Standardized code quality benchmarking for improving software maintainability

Robert Baggen, Jose Pedro Correia, Katrin Schill, Joost Visser

Software Quality Journal, 2011

Abstract

- We provide an overview of the approach developed by the Software Improvement Group for code analysis and quality consulting focused on software maintainability

- We use a model based on the ISO 9126 definition of maintainability and source code metrics

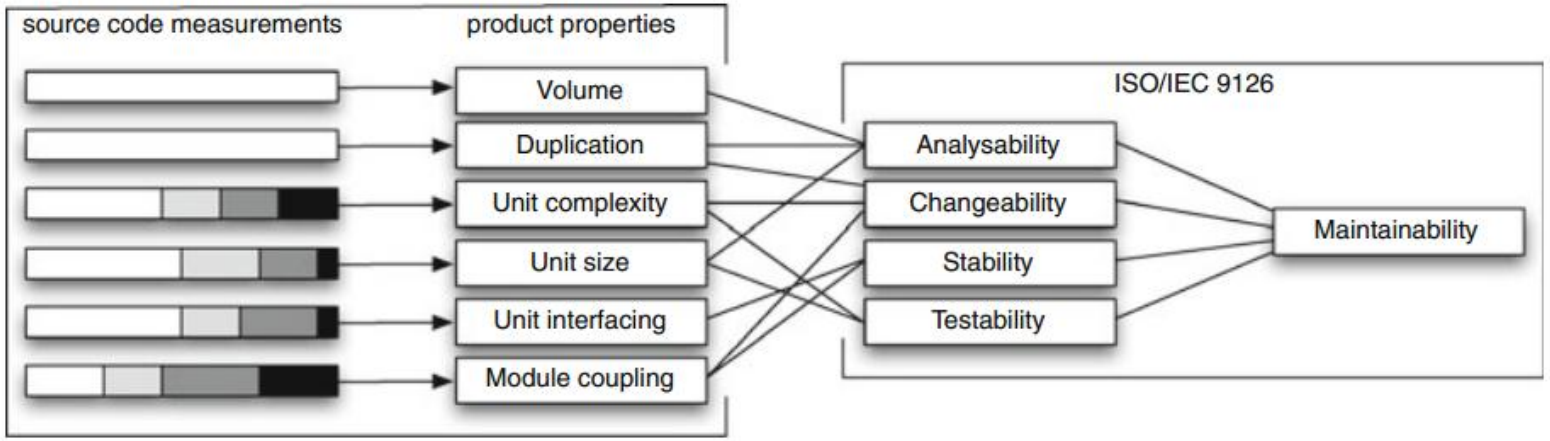
- Individual assessments are stored in a repository that allows any system to be compared to the industry wide state of the art in code quality and maintainability.

- When a minimum level of software maintainability is reached a Maintainability Certificate is issued for the software product

Measuring software

- The application of metrics for the measurement and improvement of code quality has a tradition of more than 40 years.
- However, metrics have to be chosen with clear reference to an agreed standard—e.g. the ISO/IEC 9126
- In the ISO/IEC 9126, maintainability is seen as a general quality characteristic of a software product
- And is decomposed into the following subcharacteristics:
  - Analyzability
  - Changeability
  - Stability
  - Testability

SIG.EU Quality Model



Key metrics for maintainability

- **Volume:** the larger a system, the more effort it takes to maintain since there is more information to be taken into account;
- **Redundancy:** duplicated code has to be maintained in all places where it occurs;
- **Unit size:** units as the lowest-level piece of functionality should be kept small to be focused and easier to understand;
- **Complexity:** simple systems are easier to comprehend and test;
- **Unit:** interface size units with many parameters can be a symptom of bad encapsulation;
- **Coupling:** tightly coupled components are resistant to change

Measurements aggregation

- In the model, the subcharacteristics are made quantifiable with the above source code metrics.
- For this, raw metrics have to be aggregated to the level of the whole system.
- This is done either by using a grand total (such as for Volume and Duplication) or by using so-called quality profiles.
- Quality profiles summarize the distribution of a metric at a certain level (e.g. per unit) by performing a classification into risk categories based on a set of thresholds.
  - The outcome is percentage of code in low, moderate, high, and very high risk.

Measurements aggregation

- The aggregated measurements are used to determine a rating for each source code property, based on the application of another set of thresholds.
- These are further combined to calculate ratings for the subcharacteristics
- Finally, the general maintainability score for a given system is calculated
- The ratings correspond to 5 quality levels, represented by a number of stars, from one star to five stars

Example: Assessing Duplication

Rating	Duplication
★★★★★	3%
★★★★	5%
★★★	10%
★★	20%
★	–

- A line of code is considered redundant if it is part of a code fragment (larger than 6 lines) that is repeated literally (modulo whitespace) in at least one other location in the source code.
- The percentage of redundant lines of code is used to evaluate the duplication property

Example: Assessing Complexity

Cyclomatic complexity	Risk category
1–10	Low risk
11–20	Moderate risk
21–50	High risk
>50	Very high risk

Rating	Maximum relative volume		
	Moderate	High	Very high
★★★★★	25%	0%	0%
★★★★	30%	5%	0%
★★★	40%	10%	0%
★★	50%	15%	5%
★	–	–	–

Assessing a subcharacteristic

- Subcharacteristic rating is obtained by averaging the ratings of the properties where a “x” is present in the subcharacteristic’s line in the matrix.

Table 3 Mapping of source code properties to ISO/IEC 9126 subcharacteristics

ISO 9126 maintainability	Properties					
	Volume	Duplication	Unit size	Unit complexity	Unit interfacing	Module coupling
Analyzability	×	×	×			
Changeability		×		×		×
Stability					×	×
Testability			×	×		

- For example, changeability is affected by duplication, unit complexity and module coupling, thus its rating will be computed by averaging the ratings obtained for those properties

Maintainability rating

- Finally, all subcharacteristic ratings are averaged to provide the overall maintainability rating

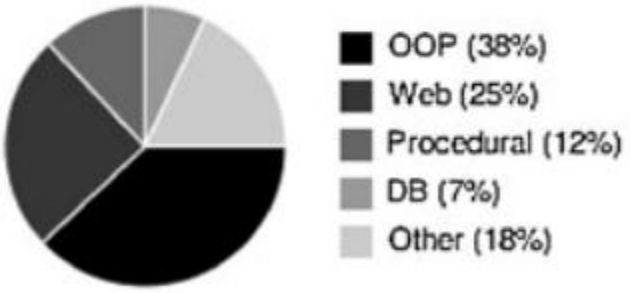
Software Benchmark Repository

- Quality indices calculated from source code remain arbitrary as long as no comparison with other systems is available.
- To provide such information, SIG maintains a benchmark repository holding the results from several hundreds of standard system evaluations carried out so far
- Currently (May 2010), the benchmark repository holds results for over 500 evaluations encompassing around 200 systems.
- These comprise proprietary systems (about 85%) as well as open-source systems
- 45 different computer languages are, including Java, C, COBOL, C#, C++ and ABAP

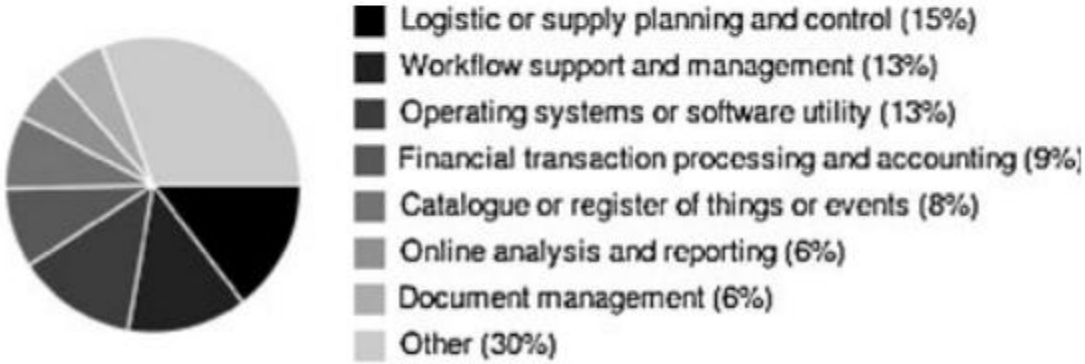
Software Benchmark Repository

Table 1 Average values for redundancy and complexity

Paradigm or group	Redundant lines (%)	Decision density (McCabe / LOC) (%)
OOP (e.g. Java, C#)	12.3	19.6
Web (e.g. JSP,ASP,PHP)	32.5	15.5
Procedural (e.g. C, COBOL)	20.2	10.0
DB (e.g. PL/SQL, T-SQL)	28.2	7.9



(a) Percentage of systems per



(b) Percentage of systems per functionality group.

Calibration

- Calibration is performed on two different levels, namely to determine thresholds for:
  - the raw metrics
  - aggregated quality profiles.

- The first level of calibration is performed by analyzing the statistical distributions of the raw metrics among the different systems.
- Thresholds are then determined based on the variability between the systems, allowing us to pinpoint the more uncommon (thus considered riskier) range of values for the metric

Calibration

- For the second level, the model is calibrated such that systems have a <5; 30; 30; 30; 5> percentage-wise distribution over 5 levels of quality.
- If a system is awarded 5 stars, it is comparable to the 5% best systems in the benchmark, in terms of maintainability.
- It would be possible to select any other distribution, since it is a parameter of the calibration algorithm
- We chose this one in particular so that only very good systems attain 5 stars, hence promoting excellence.

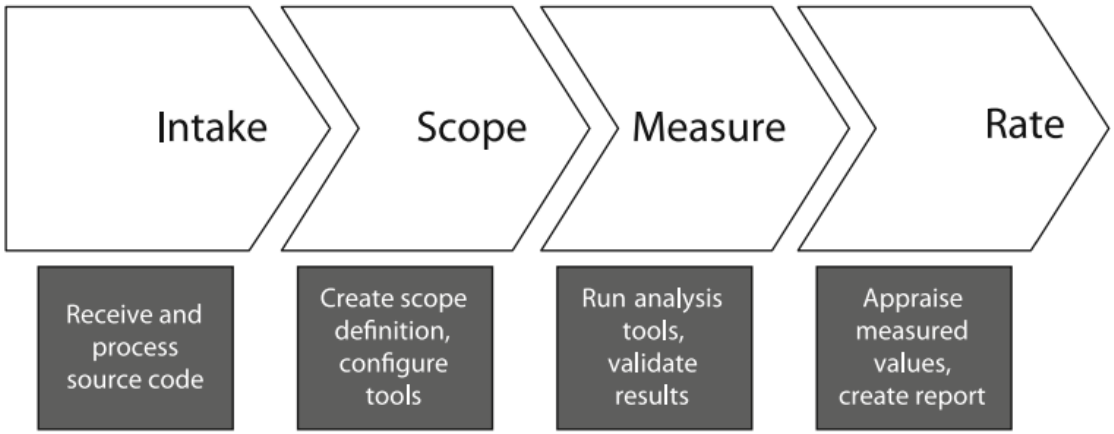
Calibration

- It would be possible to rely on expert opinions to define the thresholds

- But calibration against a large set of real-world systems brings some advantages, namely:
  - the process is more objective since it is based solely on data;
  - it can be done almost automatically
  - it creates a realistic scale, since it is constructed to represent the full range of quality achieved by real-world systems.

Standardized evaluation procedure

- A standard evaluation procedure has been defined in which the SIG quality model is applied to software products



Standardized evaluation procedure

- **Intake:** The source code is received via a secure upload and copied to a standard location.

- **Scope:** In this step, the scope of the evaluation is determined.
  - As a result, an unambiguous description of which software artifacts are to be covered becomes available.
  - This scope description includes information such as:
    - the identification of the software system (name, version, etc.),
    - a characterization of the technology footprint of the system (which programming languages and the number of files analyzed for each),
    - as well as a description of specific files excluded from the scope of the evaluation and why.

Standardized evaluation procedure

- **Measure:** In this step, a range of measurement values is determined for the software artifacts that fall within the evaluation scope.
  - Each measurement is determined automatically by processing the software artifacts with an appropriate algorithm.
  - This results in a large collection of measurement values at the level of source code units, which are then aggregated to the level of properties of the system as a whole.

- **Rate:** Finally, the values obtained in the measurement step are combined and subsequently compared against target values in order to determine quality subratings and the final rating for the system under evaluation.

Certification of maintainability

- Systems with maintainability scores above a certain threshold are eligible for the certificate called ‘TU¨ ViT Trusted Product Maintainability’

- To achieve a certificate, a system must score at least with 2 stars on all subcharacteristics and at least 3 stars on the overall maintainability score.

Case Examples

- Ministry of Justice, NRW, Germany

- The Ministry of Justice in the German state of Nordrhein-Westfalen uses BASIS-Web, a complete prison management system implemented in Java client– server technology.

- Since 2005, BASIS-Web is used for the management of prisoners data, the correct calculation of periods of detention as well as the treatment of cash balances of prisoners.

- The assessment had two goals as follows:
  - To give insight into the maintainability of the system
  - To determine whether any future performance risks can be identified from the source code

Case Examples

- KAS BANK is a European specialist in wholesale security services
- KAS BANK decided to make use of SIG's service to safeguard the maintainability of the new software in the long run.
- The Tri-Party Collateral Management system of KAS BANK achieved a certificate with 4 stars in April 2009.

Potential limitations

- Repository bias:
  - An important type of bias could be that only systems with quality problems require SIG's services, thus resulting in low standards for quality
- Quality model soundness:
  - As any model, it is not complete and provides only an estimation of the modeled variable, in this case maintainability
- Quality model stability:
  - Updating the set of systems would cause the model to change dramatically, thus reducing its reliability as a standard

Potential limitations

- Focus:
  - The evaluation procedure described in this paper has its focus on assessing a software product's maintainability.
  - However, this is just one aspect of a software product's quality, thus it is possible and even desirable to use it in combination with other quality instruments.

Related Work

- Methodologies for software process improvement (SPI), such as the Capability Maturity Model Integration (CMMI), concern the production process, rather than the product.

- SPI works under the assumption that better processes lead to better products.

- Since this relationship is not a perfect one, improving software via an objective assessment of source code quality is an independent approach usable in a complementary way

Deriving Metric Thresholds from Benchmark Data

Tiago L. Alves, Christiaan Ypma, Joost Visser

ICSM 2010

Abstract

- A wide variety of software metrics have been proposed and a broad range of tools is available to measure them.
- However, the effective use of software metrics is hindered by the lack of meaningful thresholds.
- Thresholds have been proposed for a few metrics only, mostly based on expert opinion and a small number of observations
- We designed a method that determines metric thresholds empirically from measurement data.

Abstract

- The measurement data for different software systems are pooled and aggregated after which thresholds are selected that:
  - Bring out the metric's variability between systems
  - Help focus on a reasonable percentage of the source code volume

- We applied our method to a benchmark of 100 object-oriented software systems, both proprietary and open-source, to derive thresholds for metrics included in the SIG maintainability model

Motivation

- In spite of the potential benefits of metrics, their effective use has proven elusive.

- Metrics have been used successfully for quantification, but have generally failed to support subsequent decision-making

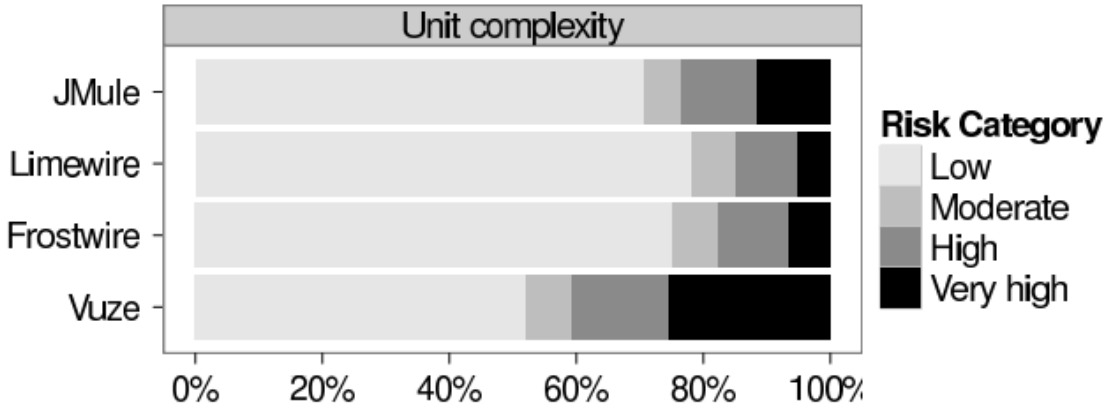
Requirements

- The method should not be driven by expert opinion but by measurement data from a representative set of systems (data-driven);
- The method should respect the statistical properties of the metric and should be resilient against outliers in metric values and system size (robust);
- The method should be repeatable, transparent and straightforward to carry out (pragmatic).

Quality Profiles

TABLE I: Quality profiles: Unit complexity

	risk:	Low	Moderate	High	Very high
JMule	0.4.1	70.52%	6.04%	11.82%	11.62%
LimeWire	4.13.1	78.21%	6.73%	9.98%	5.08%
FrostWire	4.17.2	75.10%	7.30%	11.03%	6.57%
Vuze	4.0.04	51.95%	7.41%	15.32%	25.33%



≤ 6 for low risk
]6, 8] for moderate risk
]8, 15] for high risk
> 15 for very-high risk

Proposed Approach

- Steps:
  1. Metrics extraction
  2. Weight ratio calculation
  3. Entity aggregation
  4. System aggregation
  5. Weight ratio aggregation
  6. Thresholds derivation

1. Metrics extraction

- Metrics are extracted from a benchmark of software systems.
- For each system, and for each entity belonging to this system (e.g. method), we record a metric value and weight metric
- As weight we will consider the source lines of code of the entity.
- As example, for the Vuze system:
  - There is method (entity) called MyTorrentsView.createTabs()
  - With a McCabe metric value of 17
  - Weight value of 119 LOC

2. Weight ratio calculation

- For each entity, we compute the weight percentage within its system, i.e., we divide the entity weight by the sum of all weights of the same system.

- For each system, the sum of all entities must be 100%.

- As example, for the MyTorrentsView.createTabs() method entity:
  - We divide 119 by 329,765 (total LOC for Vuze)
  - Which represents 0.036% of the overall Vuze system.

3. Entity aggregation

- We aggregate the weights of all entities per metric value, which is equivalent to computing a weighted histogram (the sum of all bins must be 100%).
- Hence, for each system we have a histogram describing the distribution of weight per metric value.
- As example, all entities with a McCabe value of 17 represent 1.458% of the overall LOC of the Vuze system

4. System aggregation

- We normalize the weights for the number of systems and then aggregate the weight for all systems.
- Normalization ensures that the sum of all bins remains 100%, and then the aggregation is just a sum of the weight ratio per metric value.
- Hence, we have a histogram describing a weighted metric distribution.
- As example, a McCabe value of 17 corresponds to 0.658% of all code.

5. Weight ratio aggregation

- We order the metric values in ascending way and take the maximal metric value that represents 1%, 2%, ..., 100% of the weight.

- This is equivalent to computing a density function, in which the x-axis represents the weight ratio (0-100%), and the y-axis the metric scale.

- As example, according to the benchmark used for this paper, for 60% of the overall code the maximal McCabe value is 2.

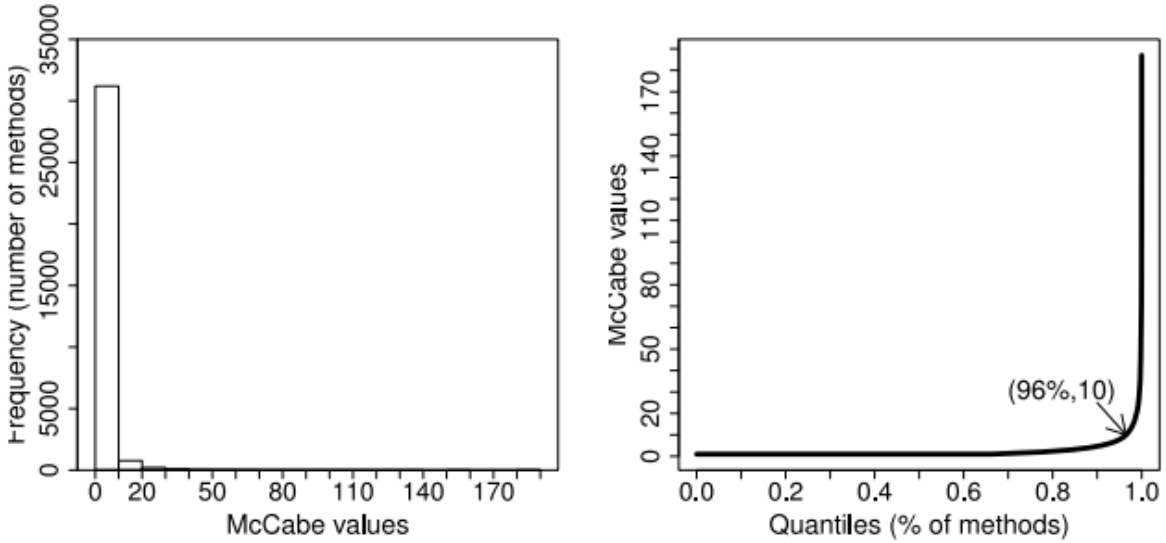
6. Thresholds derivation

- Thresholds are derived by choosing the percentage of the overall code we want to represent.

- For instance, to represent 90% of the overall code for the McCabe metric, the derived threshold is 14.

- These percentiles are used in quality profiles to characterize code according to four categories:
  - Low risk (between 0 − 70%)
  - Moderate risk (70 − 80%)
  - High risk (80 − 90%)
  - Very-high risk (> 90%)

Example: Histogram vs Quantiles



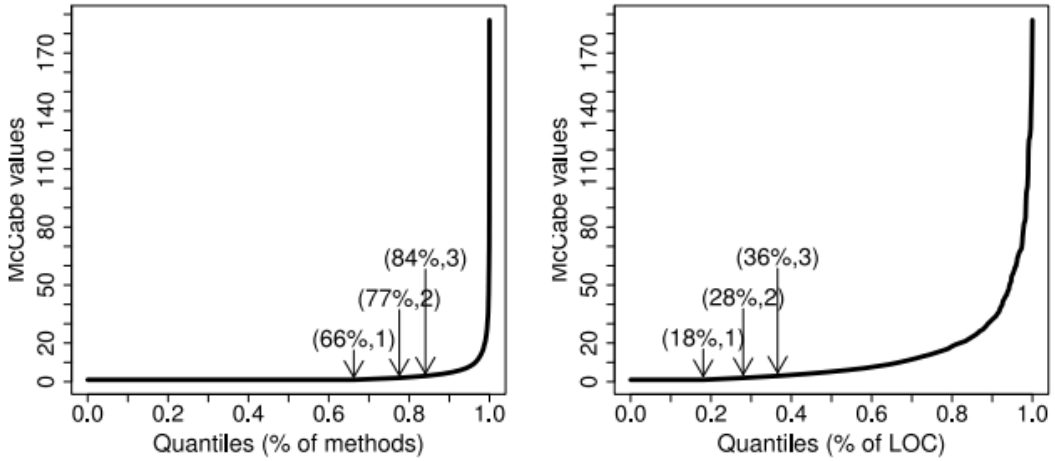
(a) Histogram (b) Quantiles

Fig. 3: McCabe distribution for Vuze system represented with a histogram and a quantile plot.

Figure 3a allows us to observe that more than 30,000 methods have McCabe ≤ 10 (the frequency of the 1st bin is 30.000)

In Figure 3b we can observe that 96% of methods have a McCabe value ≤ 10.

Non-weighted vs Weighted (1/3)

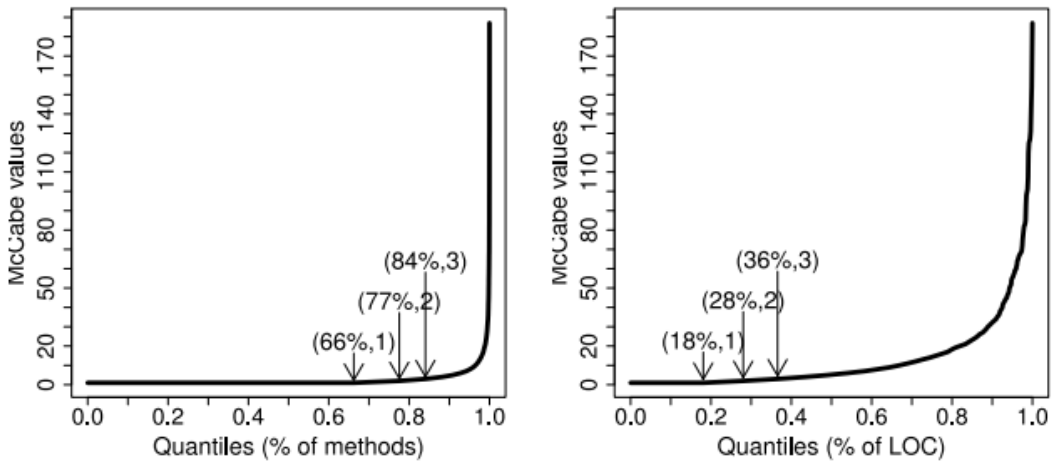


(a) Non-weighted (b) Weighted

Fig. 4: McCabe distribution for the Vuze system (non-weighted and weighted by LOC) annotated with the x and y values for the first three changes of the metric.

Figure 4a shows that the metric variation is concentrated in just a small percentage of the overall methods.
Instead of considering every method equally (every method has a weight of 1), we will use the method's LOC as its weight

Non-weighted vs Weighted (2/3)



(a) Non-weighted (b) Weighted

Fig. 4: McCabe distribution for the Vuze system (non-weighted and weighted by LOC) annotated with the x and y values for the first three changes of the metric.

Comparing Figure 4a to Figure 4b, we can observe that in the weighted distribution the variation of the McCabe values starts much earlier. The first three changes for the McCabe values are at 18%, 28% and 36% quantiles

Non-weighted vs Weighted (3/3)

- In sum, for the Vuze system, both weighted and nonweighted plots show that large McCabe values are concentrated in just a small percentage of code.

- However, while in the non-weighted distribution the variation of McCabe values happen in the tail of the distribution (66% quantile), for the weighted distribution the variation starts much earlier, at the 18% quantile

All systems

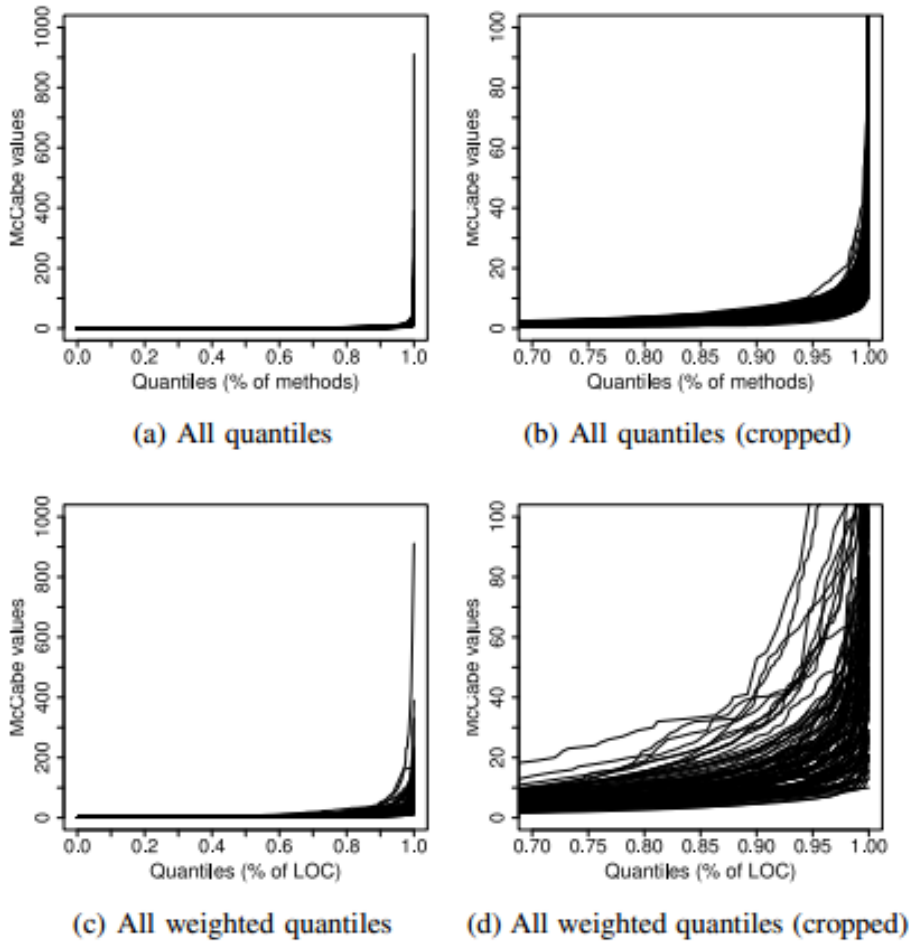


Fig. 5: Non-weighted and weighted McCabe distributions for 100 projects of the benchmark.

When comparing Figure 4 to Figure 5 we observe that, as seen for the Vuze system, weighting by LOC emphasizes the metric variability

Hence, weighting by LOC not only emphasizes the difference between methods in a single system, but also make the differences between systems more evident

Thresholds derivation

- We proposed the use of the 70%, 80% and 90% quantiles to derive thresholds.

- For the McCabe metric, these quantiles yield to thresholds 6, 8 and 14, respectively
  - For low risk, we considered the lines of code for methods with McCabe between 1−6
  - For moderate risk 7−8
  - For high risk 9−14
  - For very-high risk > 14

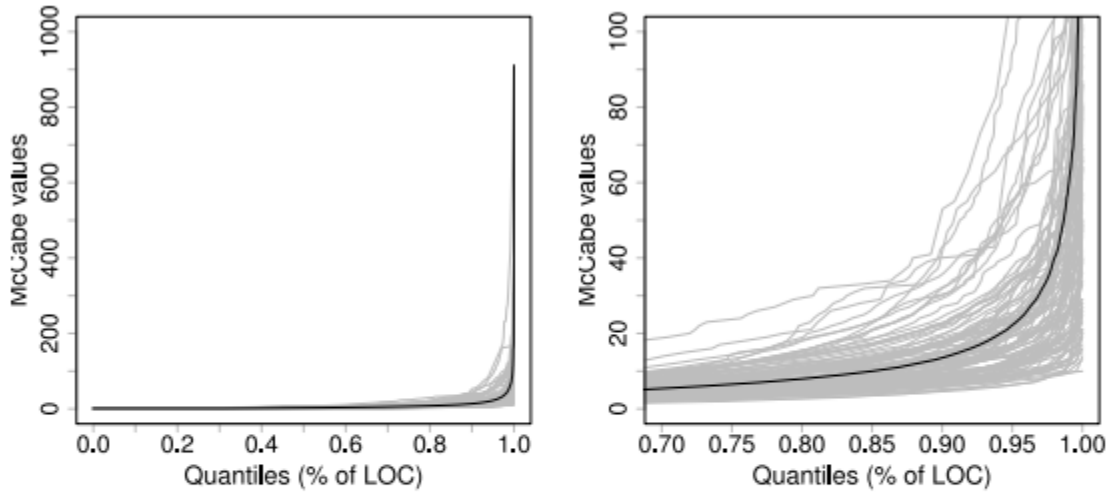
- This means, that for low risk we expect to identify around 70% of the code, and for each of the other risk categories 10%

Using relative size

- Conceptually, to summarize the McCabe metric, we have taken all curves of density functions for all systems and combined them into a single curve.

- Performing weight normalization ensures that every system is represented equally in the benchmark, limiting the influence of bigger systems over small systems in the overall result.

Using relative size

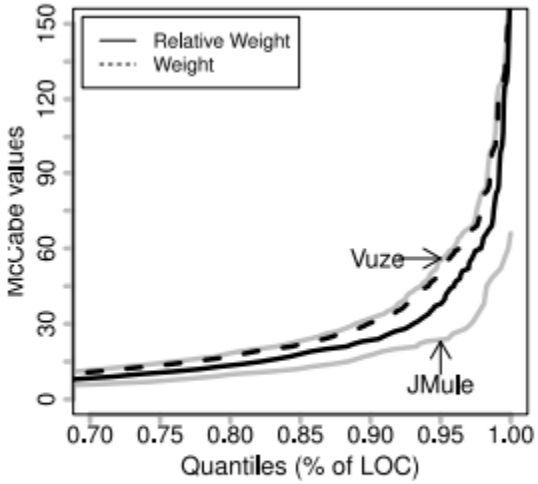


(a) Full distribution (b) Cropped distribution

Fig. 6: Summarized McCabe distribution. The line in black represents the summarized McCabe distribution. Each gray line depicts the McCabe distribution of a single system.

- Figure 6 represents the density functions of the summarized McCabe metric (plotted in black) and the McCabe metric for all individual systems (plotted in gray).
- As expected, the summarized density function respects the shape of individual system's density function.

Use of relatative weight



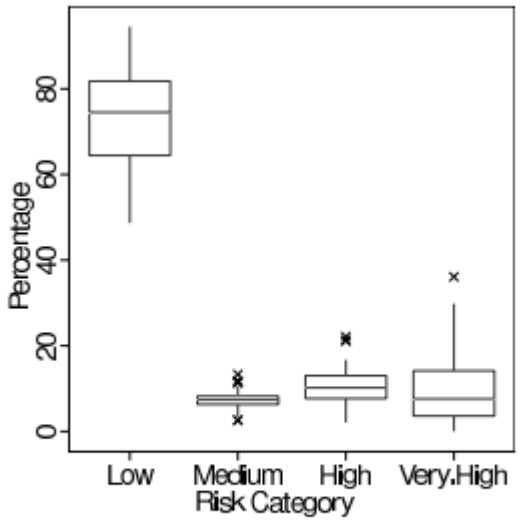
(a) Effect of large systems in aggregation.

- Figure 8a compares the influence of size between simply aggregating all measurements together (black dashed line) and using relative weight (black full line) using as example the Vuze and JMule McCabe distributions (depicted in gray).

Use of relatative weight

- Vuze has about 330 KLOC, while JMule has about 40 KLOC.
- We can observe that the dashed line is very close to the Vuze system meaning that the Vuze system has a stronger influence in the overall result.
- In comparison, using the relative weight results in a distribution in the middle of the Vuze and JMule distributions as depicted in the full line.
- Hence, it is justifiable to use the relative weight since it allows us to be size independent and it takes into account all measurements in equal proportion.

Quality profiles variability



(b) Quality profiles variability

- Figure 7b depicts a box plot for all systems per risk category.
- The x-axis represents the four risk categories, and the y-axis represents the percentage of volume (lines of code) of each system per risk category

Quality profiles variability

- We can observe that there are only a few crosses per risk category, which indicates that most of the systems are represented by the box plot.

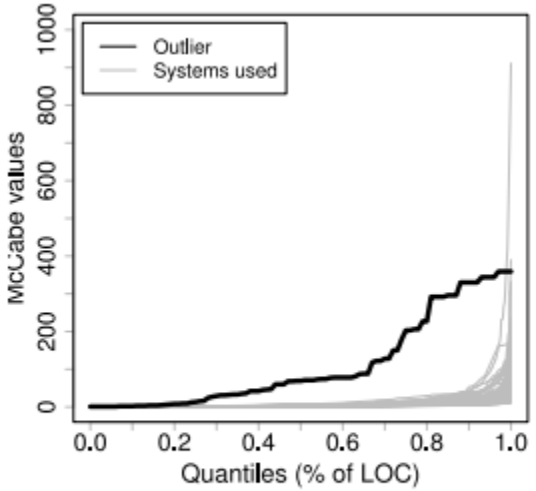
- Finally, for all risk categories, looking to the line in the middle of the box, the median of all observations, we observe that indeed the median is according to our expectations.

- For low risk category, the median is near 70%, while for other categories the median is near 10% which indicates that the derived thresholds are representative of the chosen percentiles.

Outliers

- In statistics, it is common practice to check for the existence of outliers.
- An outlier is an observation whose value is distant relative to a set of observations.
- Outliers are relevant because they can obfuscate the phenomena being studied

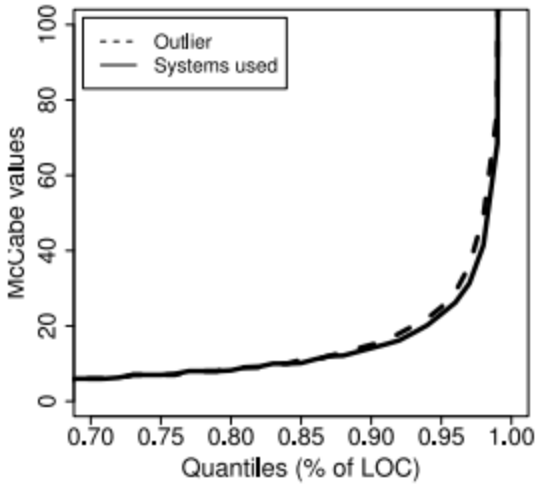
Outliers



(a) McCabe distribution with an outlier.

- Figure 9a depicts the distribution of the McCabe metric for our set of 100 systems (in gray) plus one outlier system (in black).
- We can observe that the outlier system has a metric distribution radically different from the other systems

Outliers



(b) McCabe characterization with and without an outlier.

- Full line: summarizes the McCabe distribution for 100 systems
- Dashed line: result of the 100 systems plus the outlier.

- We can observe that the presence of the outlier has limited influence in the overall result, meaning that our methodology has resilience against outliers

Conclusions

- We proposed a novel methodology for deriving software metric thresholds and a calibration of previously introduced metrics.

- Our methodology improves over others by fulfilling three fundamental requirements:
  - It respects the statistical properties of the metric, such as metric scale and distribution;
  - It is based on data analysis from a representative set of systems (benchmark);
  - It is repeatable, transparent and straightforward to carry out

- Our methodology was applied to a large set of systems and thresholds were derived by choosing specific percentages of overall code of the benchmark.

Getting What You Measure:
four common pitfalls in using software metrics for project management

Eric Bouwers, Joost Visser, Arie van Deursen

ACM Queue, 2012

Introduction

- We have used software metrics to track the ongoing development effort of more than 400 systems.
- In the course of these projects, we have learned some pitfalls to avoid when using software metrics for project management.
- This article addresses the four most important of these:
  - Metric in a bubble
  - Treating the metric
  - One-track metric
  - Metrics galore
- This article focuses on metrics calculated on a particular version of the code base of a system.

Pitfall #1: Metric in a Buble

- Using a metric without proper interpretation.
  - Recognized by not being able to explain what a given value of a metric means.
  - Can be solved by placing the metric in a context with respect to a goal.

- The usefulness of a single data point of a metric is limited.

- To be useful, the value of the metric should, for example, be compared against data points taken from the history of the project or from a benchmark of other projects.

Pitfall #2: Treating the metric

- Making alterations just to improve the value of a metric.
  - Recognized when changes made to the software are purely cosmetic.
  - Can be solved by determining the root cause of the value of a metric.

- The most common pitfall is making changes to a system just to improve the value of a metric, instead of trying to reach a particular goal.

- At this point, the value of the metric has become a goal in itself, instead of a means of reaching a larger goal.

Pitfall #3: One-track metric

- Using only a single software metric to measure whether you are on track toward your goal reduces that goal to a single dimension (i.e., the metric that is currently being measured).
- A goal is never one-dimensional, however
- Software projects experience constant tradeoffs between delivering desired functionality and nonfunctional requirements such as security, performance, scalability, and maintainability.
- Therefore, multiple metrics are necessary to ensure that your goal, including specified tradeoffs, is reached

Pitfall #4: Metrics Galore

- Although using a single metric oversimplifies the goal, using too many metrics makes it hard (or even impossible) to reach your goal.

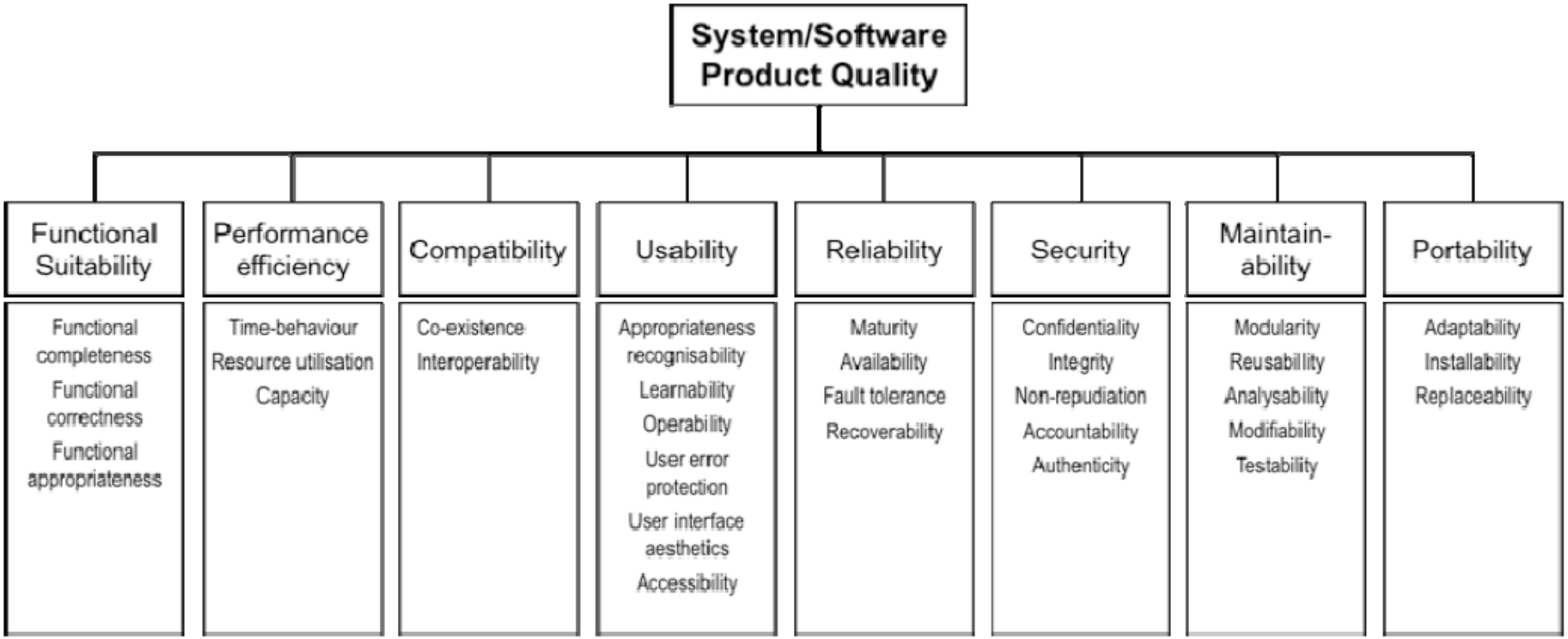
Conclusion

- To benefit from the full potential of metrics, keep the following recommendations in mind:
  - Attach meaning to each metric by placing it in context and defining the relationship between the metric and your goal, and avoid making the metric a goal in itself.
  - Use multiple metrics to track different dimensions of your goal, but avoid demotivating a team by using too many metrics.

- Video:
  - http://www.youtube.com/watch?v=2DZ87M3rKEE

ISO/IEC 25010

Product quality model



Mining the Impact of Evolution Categories on Object-Oriented Metrics

Henrique Rocha, Cesar Couto, Cristiano Maffort, Rogel Garcia, Clarisse Simoes, Leonardo Passos, Marco Tulio Valente

Software Quality Journal, 2013

Introduction

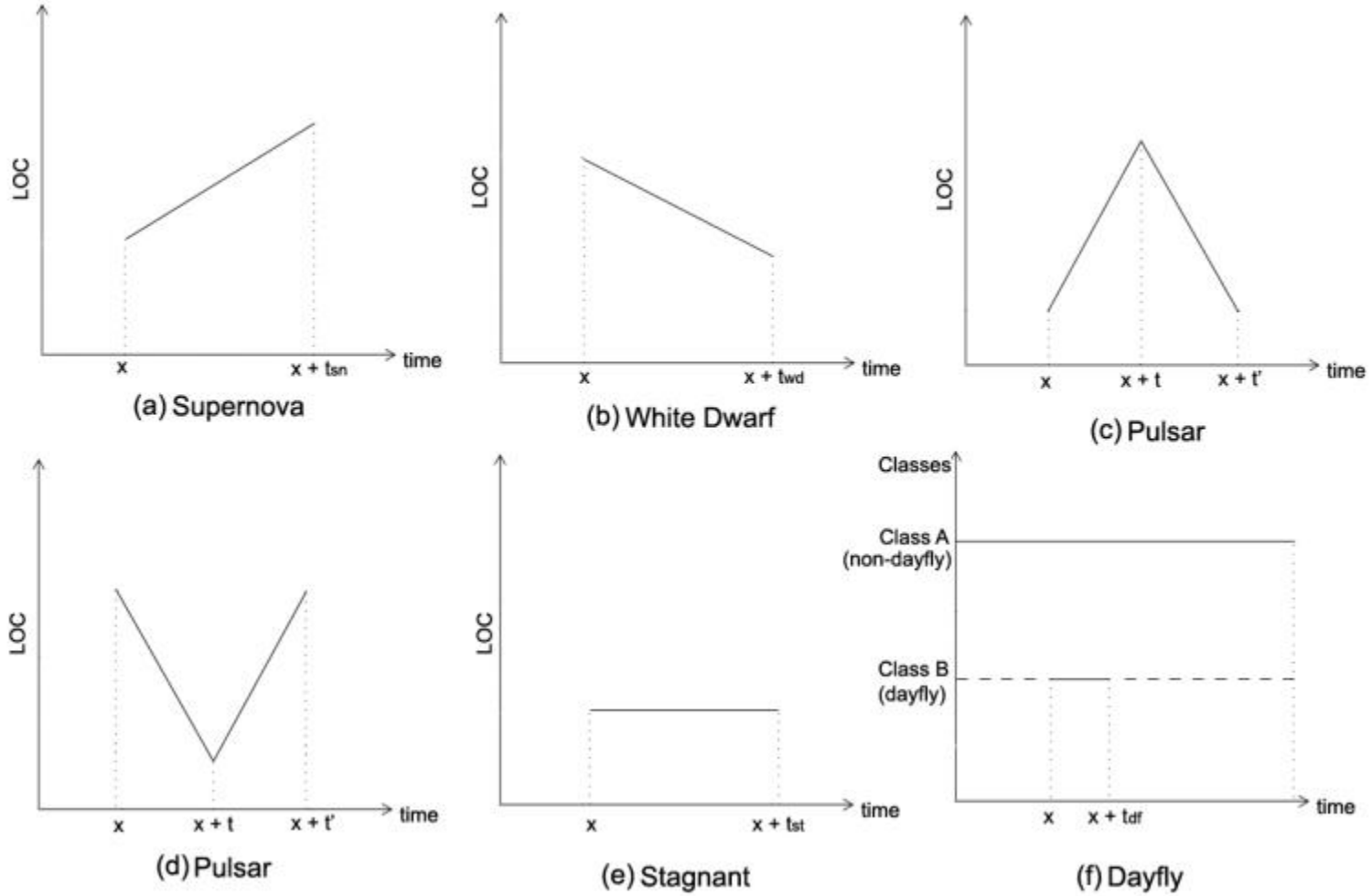
- As expressed by the Laws of Software Evolution, evolution usually contributes to increased software size and complexity

- However, despite the importance of the evolution phase, there are few studies in the literature aiming to evaluate the main patterns that describe the growth of software systems

Lanza's Evolution Categories

- One study of patterns in software evolution is the work of Lanza.
- The categories proposed by Lanza rely on a vocabulary mostly taken from the astronomy domain to model the evolution of classes.
- The proposed categorization covers the following phenomena:
  - rapid growth in class size (supernova)
  - rapid decrease in class size (white dwarf)
  - rapid growth followed by rapid decrease in class size or vice-versa (pulsar)
  - class stability (stagnant)
  - limited class lifetime (dayfly).

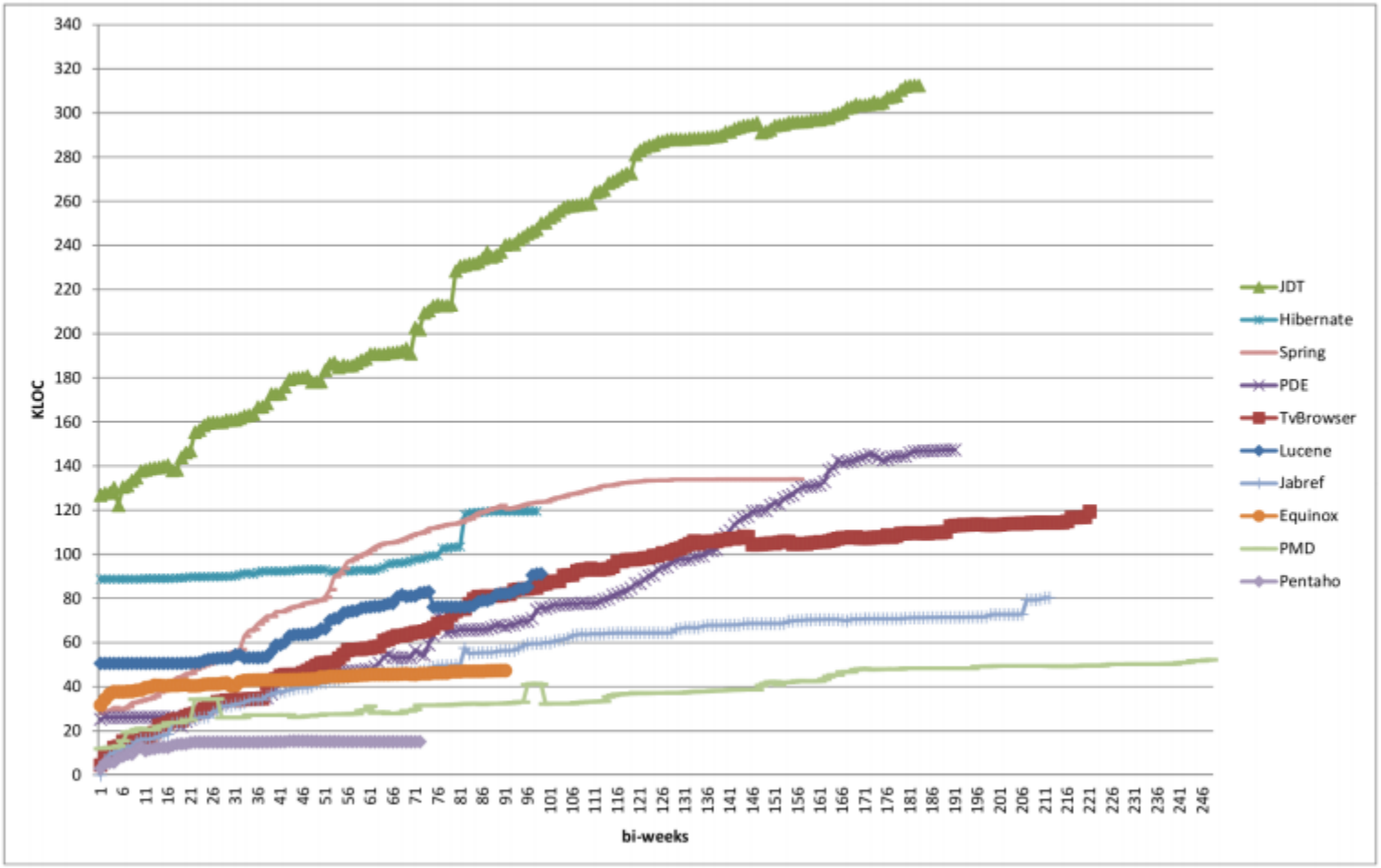
Lanza's Evolution Categories



Dataset

System	Period	# Classes	# Versions
Eclipse JDT	07/01/2001 - 06/14/2008	1368	183
Eclipse PDE	06/01/2001 - 09/06/2008	3082	191
Equinox	01/01/2005 - 06/14/2008	431	91
Lucene	01/01/2005 - 10/04/2008	946	99
Hibernate	06/13/2007 - 03/02/2011	1216	98
Spring	12/17/2003 - 11/25/2009	1845	156
JabRef	10/14/2003 - 11/11/2011	823	212
PMD	06/22/2002 - 12/11/2011	1425	248
TV-Browser	04/23/2003 - 08/27/2011	1229	221
Pentaho	04/01/2008 - 12/07/2010	317	72
Total	-	12,682	1571

Growth Functions



Best Growth Model

System	Best Growth Model	Growth Equation	R^2
Eclipse JDT	linear	$y = 1106.3\,x + 129523$	0.97
Eclipse PDE	linear	$y = 720.85\,x + 9571.1$	0.97
Equinox	linear	$y = 115.9\,x + 37885$	0.90
Lucene	linear	$y = 442.62\,x + 44250$	0.91
Hibernate	superlinear (quadratic)	$y = 6.6x^2 - 336.8\,x + 92486$	0.91
Spring	sublinear (quadratic)	$y = -6.1\,x^2 + 1698.5\,x + 16397$	0.99
JabRef	sublinear (quadratic)	$y = -1.7x^2 + 655.8\,x + 11066$	0.99
PMD	linear	$y = 134.6\,x + 20997$	0.92
TV-Browser	sublinear (quadratic)	$y = -2.6\,x^2 + 1066.9\,x + 5169.6$	0.99
Pentaho	sublinear (logarithm)	$y = 2785.9\,ln(x) + 4584.7$	0.87

Class-level Growth

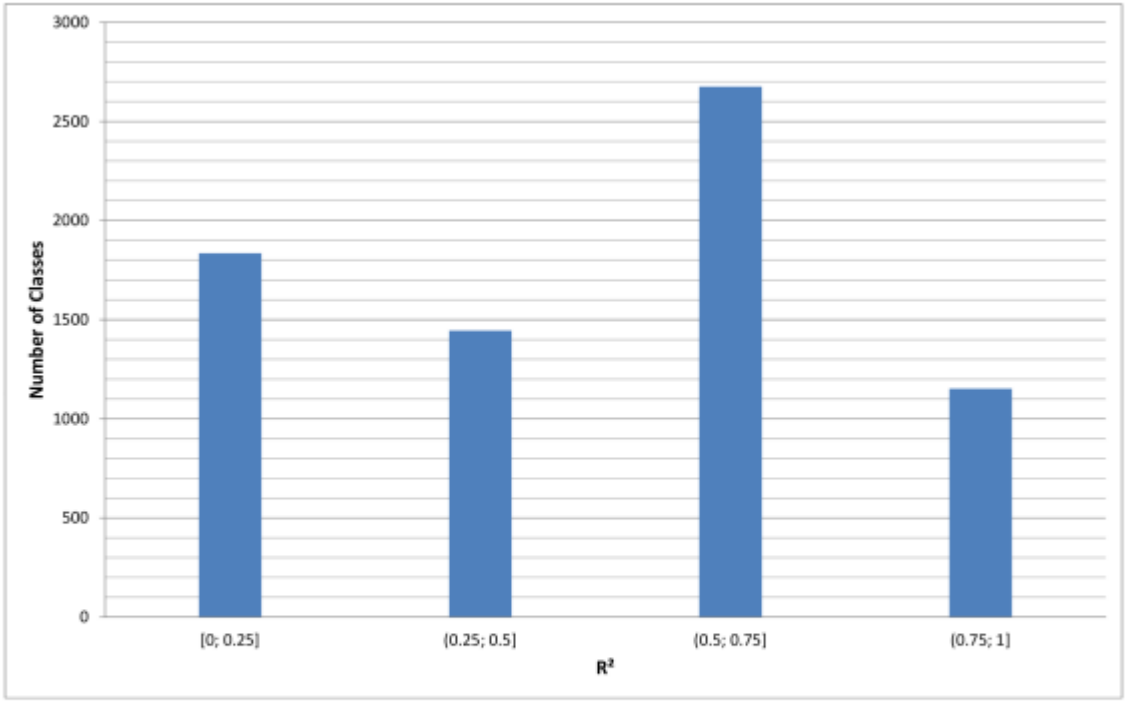
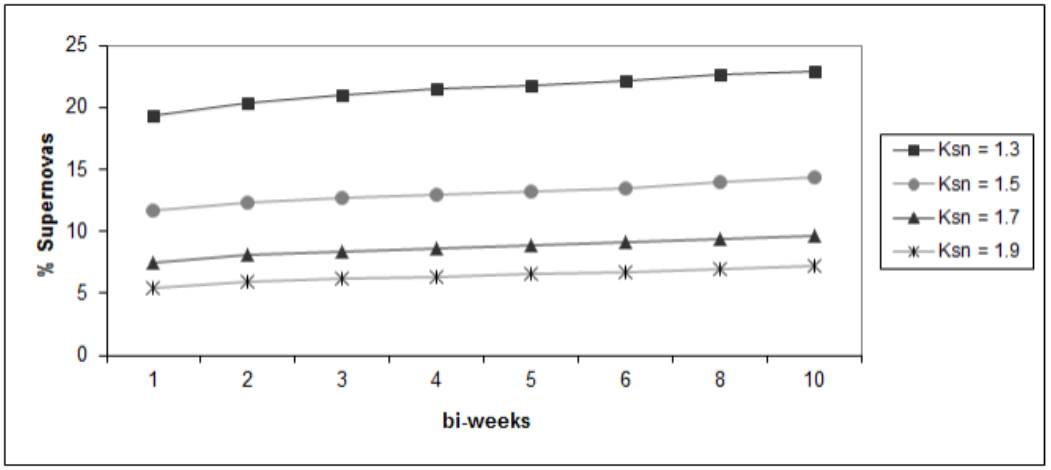
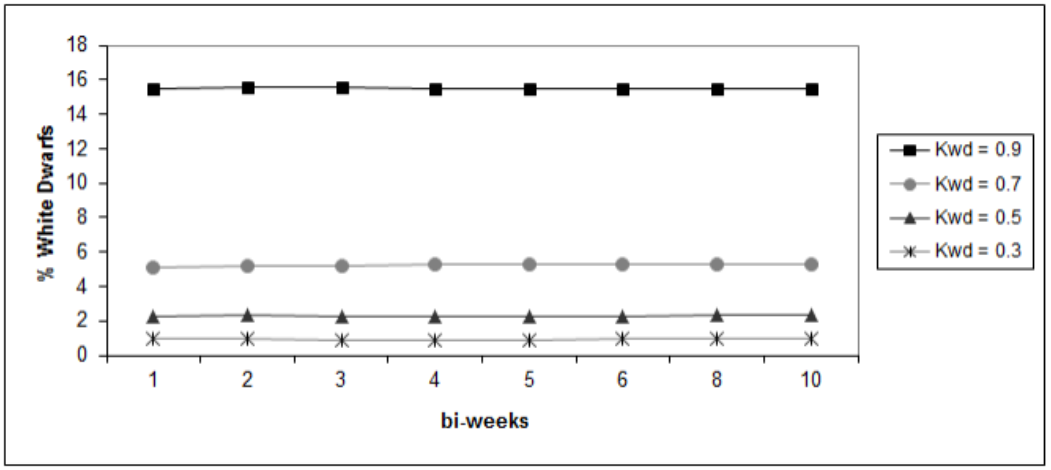


Figure 3: R^2 coefficients for the linear regressions at the class level

Calibration



(a) Supernovas



(b) White Dwarfs

Mining for Categories of Evolution

System	Stagnant	Supernova	White Dwarf	Dayfly	Pulsar
Eclipse JDT	1560 (114.0)	285 (20.8)	89 (6.5)	67 (4.8)	4 (0.2)
Eclipse PDE	1439 (46.6)	380 (12.3)	188 (6.0)	655 (21.2)	13 (0.4)
Equinox	233 (54.0)	40 (9.2)	28 (6.4)	13 (3.0)	1 (0.2)
Lucene	589 (62.2)	91 (9.6)	25 (2.6)	16 (1.6)	2 (0.2)
Hibernate	1449 (119.1)	89 (7.3)	13 (1.0)	5 (0.4)	2 (0.1)
Spring	2137 (115.8)	575 (31.1)	170 (9.2)	168 (9.1)	6 (0.3)
JabRef	1865 (226.6)	136 (16.5)	33 (4.0)	32 (3.8)	0 (0.0)
PMD	1545 (108.4)	173 (12.1)	87 (6.1)	78 (5.4)	11 (0.7)
TV-Browser	1685 (137.1)	244 (19.8)	77 (6.2)	91 (7.4)	3 (0.2)
Pentaho	106 (33.4)	40 (12.6)	13 (4.1)	86 (27.1)	0 (0.0)

Table 3: Absolute and relative occurrences of the evolution categories

Impact on CK Metrics

Metric	Description	Properties
WMC	Weighted Method Count	Size
DIT	Depth of Inheritance Tree	Length
RFC	Response For Class	Size, Coupling
NOC	Number of Children	Size
CBO	Coupling Between Objects	Coupling
LCOM	Lack of Cohesion in Methods	Cohesion

Table 4: CK metrics

Precision and Recall

M	*evol*			
	Dayfly	**Stagnants**	**Supernova**	**White Dwarf**
CBO	1.00 ± 0.00	0.89 ± 0.04	0.38 ± 0.07	0.41 ± 0.16
LCOM	1.00 ± 0.00	0.98 ± 0.01	0.28 ± 0.05	0.20 ± 0.09
WMC	1.00 ± 0.00	0.98 ± 0.01	0.70 ± 0.10	0.71 ± 0.07
RFC	1.00 ± 0.00	0.93 ± 0.04	0.54 ± 0.10	0.62 ± 0.14
DIT	1.00 ± 0.00	0.98 ± 0.02	0.02 ± 0.01	0.04 ± 0.01
NOC	1.00 ± 0.00	0.98 ± 0.01	0.01 ± 0.01	0.00 ± 0.00

Table 5: Precision results (mean $\pm$ standard deviation)

M	*evol*			
	Dayfly	**Stagnants**	**Supernova**	**White Dwarf**
CBO	1.00 ± 0.00	0.84 ± 0.06	0.53 ± 0.01	0.52 ± 0.13
LCOM	1.00 ± 0.00	0.75 ± 0.08	0.26 ± 0.08	0.21 ± 0.11
WMC	1.00 ± 0.00	0.89 ± 0.04	0.70 ± 0.07	0.72 ± 0.07
RFC	1.00 ± 0.00	0.90 ± 0.05	0.68 ± 0.09	0.65 ± 0.01
DIT	1.00 ± 0.00	0.63 ± 0.11	0.08 ± 0.08	0.13 ± 0.31
NOC	1.00 ± 0.00	0.62 ± 0.12	0.10 ± 0.12	0.14 ± 0.31

Table 6: Recall results (mean $\pm$ standard deviation)

Conclusions

- The first study shows that stagnants, supernovas, white dwarfs, and dayfies are probable events in the lifetime of classes.

- The second study shows that by monitoring only the occurrence of the evolution categories we can make reliable predictions of the values of metrics designed to measure:
  - Coupling (CBO)
  - Both coupling and size (RFC)
  - Size (WMC)
  - To a less extent cohesion (LCOM)

- On the other hand, there is no connection between the evolution categories considered in the paper and properties derived from inheritance relations (as measured by NOC and DIT metrics).