on Reverse Engineering 2007

Do Clones Lower the Quality?

There is no significant difference between the quality of cloned and non cloned methods for most of 27 metrics.

- 4,421 Java projects, 1,4mio methods)
- Code Complexity, Modularity, Documentation
- Exceptions
  - HEFF (Halstead effort)
  - NEXP (number of expressions)
  - XMET (number of external methods called)

Saini V, Sajnani H, Lopes C: Comparing Quality Metrics for Cloned and non cloned Java Methods International Conference on Software Maintenance and Evolution 2016.

Do inconsistent changes cause problems?

- Bakota et al. 2007: Detected six defects based on their analysis of 60 change anomalies in the evolution of clones.
- Selim et al. 2010: cloned code is not always more risky than non-cloned code;
  the risk seems to be system dependent.
- Göde, Koschke 2011: The number of unintentional inconsistent changes is small.

Bakota T, Ferenc R, Gyimothy T: Clone Smells in Software Evolution. International Conference on Software Maintenance, 2007.
Selim G, Barbour L, Shang W, Adams B, Hassan A, Zou Y: Studying the Impact of Clones on Software Defects. Working Conference on Reverse Engineering, 2010.
Göde N, Koschke R: Frequency and Risks of Changes to Clones. International Conference on Software Engineering, 2011

Do Clones cause Bugs?

- most bugs have very little to do with clones
- cloned code contains less buggy code
- larger clone groups don't have more bugs than smaller clone groups
- making more copies of code doesn't introduce more defects

Rahman F, Bird C, Devanbu P: Clones: What is that smell? Working Conference on Mining Software Repositories, 2010.

Software Redundancy

- **Intentional Redundancy**

  N-version programming

- **Unintentional Redundancy**

  Duplication of code or functionality

- **Implicit Redundancy**

  Redundancy in Programming Languages

Redundancy good or bad?

Mny blv tht rdndncy s bd
nd tht rmvng rdndncy
mprvs prdctvty nd mntnblty.

Redundancy
good or bad?

Many believe that redundancy is bad
and that removing redundancy
improves productivity and maintainability.

But redundancy can improve reliability,
productivity, maintenance and comprehension.

Copy and paste doesn't create bad code.
Bad programmers create bad code.

Jeff Atwood, 2009

Where :

Duplications =