Information and Software Technology 72 (2016) 48–67

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Studying the evolution of PHP web applications

Theodoros Amanatidis, Alexander Chatzigeorgiou*

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

ARTICLE INFO

Article history:
Received 26 May 2015
Revised 24 October 2015
Accepted 10 November 2015
Available online 18 December 2015

Keywords:
PHP
Software evolution
Lehman's laws
Software maintenance
Scripting languages
Software repositories

ABSTRACT

Context: Software evolution analysis can reveal important information concerning maintenance practices. Most of the studies which analyze software evolution focus on desktop applications written in compiled languages, such as Java and C. However, a vast amount of the web content today is powered by web applications written in PHP and thus the evolution of software systems written in such a scripting language deserves a distinct analysis.

Objective: The aim of this study is to analyze the evolution of open-source PHP projects in an attempt to investigate whether Lehman's laws of software evolution are confirmed in practice for web applications.

Method: Data (changes and metrics) have been collected for successive versions of 30 PHP projects while statistical tests (primarily trend tests) have been employed to evaluate the validity of each law on the examined web applications.

Results: We found that Laws: I (Continuing Change), III (Self regulation), IV (Conservation of organizational stability), V (Conservation of familiarity) and VI (Continuing growth) are confirmed. However, only for laws I and VI the results are statistically significant. On the other hand, according to our results laws II (Increasing complexity), and VIII (Feedback system) do not hold in practice. Finally, for the law that claims that quality declines over time (Law VII) the results are inconclusive.

Conclusions: The examined web applications indeed exhibit the property of constant growth as predicted by Lehman's laws and projects are under continuous maintenance. However, we have not found evidence that quality deteriorates over time, a finding which, if confirmed by other studies, could trigger further research into the reasons for which PHP web applications do not suffer from software ageing.

1. Introduction

Scripting languages originated as easy-to-use, specialized, interpreted programming languages supporting loose data typing but quickly evolved to robust, generic and high-level languages boosting the development of the Web [1]. The popularity of scripting languages nowadays is clearly evident from the statistics in open-source repository hosting providers such as SourceForge[1] and GitHub[2]. Languages such as PHP, Javascript, Python, Perl and Ruby are among the most popular choices for developing client and server side applications, supported by huge communities and vast documentation. PHP in particular has been widely employed in servers around the world as part of the LAMP (Linux-Apache-MySQL-PHP) platform. The top-ten programming languages and the accompanying project share are shown in Table 1 for two open source software repository hosting providers.

The popularity of scripting languages can possibly be attributed to their ease of use, enabling rapid application development and shielding from low-level issues such as memory management [1]. According to Prechelt [2], who contrasted the implementation time for developing in scripting languages (Perl, Python, Rexx and Tcl) with the time for programming the same functionality in C/C++/Java, development time for scripting languages is significantly smaller (about half of the time for compiled languages). Scripting languages are being viewed by various authors as more appropriate for real programming pragmatism since they unleash the programmer's creativity and imagination [1]. Back in 1998, Ousterhout [3] claimed that new applications will be written entirely in scripting languages while the so-called system programming languages will be used primarily for developing components.[3]

In this work we investigate the evolution of PHP web applications aiming at gaining insight into the way that the corresponding

* Corresponding author. Tel.: +30 2310 891886; fax: +30 2310 891290.
E-mail addresses: tamanatidis@uom.gr (T. Amanatidis), achat@uom.gr (A. Chatzigeorgiou).

[1] http://sourceforge.net

[2] http://github.com

[3] Nevertheless, the debate over the superiority of statically typed languages with respect to maintainability remains open. For example, recent empirical evidence [4] has shown that static types are beneficial to understanding undocumented code and fixing of type errors.

http://dx.doi.org/10.1016/j.infsof.2015.11.009
0950-5849/

Table 1
Top-ten languages of public open source projects hosted by sourceForge & Github.

SourceForge			Github		
Language	# of projects	Percentage (%)*	Language	# of repositories	Percentage (%)*
Java	53.575	23	JavaScript	1.666.302	22
C++	43.189	19	Java	1.413.447	19
PHP	**33.789**	**15**	Ruby	888.679	12
C	31.837	14	Python	814.449	11
C#	17.053	7	**PHP**	**697.898**	**9**
Python	16.585	7	CSS	529.392	7
JavaScript	13.884	6	C++	439.423	6
Perl	10.012	4	HTML	432.546	6
Unix Shell	4.775	2	C	386.232	5
VB .NET	4.050	2	C#	356.856	5
Total	228.749	100	Total	7.625.224	100

*Percentages refer to the ratio over the total number of projects developed in the top-ten languages.
** Data as of October/2015 has been retrieved from http://sourceforge.net and http://github.com.

software systems are maintained. The motivations for this study are the following three facts: (a) There is a latent perception that scripting languages are not suitable for proper software engineering that can support the maintenance of large-scale software projects [1]. However, such claims can hardly be found in the scientific literature possibly because they are not backed up by real evidence. (b) Academics are often skeptical about the suitability of scripting languages in the context of introductory computer science courses. Nevertheless, it should be noted that there is an increasing number of software engineering courses where concepts are illustrated on languages such as Ruby and Python [5]. (c) Finally, to the best of our knowledge, there is no empirical study investigating the evolution of software projects written in PHP (except for the work in [6]) while there is a large body of research on evolution of software in compiled languages, such as Java.

Software evolution is often studied from the perspective of Lehman's eight laws [7] which characterize trends in size, changes and quality of evolving software systems. Therefore, the main goal of this study is to investigate the validity of Lehman's laws of evolution on PHP web applications. Since similar studies have been performed previously for other programming languages, this analysis can be considered as a replication study contrasting previous findings against those derived for PHP.

The rest of the paper is organized as follows: In Section 2 we discuss related work on software evolution and Lehman's laws of software evolution in particular. The details of our case study design are presented in Section 3 along with information about the examined projects. The validity of Lehman's laws of evolution is examined in Section 4. In Section 5 we summarize our results and compare them to those of previous works. In Section 6, possible implications for software researchers and practitioners are presented. Threats to validity are discussed in Section 7 and finally, we conclude in Section 8.

2. Related work

The analysis of software evolution is one of the most well studied aspects of software development and maintenance. This kind of empirical studies is greatly facilitated by the existence of multiple available data in software repositories allowing the investigation of research questions regarding all facets of a software project, including its source code, documentation, developers, bug reports etc. A comprehensive survey on more than 80 approaches on mining software repositories to investigate aspects of software evolution has been presented by Kagdi et al. [8]. The relation between software evolution and maintenance, highlighting the concept of essential change within an environment, is discussed in the overview paper by Godfrey and German [9].

Table 2
Most updated formulation of Lehman's laws.

Law	Context
(I) Continuing change	*A system must be continually adapted to its users' needs, else it becomes progressively less satisfactory in use.*
(II) Increasing complexity	*As a system evolves, its complexity increases and becomes more difficult to evolve unless work is done to maintain or reduce the complexity.*
(III) Self regulation	*Global E-type system evolution is feedback regulated.*
(IV) Conservation of organizational stability	*The work rate of an organization evolving a software system tends to be constant over time.*
(V) Conservation of familiarity	*The newly introduced content of each new version of the system is constrained by the need to maintain familiarity.*
(VI) Continuing growth	*The size of a system continuously grows over time.*
(VII) Declining quality	*The quality of a system will appear to be declining over time, unless proactive measures are taken.*
(VIII) Feedback system	*The evolution process of software resembles a feedback system.*

Software evolution has been studied since the seventies. Lehman first formulated three basic principles of software evolution, based on the study of the OS/360 operating system, in 1974 [10]. Later, Lehman modified the existing principles and proposed two new ones [11]. In the early eighties, Lehman published a new version of laws III, IV and V [12]. Finally, Lehman published a newer formulation of the laws including additional ones [7] and republished the most current formulations in 2006 [13]. Table 2 lists the most updated formulation of the eight laws of software evolution:

With the rise of open source software, several studies investigated the validity of the laws and in some cases it was found that some of the laws are not confirmed [14]. Godfrey and Tu, examined the evolution of the Linux Kernel [15] and in later work several other open source systems [16]. Their focus was the growth of the kernel, using the LOC as size metric and it was found that Linux had been growing at a geometric rate. Robles et al. [17] examined a wider range of open source systems, including the Linux kernel, as well. In agreement with Godfrey & Tu, they found that smooth growth of systems is not that common and concluded that, in some cases, development of open-source software has not followed the laws as known. In 2008, Mens et al. [18] studied the evolution of Eclipse. They found that laws I and VI were confirmed

in practice (i.e. systems are continually adapted at a constant work rate) while law II was not confirmed (i.e. the complexity does not exhibit an increasing trend). Later, Xie et al. [19] studied the validity of all eight laws of evolution on seven open source projects. They analyzed 653 official releases and cumulatively 69 years of evolution confirming 4 out of 8 laws (I, II, III and VI). Israeli & Feitelson [20] studied the validation of the laws also on the Linux Kernel in 2010. They found that the superlinear growth found by Godfrey & Tu [15,16] and confirmed by Robles et al. [17] changed to linear from one point on. Ultimately they confirmed the 3rd and 4th law unlike the aforementioned studies. In the same year, Businge et al. [21] also examined the validation of the laws on 21 third-party plug-ins of Eclipse. They reached the conclusion that laws I, III and VI are confirmed while V is not. Later, Neamtiu et al. [22], whose work was an expansion of the study by Xie et al. [19], studied nine open source C projects. The authors validated only the 1st and the 6th law, opposing their conclusions in their previous study [19]. In a recent work [23], Kaur et al. studied two C++ projects and found that laws I, II, III, V, VI and VII hold in practice while for IV and VIII they could not reach a safe conclusion.

It is apparent that depending on the examined systems and the approach taken, different laws are confirmed by different studies. A comparative overview of the findings of several studies dealing with the validity of Lehman's laws is provided in Section 5.2 along with the ones observed for PHP code in this paper.

3. Case study design

The objective of this study is to examine whether Lehman's laws of software evolution are confirmed in practice for PHP web applications. To achieve this goal we have analyzed data from 30 PHP projects of various sizes and domains. In the following sub sections the four parts of our design are described. i.e., Goal and Research question, Selection of cases, Employed process and tools and data analysis.

3.1. Goal and research question

The goal of this study, adopting the formalism of the Goal-Question-Metrics (GQM) approach [24] can be stated as:

Analyze successive versions of web applications written in PHP

for the purpose of evaluation
with respect to their evolution
from the perspective of researchers and software developers
in the context of Lehman's laws of software evolution.

According to this goal the following research question can be formulated, that will guide this study:

RQ: *Is the evolution of web applications written in PHP compliant with Lehman's laws of evolution*?

The research question is then decomposed into eight research questions, one for each of Lehman's laws.

3.2. Selection of cases

As already mentioned, our study focuses on web applications developed with the scripting language PHP. The motivation for selecting web applications was that PHP is primarily used in a Web context and particularly in the widely employed LAMP platform (Linux-Apache-MySQL-PHP). The criteria for selecting the projects are:

- The source code should be publicly available (the code is publicly available if the project is distributed over a source code repository hosting provider, like Github).

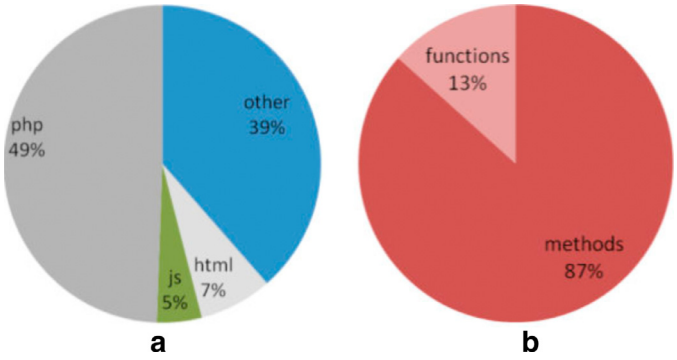

Fig. 1. (a) File and (b) function breakdown of examined projects based on their latest release.

- Projects should have varying sizes and lifespans to obtain a representative sample (e.g. we have selected an almost equal number of projects in three size clusters, 1-10 KLOC, 10-50 KLOC and > 50 KLOC).
- Projects should have at least 5 releases in their history to justify evolution analysis (this information is provided by the repositories).
- Projects should be object-oriented to allow analysis at the class and method level (this requirement has been checked by counting the number of identified classes using the employed tools).

The projects' source code has been retrieved from Github and Sourcefore because of their large collection of projects and widespread usage. The projects that have been selected for this study are obviously a subset of all projects that satisfy the aforementioned criteria. The large projects in our study, namely projects Drupal, Wordpress, laravel, symfony, phpmyadmin and Zendframework, have been selected after discussions with PHP developers who pointed to their importance and indications of high quality. The rest of the projects have been selected by browsing all projects, sorted by relevance and filtering out the ones that did not match the aforementioned criteria. A number of 30 projects has been chosen to enable the manual investigation of the findings and the visual interpretation of the identified trends.

The projects are listed in Table 3 along with an overview of their functionality, their lifespan, size in thousand lines of code and number of analyzed versions. It should be noted that some of the examined projects are relatively small (e.g. Nononsenseforum) while others are large projects with a vast community of developers and users (e.g. WordPress).

By definition web applications entail a multitude of technologies. At a first level, web applications contain source code at the server-side (written in PHP in the examined projects) as well as code that takes over the presentation of web pages to clients (written in HTML, CSS, JavaScript etc). Beyond code, a web application contains also other resources (e.g. images, fonts, media files, etc.) accessed by the codebase. It should be mentioned that object-orientation was introduced in version PHP4 and fully supported since version PHP5. However, the typical PHP web application contains both functions as well as classes (methods). To provide an overall picture of this distribution of content types, Fig. 1 presents the (a) file and (b) function and method breakdown for the latest release of the examined projects. Approximately half of the files are PHP files and almost 9 out of 10 functions are methods.

3.3. Employed process and tools

In order to perform the study, a PHP tool has been developed that is capable of parsing the directories of several project releases (uploaded as a single compressed file) and extracting changes

Table 3
Overview of examined projects.

Project	Functionality	Time frame	LOC (last version)	Versions
Boardsolution	Discussion board	Jan09–May13	88k	8
Breeze	A micro-framework for PHP 5.3+	Apr13–Jul13	9k	18
Cloudfiles	API for the Cloud Files storage system	Oct09–May12	5k	13
Codesniffer	Code Sniffer tokenizes PHP, JavaScript and CSS files and detects coding standard violations	Nov11–Sep13	45k	18
Conference_ci	EllisLab's Open Source Framework	Aug11–Oct12	49k	6
Copypastedetector	Copy/Paste Detector for PHP code	Jan09–Aug13	2k	19
Dotproject	Web-based project management framework	Aug03–Nov09	118k	10
Drupal (core)	Open source CMS	Jan07–Aug14	18k	61
Firesoftboard	Bulletin board software	Mar11–Nov12	66k	5
Generatedata	Random data generator in JS, PHP and MySQL	Jan13–Sep13	136k	11
Laravel	PHP framework	Feb12–Mar13	49k	29
Mustache	Logic-less template engine	Apr10–Aug13	7k	33
Neevo	Database abstraction layer for PHP 5.3+	Jun11–Apr13	8k	13
Nononsenseforum	Simple discussion forum	Jun11–Feb13	1k	25
Openclinic	Medical records system	Aug04–Sep13	16k	10
Phpagenda	Agenda tool	Sep06–Jun13	10k	29
Phpbeautifier	Parses source code and formats it in preferred styles	Apr05–Jun10	7k	12
Phpdaemon	Asynchronous server-side framework for Web-network applications	Oct10–Jul13	31k	10
Phpfreeradius	Web-based tool for managing a FreeRADIUS environment	Apr10–Mar12	31k	8
Phpmyadmin	Database administration tool	Mar10–Oct14	252k	68
Phpmyfaq	A multilingual, completely database-driven FAQ system	Jan10–Jul13	88k	49
Phpqrcode	QRCode generator library	Mar10–Oct10	9k	6
Simplephpblog	Blog	Nov05–Jul12	20k	12
Symfony	PHP framework	Jul11- Oct14	326k	52
Tangocms	A modular content management system	Dec09–Feb12	49k	16
Thehostingtool	Client management script geared towards free web hosting providers	May10–Apr13	27k	6
Usebb	Forum system	Feb05–Jan13	9k	32
Web2project	Business-oriented project management	Jun10–Sep13	120k	5
Wordpress	Blog tool, publishing platform and CMS	Apr05–May14	224k	77
Zendframework2	PHP framework	Sep12–Sep14	284k	25

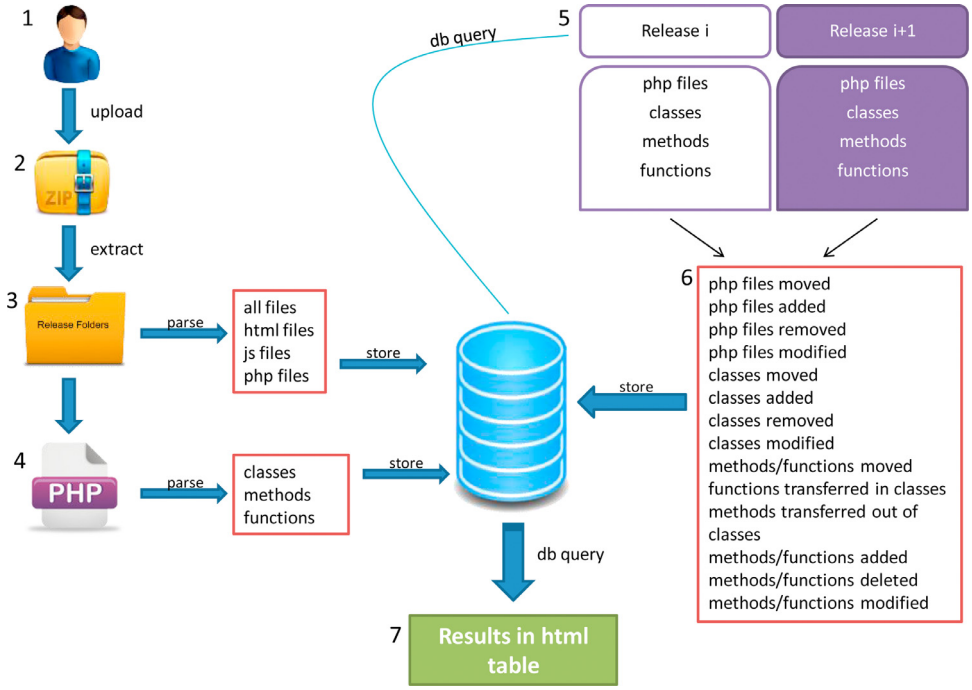

Fig. 2. Workflow for analyzing types and frequency of changes in PHP projects.

between successive releases. Additions, deletions and moves at each level are identified based on the location of the corresponding entity (file, class, function or method), while for the identification of changes the tool examines the percentage of similarity between the body of the same entity in two successive releases (after removing blank lines and comments). The entire workflow is illustrated in Fig. 2.

Once information is extracted from the analyzed source code and directory structure (steps 1–4), raw data is stored in a MySQL database. The developed tool also performs the queries to the database considering two successive releases each time (step 5) and changes are stored in the database (step 6). Eventually the tool displays the results in HTML format (step 7).

Moreover, in order to assess the validity of the laws in a quantitative manner, we employed the PHP Depend[4] tool which performs

[4] http://pdepend.org/

Table 4
Data analysis.

Laws	Variables	Data analysis
Law I (Continuing change)	[V_1] Days Between Releases (DBR)	- Trend test - Slope estimation
Law II (Increasing complexity)	[V_2] Complexity metric: Cyclomatic Complexity Number/Lines Of Code (CCN/LOC)	- Trend test - Slope estimation
Law III (Self regulation)	[V_3] Incremental growth of methods & functions	- Trend test - Slope estimation
Law IV (Conservation of organizational stability)	[$V_{4.1}$] Maintenance effort:Effort = total changes/DBR[$V_{4.2}$] Number of commits	- Trend test - Slope estimation
Law V (Conservation of familiarity)	[V_5] Incremental changes (IC) in methods & functions	- Trend test - Slope estimation
Law VI (Continuing growth)	[V_6] Lines of Code (LOC)	- Trend test - Slope estimation
Law VII (Declining quality)	[$V_{7.1}$] Afferent Coupling (CA)* [$V_{7.2}$] Efferent Coupling (CE)* [$V_{7.3}$] Depth of Inheritance Tree (DIT)* [$V_{7.4}$] Comment Ratio (CR): Commented Lines Of Code/Lines Of Code [$V_{7.5}$] Maintainability Index (MI) [$V_{7.6}$] Number of bug-related commits	- Trend test - Slope estimation
Law VIII (Feedback system)	[V_8] Actual ($\frac{dS}{dt}$) and theoretical growth rate ($c \cdot t^{-\frac{2}{3}}$)	two sample Kolmogorov-Smirnoff test

* These metrics have been measured at class level and their average values (divided by the number of *classes) have been considered.*

static code analysis and computes several software metrics for PHP applications.

3.4. Data analysis

As already made clear, the purpose of this study is to examine whether PHP web applications are evolving in agreement with the Lehman's laws of software evolution. Lehman's laws have been formulated at a rather abstract level, without direct reference (in most cases) to software metrics that can be used to assess them in a quantitative manner [25]. For the mapping of Lehman's laws to measurable indicators we have taken into consideration: (a) the original formulation or examples provided by Lehman, (b) the indicators that have been proposed in previous works that investigated Lehman's laws and (c) the suitability of available metrics which can be computed by the employed tool (PHP Depend) for PHP projects. The association between the investigated laws, involved metrics (variables in our study) and the corresponding statistical tests that will be performed to assess the validity of each law is presented in Table 4. Due to plethora of laws, the motivation for the selection of the particular metrics and the analysis conducted for each law will be separately discussed in the results section (Section 4).

As mentioned above, we mainly focused on the evolution of these metrics over time. Particularly, our goal was to examine if there is a trend in the evolution of each metric that concerns a specific law and if so, to quantify this trend in comparable numbers. The corresponding null hypothesis for each metric x can thus be expressed as:

H_0 : Metric x exhibits no trend
H_1 : Metric x exhibits a trend

In order to determine if a trend is present in the evolution of a metric we employed linear regression and the Mann–Kendall trend test [26]. Linear regression is considered a robust modeling tool. However, to consider the results of a trend test based on linear regression as valid, a number of preconditions have to be satisfied. These assumptions are:

1. Variables should be measured at the continuous level (i.e. they should be either interval or ratio variables). Due to the nature of the examined time series of metric values, this condition is always met.
2. The relationship between dependent and independent variables has to be linear.
3. No significant outliers should exist.

(The 2nd and 3rd assumption can be assessed visually by examining the scatterplot of the two variables i.e. release number and metric value).

4. Observations should be independent. This can be checked using the Durbin–Watson test which assesses whether residuals of a linear regression model exhibit autocorrelation [27].
5. The data should be characterized by homoscedasticity. This can be checked using the Breusch–Pagan test for homoscedasticity [28].
6. The residuals (errors) of the regression line should be normally distributed. This can be checked by conducting the Shapiro–Wilk test of normality [29] on the residuals of the model yielded from the linear regression.

In case the aforementioned assumptions do not hold, one should use a non-parametric test instead. A trend test which can provide reliable results when no distribution can be assumed is the Mann–Kendall trend test [26].

We should note that in the majority of projects one or more assumptions are violated and thus, the Mann–Kendall trend test was mainly used in our study. This is not uncommon when working with real-world data rather than artificially made examples. When according to the Mann–Kendall trend test a trend is clearly evident, i.e. the null hypothesis can be rejected, the Theil–Sen estimator [30] was used in order to calculate the slope of the fitted trendline. The slope obtained by the Theil–Sen estimator is essentially the median slope among all lines through all pairs of points in the dataset.

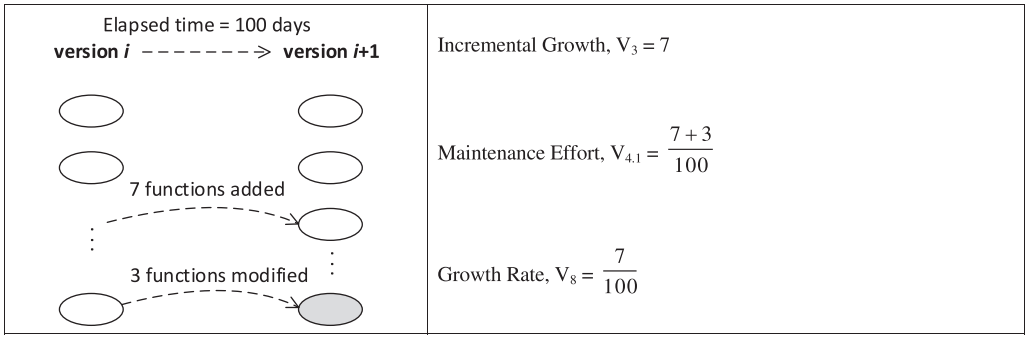

Fig. 3. Calculation of incremental growth, maintenance effort and growth rate (example).

Table. 5
Correlation between variables.

	V_1	V_2	V_3	$V_{4.1}$	$V_{4.2}$	V_5	V_6	$V_{7.1}$	$V_{7.2}$	$V_{7.3}$	$V_{7.4}$	$V_{7.5}$	$V_{7.6}$	V_8
V_1														
V_2							19/30(−0.88)							
V_3														22/30(+0.769)
$V_{4.1}$														18/30(+0.829)
$V_{4.2}$														
V_5														
V_6										15/30(+0.858)				
$V_{7.1}$									18/30(+0.906)					
$V_{7.2}$											18/30(+0.834)			
$V_{7.3}$														
$V_{7.4}$														
$V_{7.5}$														
$V_{7.6}$														
V_8														

*Statistical significance is assessed at the 0.05 level.

To enable the comparison of the steepness of slopes among different projects, slopes should be scale independent. To this end, we performed the trend test analysis (either linear regression or Mann–Kendall trend test) on a normalized version of the original dataset. In particular, each value of an examined time series was divided by the maximum value in the time series yielding a normalized value in the range [0..1] exhibiting the same slope as the original dataset. Moreover we expressed the slope as a percentage to allow easier interpretation of the results.

Due to the nature of Lehman's laws, many of the variables seem to be akin. Especially the variables related to the 3rd, 4th and 8th law seem to be quite similar. For this reason: (a) we illustrate through a simplified example the difference between variables V_3, $V_{4.1}$, and V_8 and (b) we performed correlation analysis among all pairs of selected variables for all 30 examined projects.

Fig. 3 illustrates a hypothetical system that evolved from version *i* to version *i*+1 over a period of 100 days. We assume for simplicity that 7 new functions (methods and functions) have been added, while 3 existing functions have been modified (as changes we would also count removals and moves). The actual values of variables V_3, $V_{4.1}$, and V_8 would then be obtained as shown in the right-hand side of the figure. As it can be observed these values are indeed closely related but capture different aspects of system evolution.

To provide further insight into possible correlation between the selected measures, the filled cells in Table 5 indicate cases where the corresponding row and column variables have a statistically significant correlation (with the same sign in the corresponding Pearson's correlation coefficient) in 50% or more of the projects. For example, variable V_2 (CCN/LOC) has a negative correlation to V_6 (LOC) in 19 out of the 30 projects. The average correlation coefficient for these projects is -0.88. This is rather reasonable, since variable V_6 (LOC) is the denominator of variable V_2 (CCN/LOC). However, we deliberately retain both variables, since measuring the complexity of an evolving system would yield a monotonically increasing trend due to the constant addition of new code, as it will be explained in the next section.

Variables $V_{7.1}$ (afferent coupling) and $V_{7.2}$ (efferent coupling) also appear to have a rather strong correlation. However, these variables quantify different aspects of coupling and we prefer to keep them both in the investigation of the 7th law (nevertheless, it would be worth investigating why these aspects of coupling are correlated in PHP systems).

A strong correlation has been found also between variables $V_{7.2}$ (efferent coupling) and $V_{7.4}$ (comment ratio). We do not have any data to explain this rather unexpected correlation, but we decided to keep comment ratio in the investigation of quality evolution as it quantifies a distinct property of both functions and methods.

Finally, a strong correlation is observed between the variables discussed in the example of Fig. 3, namely between incremental growth (V_3) and growth rate (V_8), and between maintenance effort ($V_{4.1}$) and growth rate. As explained previously, it is reasonable that these variables are correlated as they depend on some common measures. However, because the formulation of the 8th law follows strictly a quantification approach proposed by Turski [31] we did not discard this variable.

Other variables for which we have found a strong correlation to some of the selected ones, have been excluded from the analysis.

The entire dataset on which the study has been performed is publicly available[5].

[5] http://se.uom.gr/index.php/projects/evolution-analysis-php-applications/

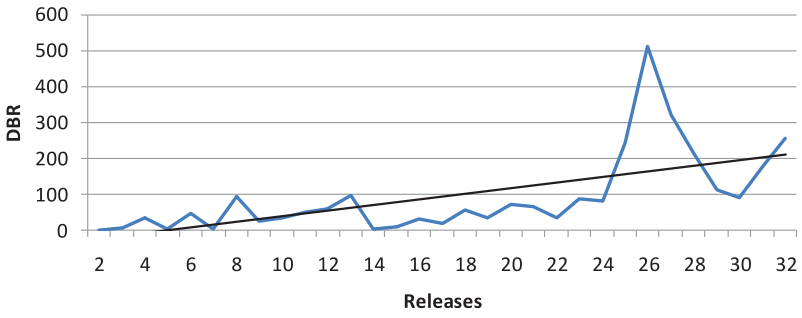

Fig. 4. Trend of days between releases metric for project usebb.

4. Results and discussion

In this section we are going to present and discuss the results concerning the research question of whether the evolution of web applications written in PHP is compliant with Lehman's laws of evolution. To facilitate understanding, a brief reminder of each law will be provided. The hypothesis, the analyzed variables as well as the corresponding type of analysis is also presented for each law. Finally, we explain briefly the rationale behind the selection of the corresponding metrics as well as any concerns that someone could have with the applied approach.

At this point the following clarification should be made: For the laws where the results allow us to draw a conclusion that is supported by statistically significant trend test results, we note that the corresponding law is *statistically* validated or not. However, there are laws, where although the results do not allow the extraction of a statistically significant conclusion, the actual examination of the cases reveals the lack of any evident trend. In these cases, we note whether the corresponding law is *practically* validated or not.

4.1. Law I: continuing change

The law states that a program continuously changes and adjusts to its users' needs else it becomes progressively less satisfactory [7]. This is another way of stating that system maintenance is an inevitable process [32]. It is a general observation which is valid for all projects that deliver consecutive releases in a repository, otherwise there wouldn't be a need to release new versions. Law I is confirmed by all studies on Lehman's laws (see Section 5.2 – Comparison with previous work), including ours. The usual way to assess the validity of this law has been to investigate the cumulative number of modified modules [22]. We have also employed the cumulative number of changed methods and functions in PHP code and found a steady increasing trend in all projects, implying that changes are present throughout projects' lifespan. However, a trend is by definition almost always present in a cumulative function, unless no modules are introduced at all during the course of a project, which is rather unlikely. Therefore, our goal was not only to assess the validity of the law *per se*, but also to quantify whether the validity of the law becomes weaker over time or not.

To obtain an insight on whether the first law of Lehman weakens or strengthens over time, we have measured the Days Between Releases (DBR), denoting the number of days that elapsed from the release of one version in the repository up to the release of the next one. In other words, DBR quantifies the frequency at which new releases are published. An increase of DBR over time means that the rate of publishing new releases decreases, which in turn can be interpreted as a weakening of the validity of the law for a particular project. Thus the corresponding hypothesis can be expressed as:

Hypothesis	Variable	Analysis
H_0: The evolution of the time interval between two successive releases exhibits no trend. H_1: The evolution of the time interval between two successive releases exhibits a trend.	[V_1]: Days Between Releases (DBR)	- Trend test - Slope estimation

Rationale for selected variable: Previous research has used the cumulative number of modified functions/methods; however, a cumulative number would be monotonically increasing. Therefore, we assume that the law is valid and measure the Days Between Releases to assess the frequency at which new releases are published (i.e. whether the law is strengthened over time).
Concerns: The elapsed time between releases does not necessarily reflect the amount of changes that have been carried out, especially in open-source projects.

For example, Fig. 4 illustrates the evolution of DBR for the successive versions of project usebb. It appears that the number of days required to release a new version increases over time (less than 50 days for the initial versions which climbs to more than 200 days for the final versions) implying that more effort is required to adapt the system to additional requirements.

As already mentioned, to perform a systematic analysis regarding the presence of a trend in a time series, we will be using appropriate trend tests and slopes estimation (as explained in Section 3.4). Table 6 lists the results of the conducted trend test for each project as well as the slopes for the cases where the trend is statistically significant. In the 'Trend' column an up-pointing/down-pointing arrow indicates the presence of a statistically significant trend while a blank cell indicates that there is no evidence for the existence or the absence of a trend.

As it can be observed, in 2 out of the 30 projects DBR decreases over time (i.e. a negative slope is observed) and in 7 out of 30 projects DBR increases. For 21 projects there is no statistical evidence for the existence or the absence of a clear trend. Therefore, we cannot argue about the validity of this law based on statistically significant results. However, to shed light on the evolution of DBR for the majority of the projects that do not exhibit a statistically significant trend, we depict graphically their evolution in Fig. 5. The *x*-axis corresponds to normalized version numbers, in the sense that all project lifespans are plotted as equal, for the sake of clarity. The *y*-axis does not contain units, as the curves have been adjusted to minimize their overlap.

As it can be observed, indeed most of the projects shown in Fig. 5 do not exhibit a clear trend but rather have fluctuations in the variable of interest (DBR). One could argue, that DBR does not increase nor decrease steadily during the examined period and characterize this evolution as ratheryes stable.

These observations imply that the first law of Lehman does not become stronger (changes are not becoming more frequent) or weaker over time. In other words, findings suggest that PHP systems continuously change but, in this study it cannot be determined whether these changes happen at a slower or a faster pace.

Table 6
Statistical results on law I (continuing change).

	Project	DBR				Project	DBR		
		p-value	Trend	Slope (%)			*p*-value	Trend	Slope (%)
1	Boardsolution	0.287			16	Phpagenda	0.001	↑	0.07
2	Breeze	0.041	↑	0.16	17	phpbeautifier	0.310		
3	Cloudfiles	0.086			18	Phpdaemon	0.029	↓	−2.78
4	Codesniffer	0.366			19	Phpfreeradius	0.764		
5	Conference_ci	0.462			20	Phpmyadmin	0.001	↓	−0.39
6	Copypastedetector	0.471			21	Phpmyfaq	0.557		
7	Dotproject	0.754			22	Phpqrcode	0.086		
8	Drupal (core)	0.927			23	Simplephpblog	0.087		
9	Firesoftboard	1.000			24	Symfony*	∼0.000	↑	0.14
10	Generatedata	0.525			25	Tangocms	0.546		
11	Laravel	0.003	↑	0.83	26	Thehostingtool	1.000		
12	Mustache	0.025	↑	1.19	27	Usebb	∼0.000	↑	1.01
13	Neevo	0.783			28	Web2project	1.000		
14	Nononsenseforum	0.274			29	Wordpress	0.805		
15	Openclinic	0.602			30	Zendframework2	0.014	↑	1.25

* Linear regression has been used for these projects and the Mann–Kendall trend test for the rest to obtain *p*-values.

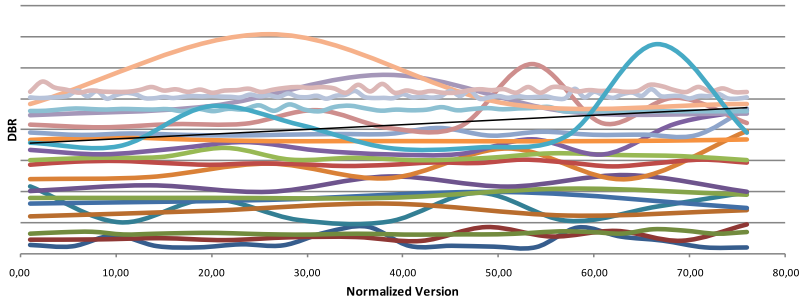

Fig. 5. Evolution of days between releases metric for projects with *p*-value > 0.05.

4.2. Law II: increasing complexity

According to this law the complexity of software increases over time unless proactive measures are taken to reduce or stabilize the complexity [7]. Although the complexity of a software project can be quantified in many ways, we have chosen the widely acknowledged cyclomatic complexity measure [33] since it manages to assess the complexity of both functions and methods present in most PHP web applications nowadays. However, the CCN metric provided by the PHP Depend tool counts the total available decision paths in the entire program, and thus would be monotonically increasing as the system becomes larger in size over time. Therefore, we normalized its value over the lines of code, i.e. we calculate CCN/LOC. An increase of CCN/LOC over time implies that the overall complexity increases and that the law is valid. The corresponding hypothesis can be expressed as:

Hypothesis	Variable	Analysis
H_0: The evolution of complexity exhibits no trend. H_1: The evolution of complexity exhibits a trend.	[V_2]: CCN/LOC	- Trend test - Slope estimation

Rationale for selected variable: Cyclomatic complexity is a well-studied and widely acknowledged complexity measure which has also been employed in previous studies for the examination of the validity of the 2nd Law.
Concerns: The normalization by dividing with the size might not capture changes in total complexity due to the addition of new code.

The trend of CCN/LOC over all examined versions for each project is shown in Table 7. Fig. 6 illustrates the trendline fitted to the evolution of CCN/LOC, for those projects where a statistically significant trend has been found. The x-axis corresponds to normalized version numbers, in the sense that all project lifespans are plotted as equal, for the sake of clarity.

As it can be observed from Table 7, in 18 projects (more than half of the projects) there is either a positive or a negative trend in the evolution of the aforementioned complexity measure. Out of the 18 projects in which the null hypothesis is rejected (meaning that a statistically significant trend is present), only in 6 projects there is a deterioration in the evolution of the aforementioned complexity measure, implying that the law is not valid for the examined PHP projects. For the majority of the projects, complexity decreases. This generally decreasing trend is also evident from the CCN/LOC trendlines in Fig. 6. To be accurate, we should remind that Lehman acknowledged the possibility of a non-increasing complexity if care is exercised by the maintenance team and this seems to be the case for the examined PHP projects. This observation is in agreement with a previous study [6] on the evolution of large-scale PHP web applications, which suggested that systems like phpMyAdmin, WordPress and Drupal exhibit signs of careful maintenance decisions resulting in non-increasing complexity.

4.3. Law III: self regulation

Lehman [7] suggested that "*system evolution process is self regulating*". In contrast to other rules, mapping this claim to the evolution of quantitative measures is non-trivial. According to Xie et al. [19] the regulation of size throughout the lifespan of a project, translates to observing negative and positive adjustments ("ripples") in the growth trend. The same interpretation of the third law has been adopted by Businge et al. [21] who observed ripples in the incremental growth of Eclipse plugins. To this end, we have measured the changes in the total number of functions and methods. For example, such changes for project phpMyFAQ are graphically depicted in Fig. 7. As it can be observed, ripples are present; positive adjustments are more frequent than negative,

Table 7
Statistical results on law II (increasing complexity).

	Project	CCN/LOC				Project	CCN/LOC		
		p-value	Trend	Slope (%)			*p*-value	Trend	Slope (%)
1	Boardsolution	0.319			16	Phpagenda	~0.000	↓	−0.40
2	Breeze	0.034	↑	0.03	17	Phpbeautifier	0.019	↓	−0.34
3	Cloudfiles	0.853			18	Phpdaemon	0.474		
4	Codesniffer	~0.000	↑	1.22	19	Phpfreeradius	0.711		
5	Conference_ci	0.181			20	Phpmyadmin	~0.000	↓	−0.51
6	Copypastedetector	0.003	↓	−0.14	21	Phpmyfaq	0.016	↓	−0.18
7	Dotproject	0.371			22	Phpqrcode	0.035	↓	−0.79
8	Drupal (core)	~0.000	↓	−0.86	23	Simplephpblog	0.099		
9	Firesoftboard*	0.011	↓	−0.02	24	Symfony	~0.000	↓	−0.05
10	Generatedata	0.002	↓	−0.17	25	Tangocms	~0.000	↑	0.19
11	Laravel	0.763			26	Thehostingtool	0.024	↑	2.01
12	Mustache	0.026	↓	−0.60	27	Usebb	0.909		
13	Neevo	0.112			28	Web2project	0.086		
14	Nononsenseforum	~0.000	↑	1.91	29	Wordpress	~0.000	↓	−0.20
15	Openclinic	0.149			30	Zendframework2	~0.000	↑	0.07

* Linear regression has been used for these projects and the Mann-Kendall trend test for the rest to obtain p-values.

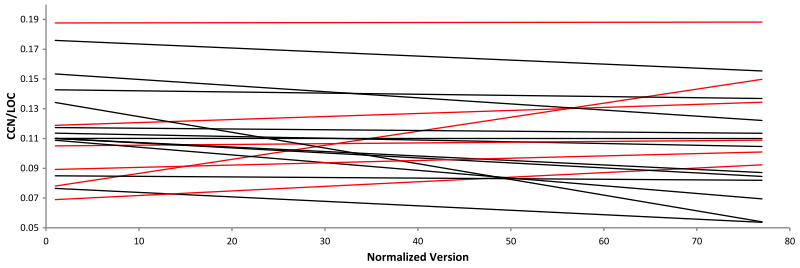

Fig. 6. Trendlines of CCN/LOC for projects with *p*-value < 0.05.

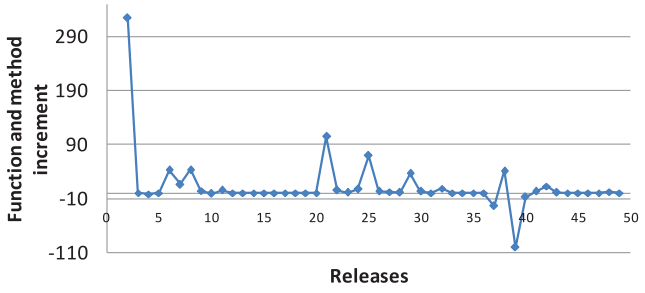

Fig. 7. Ripples in the total number of functions/methods for phpMyFAQ.

in agreement to what has been observed by the study of Xie et al. [19] and Businge et al. [18]. However, no global trend appears to be present. To have a common interpretation of whether the law is confirmed across all projects, we investigate whether there is a statistically significant trend in the data. The law should be considered as invalidated when there is a trend at the incremental growth of the methods and functions of the system. The corresponding hypothesis can be expressed as:

Hypothesis	Variable	Analysis
H_0: The evolution of incremental growth exhibits no trend. H_1: The evolution of incremental growth exhibits a trend.	[V_3]: incremental growth of methods & functions	- Trend test - Slope estimation

Rationale for selected variable: Methods and functions in PHP code cumulatively reflect the amount of delivered functionality. Incremental growth of system characteristics (e.g. functions, dependencies) has been used in other studies as well.
Concerns: Evolution might occur at a lower level than methods and functions (i.e. at the code line level) without affecting the number of methods and classes.

The results of the statistical analysis are summarized in Table 8. Only in 3 out of the 30 projects a trend in the incremental growth of methods and functions is present. In *laravel* and *symfony* there is an increasing trend, meaning that more and more functionality is added over time, while in *dotproject* the trend is decreasing. In the rest of the projects, we cannot reject the null hypothesis that the incremental growth of the system exhibits no trend. However, if we take a look at Fig. 8, which illustrates graphically the evolution of the incremental growth for all 27 projects where no statistically significant trend has been found, we can observe that indeed there is no evidence for a constant increase or decrease in the number of incremental methods and functions at every new version. This means that the examined systems do grow, but the growth rate remains relatively stable. To sum up, we cannot conclude in terms of statistical power that the 3rd Law is valid, but the actual evidence point to the conclusion that the evolution of PHP projects is indeed regulated under a stable growth pace during system's lifespan. Hence, we consider the law as *practically* validated.

4.4. Law IV: conservation of organizational stability

The law stipulates that the activity/work rate between successive releases remains stable. Estimating effort in open-source projects can hardly be accurate and only indirect measures can be considered. In analogy to the study by Xie et al. [19] we measure the work rate as the number of changes (in the number of methods and functions) in a release *i*, over the elapsed time (in days) from the previous release *i*-1. As suggested by Lehman [34,35] we count as changes all handled elements accounting for removed, modified, added and moved functions and methods. Moreover, to provide an alternative measure for the estimation of work rate we analyzed the number of commits to the corresponding repository over time. Since a commit implies an 'official' submission of performed work, it can be considered as a reliable indicator of effort.

Table 8
Statistical results on law III (self regulation).

	Project	INCREMENTAL GROWTH				Project	INCREMENTAL GROWTH		
		p-value	Trend	Slope (%)			*p*-value	Trend	Slope (%)
1	Boardsolution	1.000			16	Phpagenda	0.533		
2	Breeze	0.426			17	Phpbeautifier	0.065		
3	Cloudfiles	0.528			18	Phpdaemon	0.602		
4	Codesniffer	1.000			19	Phpfreeradius	0.095		
5	Conference_ci	0.579			20	Phpmyadmin	0.277		
6	Copypastedetector	0.811			21	Phpmyfaq	0.285		
7	Dotproject	0.016	↓	−0.24	22	Phpqrcode	0.267		
8	Drupal (core)	0.079			23	Simplephpblog	0.436		
9	Firesoftboard	0.734			24	Symfony	0.011	↑	0.01
10	Generatedata	0.653			25	Tangocms*	0.118		
11	Laravel	0.024	↑	0.04	26	Thehostingtool	1.000		
12	Mustache	0.960			27	Usebb	0.901		
13	Neevo*	0.077			28	Web2project	0.734		
14	Nononsenseforum	0.248			29	Wordpress	0.811		
15	Openclinic	0.295			30	Zendframework2	0.130		

* Linear regression has been used for these projects and the Mann–Kendall trend test for the rest to obtain p-values.

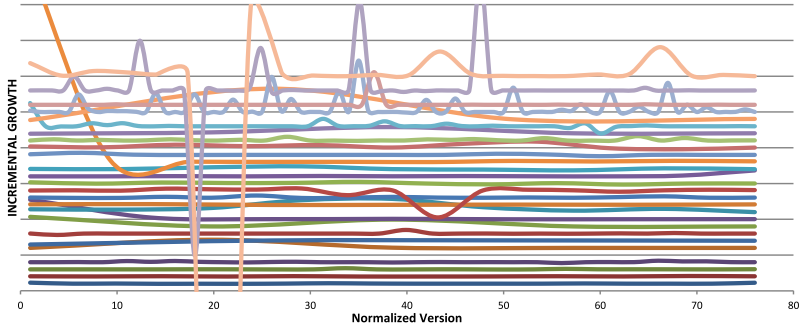

Fig. 8. Evolution of incremental growth for projects with *p*-value > 0.05.

Although this law is considered *sub judice* (under judgment) in the corresponding study by Lehman, we attempt to investigate the validity of the law by assessing the slope of the fitted trendline of maintenance effort, as reflected in the two variables. The statistical results for the trend test on variable $V_{4.1}$ and $V_{4.2}$ are shown in Table 9.

Hypothesis	Variable	Analysis
H_0: The evolution of maintenance effort exhibits no trend. H_1: The evolution of maintenance effort exhibits a trend.	[$V_{4.1}$]: maintenance effort = changes/DBR [$V_{4.2}$]: number of commits	- Trend test - Slope estimation
Rationale for selected variables: As suggested by Lehman we counted the changes in methods and functions throughout a project's lifespan. Moreover, a commit constitutes an actual and 'official' submission of work by the developers. **Concerns**: The work that has been performed to release a new version is not reflected accurately when counting source code modifications only, since other types of activities (such as understanding and testing) might have been carried out.		

As it can be observed only in 9 projects (for $V_{4.1}$) and in 10 projects (for $V_{4.2}$) there is a statistically significant trend in the maintenance effort. For the majority of projects, we cannot reach any safe conclusion regarding the evolution of maintenance effort. Once again, we plot these non-statistically significant cases in Fig. 9 for $V_{4.1}$ and in Fig. 10 for $V_{4.2}$.

The visual interpretation of Fig. 9 indicates that in general, the work rate does not increase or decrease drastically as the projects evolve. It should be noted that although some lines appear almost straight, the statistical power was low because of the small number of data points. The evolution of the number of commits in Fig. 10 exhibits fluctuations for some of the projects, but again no conspicuous trend is present. Overall, PHP projects seem to evolve in agreement with the 4th Law. An increasing trend would imply that more and more features (or bug fixes) are added to the evolving project in the same period of time, or that the same amount of functionality is added in less and less time. However, it is reasonable to assume that increasing addition of functionality is rather rare for mature open-source projects and especially web applications which have to deliver their core functionality right from their first versions. On the other hand, a decreasing trend would imply that the system suffers from poor maintainability, in the sense that equal amounts of functionality required more time to be added. However, this phenomenon has not been observed meaning that the majority of the examined web applications do not suffer from this kind of maintainability issues. We tag this law as *practically* validated.

4.5. Law V: conservation of familiarity

According to Lehman, "During the active life of a program the release content of the successive releases of an evolving program is statistically invariant" [7]. The law resulted by noticing the inherent tradeoff between the increased difficulty of understanding changes contained in a new release and the organizational pressure for delivering novel features along with the constant demand for corrections and changes [13]. In order to assess the validity of the law in a quantitative manner, the Incremental Changes (IC) metric has been proposed [36]. IC is obtained by subtracting the total number of changes that occurred in methods and functions in one release from the total number of changes in methods and functions of the next release. An absence of trend for IC indicates the absolute validity of the law. A decreasing trend implies that the performed changes become less and less over time, which in

Table 9
Statistical results on law IV (conservation of organizational stability).

	Project	Maintenance effort			Number of commits		
		p-value	Trend	Slope (%)	*p*-value	Trend	Slope (%)
1	Boardsolution	0.368			0.251*		
2	Breeze	0.091			N/A	N/A	N/A
3	Cloudfiles	0.732			0.069*		
4	Codesniffer	0.711			0.038	↑	0.93
5	Conference_ci	0.462			0.007	↓	−1.8
6	Copypastedetector	0.622			0.746		
7	Dotproject	0.175			0.001	↓	−1.12
8	Drupal (core)	0.589			0.189		
9	Firesoftboard	0.105*			0.450*		
10	Generatedata	1.000			0.463		
11	Laravel	0.402			0.002	↓	−2.94
12	Mustache	0.023	↓	−0.14	0.194		
13	Neevo	0.033	↓	−2.27	0.039	↓	−5.32
14	Nononsenseforum	0.049	↑	0.09	0.034	↓	−5.69
15	Openclinic	0.754			0.656		
16	Phpagenda	0.020	↓	−0.11	N/A	N/A	N/A
17	Phpbeautifier	1.000			0.332		
18	Phpdaemon	0.016	↑	7.46	0.653		
19	Phpfreeradius	0.133			0.033*	↓	−21.16
20	Phpmyadmin	0.152			∼0.000	↑	0.23
21	Phpmyfaq	0.709			0.241		
22	Phpqrcode	0.221			N/A	N/A	N/A
23	Simplephpblog	0.119			N/A	N/A	N/A
24	Symfony	0.033	↑	0.02	0.833		
25	Tangocms	0.266			0.634*		
26	Thehostingtool	0.807			0.432		
27	Usebb	∼0.000	↓	−0.86	0.001	↓	−0.83
28	Web2project	0.029*	↓	−29.2	0.134		
29	Wordpress	1.000			∼0.000	↑	1.10
30	Zendframework2	0.001	↓	−0.37	0.761		

* Linear regression has been used for these projects and the Mann–Kendall trend test for the rest to obtain *p*-values.

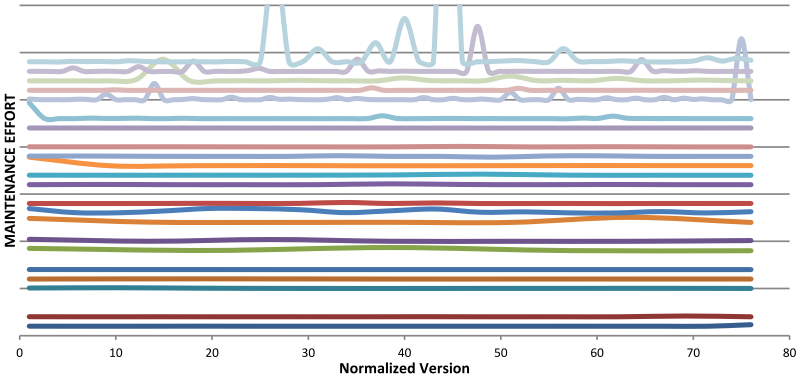

Fig. 9. Evolution of maintenance effort ($V_{4.1}$) for projects with *p*-value > 0.05.

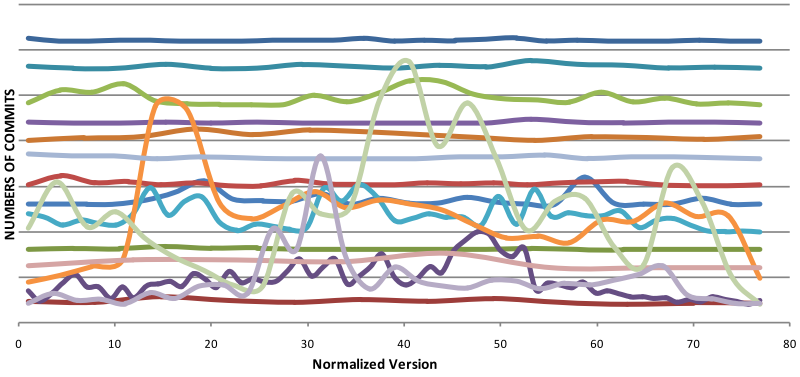

Fig. 10. Evolution of number of commits ($V_{4.2}$) for projects with *p*-value > 0.05.

Table 10
Statistical results on law V (conservation of familiarity).

	Project	Incremental changes				Project	Incremental Changes		
		p-value	Trend	Slope (%)			*p*-value	Trend	Slope (%)
1	Boardsolution	1.000			16	Phpagenda	0.872		
2	Breeze	0.837			17	Phpbeautifier	0.479		
3	Cloudfiles	0.627			18	Phpdaemon	0.754		
4	Codesniffer	0.509			19	Phpfreeradius	1.000		
5	Conference_ci	1.000			20	Phpmyadmin	0.705		
6	Copypastedetector*	0.592			21	Phpmyfaq	0.986		
7	Dotproject	0.917			22	Phpqrcode	0.807		
8	Drupal (core)	0.753			23	Simplephpblog	0.533		
9	Firesoftboard	0.308			24	Symfony	0.592		
10	Generatedata	0.032	↑	0.30	25	Tangocms	0.691		
11	Laravel	0.634			26	Thehostingtool	0.807		
12	Mustache	0.770			27	Usebb*	0.677		
13	Neevo*	0.668			28	Web2project	0.734		
14	Nononsenseforum	0.823			29	Wordpress	0.993		
15	Openclinic	0.348			30	Zendframework2	0.941		

* Linear regression has been used for these projects and the Mann–Kendall trend test for the rest to obtain *p*-values.

turn can be attributed to the increased effort that developers need to understand and modify the program's source code [19].

Hypothesis	Variable	Analysis
H_0: The evolution of incremental changes exhibits no trend. H_1: The evolution of incremental changes exhibits a trend.	[V_5]: Incremental changes (IC) in methods & functions	- Trend test - Slope estimation

Rationale for selected variable: We measured the incremental changes in methods and functions as it captures the potential to provide more and more functionality in each new version. If this is not possible, the release content should be considered invariant.
Concerns: The number of new/modified/deleted functions is only one way of capturing the provision of novel features in a new version.

The results of Table 10 do not allow us to reach a statistically safe conclusion as only in one project a statistically significant trend of IC is evident. For the rest of the projects, trend tests yielded a *p*-value of more than 0.05 implying that we cannot reject the null hypothesis. For this reason, we plotted the actual evolution of these cases in order to visually check the existence of a trend. As it can be observed in Fig. 11, in the majority of the projects, evolution of IC does not exhibit an increasing or a decreasing trend. In other words, the number of additional changes at the method and function level between successive versions might fluctuate temporarily, but is generally invariant over time. This translates to conservation of the release content of each new version in PHP applications which in turn suggests the validity of the 5th law. Thus, we tag this law as *practically* validated.

This law is quite similar to the previous one and the findings also match. However, according to our interpretation, the dimension of time is not taken into account for the 5th law in the sense that the number of incremental changes is not normalized over the elapsed time from the previous release. An increasing trend for the 5th law would imply that the amount of functionality added or modified in each new release is steadily increasing. Such a trend cannot be expected continuously and even if it is present in the initial versions of a new project, it would be unrealistic for mature projects. On the other hand, a decreasing trend would imply that fewer and fewer functions and methods are added or changed over time, signifying a slowly 'dying' project. None of the examined projects exhibits such a trend and it would be worth investigating which kind of actual projects are being gradually abandoned.

4.6. Law VI: continuing growth

The law stipulates that a program grows over time to address the new needs of its clients. Although several measures can be employed to assess this growth, most previous studies have used size metrics such as Lines of Code (LOC) [19] or the number of modules [7]. We have also measured the evolution of LOC to capture both additions of statements within functions as well as additions of new functions and classes (methods). An increasing trend for LOC validates the law. The results concerning the trend test are summarized in Table 11, while Fig. 12 depicts the corresponding trendlines for the majority of the projects where a statistically significant trend has been found.

Hypothesis	Variable	Analysis
H_0: The evolution of system's size exhibits no trend. H_1: The evolution of system's size exhibits a trend.	[V_6]: LOC	- Trend test - Slope estimation

Rationale for selected variable: We examined the evolution of the size of each project in terms of LOC, as did most of the previous studies.
Concerns: -

From the results of Table 11 and the trendlines in Fig. 12, it becomes apparent that in the majority of PHP projects (23/30), the size in terms of LOC increases steadily over time. Although deletions of code also occur, in the examined web applications it is evident that development teams keep adding new code to enhance the offered functionality. As a result we can reach the conclusion that the 6th law of software evolution holds in practice. This law has been confirmed in all previous studies (see Section 5.2 – Comparison with previous work).

4.7. Law VII: declining quality

The law states that the quality of software deteriorates over time unless proactive measures are taken. Degradation of software quality over time is a widely investigated phenomenon known under different names, such as "software ageing" [37] or accumulation of technical debt [38]. A number of internal quality metrics and one external quality indicator have been examined to evaluate the validity of this law for PHP applications. Specifically, we investigated metrics which can be calculated at the level of individual classes and can be associated to an aspect of design quality. Moreover, we included two metrics which concern both functions and methods to assess the quality of non-object-oriented code as well. Finally, we measured the number of bug related commits to assess whether the number of bugs increases or decreases over time. In order to avoid any misleading statistical interpretations, we performed only a trend test on the evolution of each metric but we did not attempt to extract an overall statistical measure

Table 11
Statistical results on law VI (continuing growth).

	Project	LOC				Project	LOC		
		p-value	Trend	Slope (%)			*p*-value	Trend	Slope (%)
1	Boardsolution	0.002	↑	0.19	16	Phpagenda	~0.000	↑	0.56
2	Breeze*	~0.000	↑	0.91	17	Phpbeautifier	~0.000	↑	0.57
3	Cloudfiles	0.001	↑	0.91	18	Phpdaemon	~0.000	↑	5.86
4	Codesniffer	0.256			19	Phpfreeradius	0.001	↑	1.98
5	Conference_ci	0.566			20	Phpmyadmin	~0.000	↑	0.85
6	Copypastedetector	~0.000	↑	2.21	21	Phpmyfaq	~0.000	↑	0.85
7	Dotproject	~0.000	↑	1.59	22	Phpqrcode*	0.012	↑	12.90
8	Drupal (core)	~0.000	↑	1.63	23	Simplephpblog	0.837		
9	Firesoftboard	0.807			24	Symfony	~0.000	↑	1.08
10	Generatedata	~0.000	↑	0.39	25	Tangocms	0.051		
11	Laravel	~0.000	↑	2.60	26	Thehostingtool	0.024	↑	3.40
12	Mustache	~0.000	↑	2.86	27	Usebb	~0.000	↑	1.87
13	Neevo	0.005	↑	1.19	28	Web2project	0.807		
14	Nononsenseforum	~0.000	↑	2.99	29	Wordpress	~0.000	↑	1.27
15	Openclinic	0.003	↑	1.76	30	Zendframework2	0.293		

* Linear regression has been used for these projects and the Mann–Kendall trend test for the rest to obtain p-values.

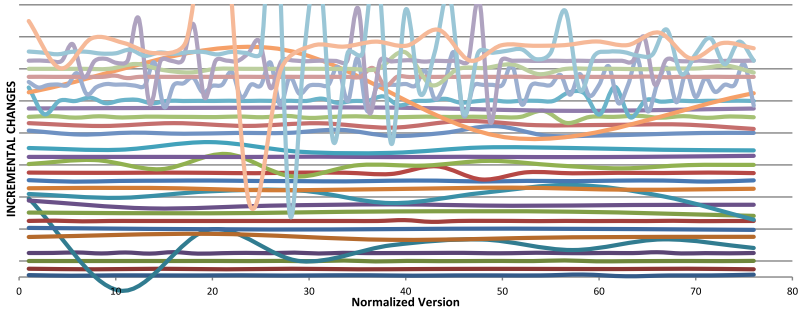

Fig. 11. Evolution of incremental changes for projects with p-value > 0.05.

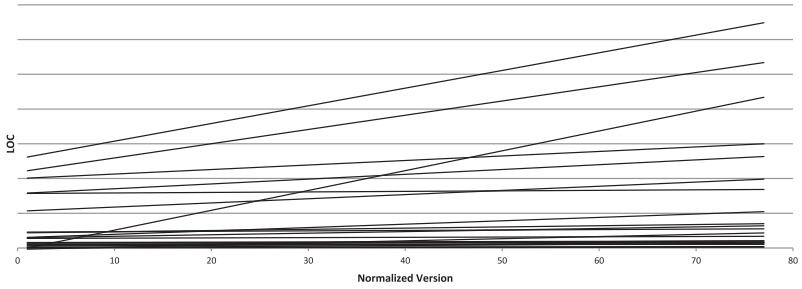

Fig. 12. Trendlines of LOC for projects with p-value < 0.05.

considering all metrics. A brief discussion of the employed metrics follows next.

Coupling is one of the classic internal metrics used to assess the quality of a design and for this reason we measured the average Afferent Coupling (CA) and the average Efferent Coupling (CE) of each class. Afferent coupling refers to the number of unique incoming dependencies for a software artifact (i.e. it is representative of a class' fan-in). Therefore, it is an indicator of the extent by which a module is used by other modules, and under normal circumstances, it is suggested to keep the fan-in high [39]. Typical examples of modules/packages with high fan-in are core packages and components, like error and exception handling, or unit testing framework classes.

Efferent coupling counts the number of software artifacts that a software entity depends on. A high efferent coupling (i.e. the module has a high fan-out) implies that the component depends on several other implementation details and this makes the component itself instable, because an incompatible change between two versions or a switch to a different library may break the dependent component. Moreover, the comprehensibility and reusability of a module with high efferent coupling is limited. Therefore it is considered a good practice to keep the efferent coupling for all artifacts at a minimum [39].

The quality of an object-oriented design has also been assessed from the perspective of inheritance qualities. Although specific thresholds for the optimum depth of an hierarchy are hard to extract by means of empirical studies, Harrison and Counsell [40] have found that deeper inheritance trees are harder to understand and maintain, a view shared also in the early discussions on inheritance heuristics by Riel [41]. In our study we tracked the evolution of the 'Depth of Inheritance Tree' metric (DIT) as PHP systems evolve.

Several studies assess the understandability of code (which is a sub-characteristic of maintainability) by the comment ratio (CR) that is the ratio of commented lines of code over the total lines of code. The higher the ratio for a piece of code is, the more readable and thus maintainable the code can be considered to be [42]. This metric allows us to assess the evolution of both function and methods and has been selected as the fourth internal quality indicator.

Another widely used and discussed measure of quality is the Maintainability Index (MI) which has been originally introduced by Oman and Hagemeister in 1991 [43]. MI is a composite metric that considers for an assessed module its Halstead's volume, cyclomatic complexity and size in terms of lines of code. There have been numerous studies on the validity of MI, some of which have found that MI can successfully predict actual maintenance effort and others which have questioned its accuracy. Nevertheless, in this study we have used MI as an indicator of internal quality because it is not restricted to object-oriented code, and because that regardless of its accuracy as a maintainability predictor, an increasing trend of MI would imply efforts to improve three aspects of quality within functions or methods.

Finally, since all the aforementioned metrics focus on internal quality we have also included a measure that aims at addressing quality as perceived by users or developers testing the functionality of the system. An indisputable indicator of external quality would be the number of bugs/errors found during system evolution, as an increasing number of bugs implies quality degradation. However, although the examined applications are supported by an issue tracking system, for the examined PHP projects, we found that it would be unreliable to count the number of issues (since in numerous cases the reported issues do not concern bugs). For this reason we have opted for the number of commits (i.e. actual code changes) for which we could infer that they are related to the fixing of a bug or issue. As in other studies (e.g. [44]) we identified bug related commits by filtering those that contain error related keywords, such as 'error', 'bug', 'fix' and 'issue' in the corresponding commit message.

For measures CA, CR and MI an increasing trend implies that quality is improving from this perspective. On the other hand, for measures CE, DIT and number of bug related commits, quality is improving if their values get lower. In Table 12 we report the trend of the aforementioned quality measures over all examined versions for each project. To facilitate the interpretation of the results, we have marked with shaded cells the cases in which the evolution of a metric suggests deterioration of the system's quality.

Hypothesis	Variable	Analysis
H_0: The evolution of system's quality exhibits no trend. H_1: The evolution of system's quality exhibits a trend.	[$V_{7.1}$]: CA [$V_{7.2}$]: CE [$V_{7.3}$]: DIT [$V_{7.4}$] CR [$V_{7.5}$] Maintainability Index (MI) [$V_{7.6}$] Number of bug-related commits	- Trend test - Slope estimation

Rationale for selected variables: The assessment of quality evolution is based on a mixture of internal quality metrics (for object-oriented and procedural code) and one external quality indicator related to the number of bugs. The selected metrics have been tested for correlation among them, as explained in Section 3.4

Concerns: Internal quality metrics do not necessarily map to external quality. The number of bug-related fixes is sensitive on the style of commit messages employed in a project.

As it can be observed from the number of projects in which a statistically significant trend has been found, the overall picture is rather mixed across the examined quality indicators. For afferent coupling quality is increasing in 10 out of the 14 projects and for the maintainability index quality is increasing in 15 out of the 23 projects with a statistically significant trend. Quality is decreasing in 12 out of 16 projects for efferent coupling and in 12 out of 18 projects for the depth of inheritance. In terms of comment ratio in about half of the 21 projects quality is increasing and for the rest quality decreases. For bug related commits, a trend was found only in 8 out of the 20 projects.

The picture is mixed even if tables is analyzed horizontally that is, by examining each project separately to identify how often the quality of a project deteriorates or improves over time. Thus, there is no supporting evidence neither for the confirmation nor for the confutation of the 7th law. In other words, it cannot be claimed in general that the quality of the examined PHP projects is declining or improving over time.

4.8. Law VIII: feedback system

The corresponding claim was stated in 1980 but has been formalized as a law in 1996 [7]. According to Lehman [34], the evolution process of software resembles a feedback system. In other words, the size of a software system in a given release can be described in terms of the size in the previous release and the effort for developing the new release. Turski [31] formulated a model suggesting that the growth of a system, in terms of number of changed modules, is sub-linear, slowing down during the evolution of the project, exactly because the system becomes larger and more complex. The number of modules is preferred over low-level measures such as LOC since according to Turski system functionality changes are reflected in added, removed or otherwise handled modules, a view shared by Lehman in his early studies [13]. Turski proposed a difference equation according to which the size of version i can be estimated as:

$$S_i = S_{i-1} + \frac{\overline{E}}{S_{i-1}^2} \tag{1}$$

where (interpretation is fitted to the case of PHP applications):

S_i is the size of version i measured in number of methods and functions and,
$\overline{E}$ is the effort spent on the development of each software release, which is considered constant according to the fourth law of Lehman.

The intuition behind this formulation is that the larger the size of a version, the greater the resistance to change it, in analogy to the effect of mass in a mechanical system or capacity in an electrical system.

Later, Turski generalized the model to a differential form [45] and extracted a closed form for the growth equation as:

$$S(t) = a \cdot t^{\frac{1}{3}} + b \tag{2}$$

where α and b are constants.

By obtaining the derivative of the growth equation, the corresponding rate of growth is:

$$\frac{dS}{dt} = c \cdot t^{-\frac{2}{3}} \tag{3}$$

where

c is a constant,
and t is the elapsed time (in days) from the initial release.

If the law holds in practice, the rate of growth should be proportional to $t^{-\frac{2}{3}}$, so it is relatively straightforward to check its validity. The actual evolution of $\Delta S/\Delta t$ for all successive release pairs, can be compared to the theoretical evolution by employing the two-sample Kolmogorov–Smirnoff test [46].

As an example let us consider the evolution of the growth rate for project *mustache* (Fig. 13). The solid line represents the observed changes in the growth rate ($\Delta S/\Delta t$), while the dashed line corresponds to the evolution predicted by Lehman's 8th law according to Turski's model. As it becomes evident the actual $\Delta S/\Delta t$ trend line is well above the rate predicted by the law and the

Table 12
Statistical results on law VII (declining quality).

	Project	CA			CE			DIT		
		p-value	Trend	Slope (%)	*p*-value	Trend	Slope (%)	*p*-value	Trend	Slope (%)
1	Boardsolution	0.003	↑	0.08	0.022	↑	0.07	0.067		
2	Breeze	0.128			0.969			∼0.000	↑	0.35
3	Cloudfiles	0.260			0.260			0.014	↑	0.05
4	Codesniffer	0.096			0.185			∼0.000	↓	−6.29
5	Conference_ci	0.105			0.411			0.105		
6	Copypastedetector	0.885			0.017	↑	0.59	∼0.000	↑	1.60
7	Dotproject	0.105			0.358			0.006	↑	1.66
8	Drupal (core)	1.000			0.207			∼0.000	↑	0.96
9	Firesoftboard	0.613			0.129			1.000		
10	Generatedata	0.012	↓	−0.12	0.024	↓	−0.14	0.012	↑	0.25
11	Laravel	∼0.000	↓	−0.67	0.008	↓	−0.43	∼0.000	↑	2.00
12	Mustache	∼0.000	↑	2.62	∼0.000	↑	2.49	∼0.000	↓	−1.02
13	Neevo	1.000			0.009	↑	1.43	0.001	↑	0.34
14	Nononsenseforum	∼0.000	↑	5.55	∼0.000	↑	5.00	∼0.000	↓	−3.94
15	Openclinic	0.021	↑	2.98	0.001	↑	7.14	0.165		
16	Phpagenda	∼0.000	↓	−1.21	∼0.000	↓	−1.08	∼0.000	↓	−0.90
17	Phpbeautifier	∼0.000	↑	1.49	0.823			0.148		
18	Phpdaemon	0.088			0.059			0.009	↑	4.98
19	Phpfreeradius	0.421			0.789			0.421		
20	Phpmyadmin	0.004	↑	0.08	0.475			∼0.000	↑	0.81
21	Phpmyfaq	0.359			0.045	↓	−0.13	0.005	↓	−0.16
22	Phpqrcode	1.000			0.008	↑	6.91	1.000		
23	Simplephpblog	0.453			0.015	↑	14.00	0.078		
24	Symfony	∼0.000	↑	0.10	∼0.000	↑	0.07	∼0.000	↑	0.13
25	Tangocms	0.021	↓	−0.05	0.006	↑	0.04	0.498		
26	Thehostingtool	∼0.000	↑	3.81	0.181			0.100		
27	Usebb	1.000			1.000			1.000		
28	Web2project	0.267			0.267			0.149		
29	Wordpress	∼0.000	↑	0.68	∼0.000	↑	0.52	∼0.000	↑	1.27
30	Zendframework2	∼0.000	↑	0.21	∼0.000	↑	0.40	∼0.000	↓	−0.18
	Project	CR			MI			BUG COMMITS		
		p-value	Trend	Slope (%)	*p*-value	Trend	Slope (%)	*p*-value	Trend	Slope (%)
1	Boardsolution	0.018	↑	0.01	∼0.000*	↑	1.76	0.529		
2	Breeze	∼0.000	↓	−0.08	∼0.000	↓	−0.95	N/A	N/A	N/A
3	Cloudfiles	0.358			∼0.000	↓	−0.15	0.064*		
4	Codesniffer	0.019	↓	−0.07	0.502			∼0.000	↑	1.38
5	Conference_ci	0.848			∼0.000*	↑	0.12	0.691		
6	Copypastedetector	0.888			0.772			0.117		
7	Dotproject	0.032	↓	−0.22	N/A	N/A	N/A	0.678		
8	Drupal (core)	∼0.000	↑	1.17	0.186			∼0.000	↑	0.38
9	Firesoftboard	0.807			∼0.000*	↑	1.30	0.945		
10	Generatedata	0.008	↓	−0.28	0.024	↑	1.06	0.002*	↓	−11.8
11	Laravel	∼0.000	↓	−0.6	0.044	↑	0.09	0.008	↓	−3.34
12	Mustache	0.466			0.025	↑	1.07	0.591		
13	Neevo	0.005	↓	−0.44	∼0.000*	↑	1.9	0.212*		
14	Nononsenseforum	∼0.000	↑	1.12	∼0.000	↑	3.81	1.000		
15	Openclinic	0.243			∼0.000*	↓	−0.16	N/A	N/A	N/A
16	Phpagenda	∼0.000	↑	0.32	1.000			N/A	N/A	N/A
17	Phpbeautifier	0.002	↓	−0.2	0.115			0.066		
18	Phpdaemon	0.127			0.001	↑	5.98	0.212		
19	Phpfreeradius	0.004	↓	−0.21	0.035	↓	−3.48	N/A	N/A	N/A
20	Phpmyadmin	0.013	↓	−0.06	0.026	↑	0.03	∼0.000	↑	0.86
21	Phpmyfaq	∼0.000	↓	−0.5	∼0.000	↓	−0.15	0.446		
22	Phpqrcode	0.085			0.011*	↓	−18.9	N/A	N/A	N/A
23	Simplephpblog	0.002	↑	2.44	∼0.000*	↑	10.01	N/A	N/A	N/A
24	Symfony	0.003	↑	0.02	∼0.000	↑	0.12	∼0.000	↑	4.18
25	Tangocms	∼0.000	↓	−0.08	∼0.000*	↑	0.15	0.837*		
26	thehostingtool	∼0.000*	↑	1.53	∼0.000*	↓	−6.38	0.065		
27	Usebb	0.009	↑	0.36	0.022	↓	−0.23	0.003	↓	−1.17
28	Web2project	∼0.000*	↑	1.66	0.051*			0.155		
29	Wordpress	∼0.000	↑	0.84	∼0.000	↑	1.54	∼0.000	↑	0.83
30	Zendframework2	0.441			0.003	↑	0.12	0.112		

* Linear regression has been used for these projects and the Mann-Kendall trend test for the rest to obtain *p*-values.

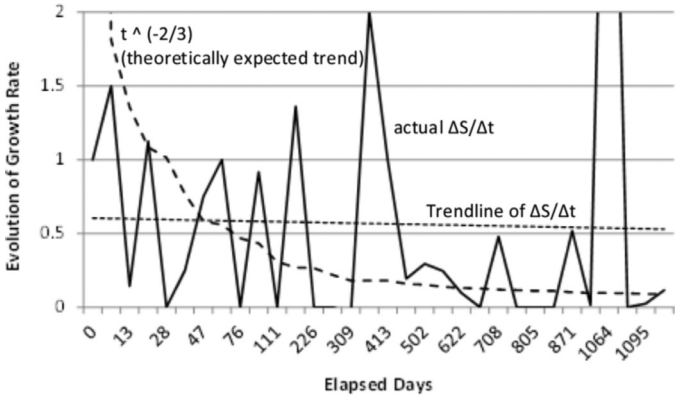

Fig. 13. Examination of the validity of the *8*th law in project "*mustache*".

Table 13
Statistical results on law VIII (feedback system).

	Project	Kolmogorov–Smirnoff *p*-value		Project	Kolmogorov–Smirnoff *p*-value
1	Boardsolution	0.541	16	Phpagenda	0.000
2	Breeze	0.006	17	Phpbeautifier	0.206
3	Cloudfiles	0.249	18	Phpdaemon	0.000
4	Codesniffer	0.000	19	Phpfreeradius	0.203
5	Conference_ci	0.082	20	Phpmyadmin	0.000
6	Copypastedetector	0.001	21	Phpmyfaq	0.000
7	Dotproject	0.002	22	Phpqrcode	0.329
8	Drupal (core)	0.000	23	Simplephpblog	0.023
9	Firesoftboard	0.699	24	Symfony	0.000
10	Generatedata	0.001	25	Tangocms	0.003
11	Laravel	0.007	26	Thehostingtool	0.819
12	Mustache	0.002	27	Usebb	0.000
13	Neevo	0.100	28	Web2project	0.211
14	Nononsenseforum	0.000	29	Wordpress	0.000
15	Openclinic	0.699	30	Zendframework2	0.000

growth rate is not declining as predicted. For this case we can conclude (by visual examination) that the law is not confirmed for this particular project.

Hypothesis	Variable	Analysis
H_0: The empirically observed rate of growth matches the theoretically expected one. H_1: The empirically observed rate of growth does not match the theoretically expected one.	$[V_8]$: rate of growth	Two sample Kolmogorov–Smirnoff test

Rationale for selected variable: We examined the evolution of the rate of growth of each project and compared it with the theoretical one as proposed by Turski and shared by Lehman.
Concerns: The primary concern here is the interpretation of the notion of feedback system. In this study we adopt the mathematical interpretation provided by Turski [31]

The results from the statistical investigation of the validity of the *8*th law are presented in Table 13 listing the significance value of the Kolmogorov–Smirnoff test conducted for each project in order to examine whether the actual growth rate ($\Delta S/\Delta t$) matches the theoretically expected rate. A significance value less than 0.05, means that the null hypothesis can be rejected, implying that the law is not confirmed (the corresponding cases are shaded in the Table).

The growth rate does not match the theoretical expectation in 19 out of 30 projects as marked by the shaded rows in Table 13. Thus, one could argue that the law is not confirmed by our results for the examined PHP applications. In other words, the rate of increase in project size indeed attenuates over time, however, not at the fast rate predicted by Turski's model. It should be noted that the outcome for this law is not in contrast to the findings for the 5th law and 6th law. The results for Law V suggested that we cannot claim that more and more (or less and less) code (incremental changes) is practically added in successive versions, without however considering the time elapsed between releases, whereas the results for Law VI confirmed that systems continuously grow. The findings for this law, which assumes that software processes operate as a feedback system where current size dictates the rate of increase in the next release, suggest that the growth rate is attenuating, i.e. that if time is taken into account, less code is added in a given amount of time. In other words, as the examined applications mature either there is less left to be added in terms of functionality or the system size prevents the development team from keeping the same pace of adding new code. Nevertheless, system development is slowing down at a rather low rate.

5. Overview and comparison to previous work

5.1. Summary of results

To facilitate the interpretation of the findings regarding the eight laws of Lehman, we summarize in Table 14 the corresponding claims and contrast them to the results for the examined PHP applications. The laws are grouped in three categories based on the generic aspect/property that they address. As it can observed, from the two laws (II & VII) concerning the evolution of quality the 2nd has not been confirmed for the examined PHP applications while for the 7th law the results were inconclusive. With respect to the laws discussing changes in an evolving system (I, IV & V) we found that all laws are confirmed (the 1st with statistical significance while the other two only at a practical level). In other words systems continuously undergo changes but no trend has been observed for the work rate or the incremental changes. As a general observation one could claim that the examined PHP applications are maintained without reaching any maintenance stagnation.

Finally, with respect to the laws that address the growth of an evolving system (III, VI & VIII), systems indeed continuously grow and exhibit positive and negative adjustments of

Table 14
Summary of findings about Lehman's laws.

Property	Law	Lehman claims:	Our finding (PHP)
Quality	II	*Complexity increases*	Complexity does not increase
	VII	*Quality declines*	Inconclusive results
Changes	I	*System continuously change*	Indeed
	IV	*Work rate remains stable*	Indeed (no statistical significance)
	V	*Incremental changes remain invariant*	Indeed (no statistical significance)
Growth	III	*Incremental growth exhibits negative and positive adjustments (systems are self-regulated)*	Indeed (no statistical significance)
	VI	*Systems continuously grow*	Indeed
	VIII (Turski's form)	*Growth rate decreases at a rate proportional to* $t^{-2/3}$	Growth rate does not decrease that fast

incremental growth. However, we could not confirm that the growth rate decreases according to the theoretically prescribed rate. In other words the examined PHP applications do get bigger, are maintained and there are no clear signs of quality degradation or improvement. Further research into the reasons that drive this evolution patterns of PHP web applications would be extremely valuable.

The present study has not been designed to identify the reasons for which certain laws are confirmed for some projects while others are violated. Nevertheless, we will attempt to provide an explanation, noting that it is not based on hard evidence. It is reasonable to assume that the reason for which PHP web applications continuously change and grow is to provide novel services and features to clients in the shortest time possible. This is a necessity in order to withstand the competition caused by the perpetuous outspread of the Web. Such a competitive environment is normally driving the accumulation of the so-called 'technical debt' [47]. In other words, speeding-up development time normally compromises software quality, thereby hindering its sustainability. However, this accumulation of technical debt is not evident for PHP web applications which manage to evolve without increasing their complexity and without demanding increased effort. We postulate that this phenomenon is due to the productivity of the language, which allows developers to rapidly produce functional code, and to the widespread usage of reliable libraries and frameworks.

5.2. Comparison to previous work

An overview of the approach and the findings regarding the validity of the eight laws of Lehman in previous research is provided in Tables 15 and 16, along with the results in this study. To provide insight into the approach that has been employed by each research group for the quantification of the examined laws, Table 15 briefly outlines the corresponding measures used in 8 previous studies. (When a law is not investigated in the context of a work, the corresponding cell is left blank). Because of the way that the laws have been stated, as it can be observed from Table 15, the employed measures vary. However, there are laws which are quantified by most of the studies in the same or in a similar manner. For example law VI is quantified by most of the studies using the LOC metric, and Law III is quantified mainly through the number of functions. On the other hand, law VII, which does not specify which aspect of quality has to be considered, is quantified through a variety of quality indicators.

To allow a comparison with the conclusions derived in other studies about Lehman's laws (which however have not focused on PHP web applications), Table 16 lists the findings from the aforementioned 8 previous studies. A '✔' symbol indicates confirmation, a '×' symbol indicates that the law has not been validated, while the '∼' symbol implies that the results have been inconclusive. When a law is not investigated in the context of a work, the corresponding cell is left blank. It should be noted that in this Table we list the conclusions as derived by the authors of the corresponding papers (for the studies by Godfrey & Tu [15] and by Robles et al. [17] the validity of the 1st, 6th and 8th law is not directly investigated but can be easily deduced from the provided information).

As it can be observed, the 1st law regarding continuing change and the related 6th law on continuing growth are, as expected, validated by all studies. In some studies system growth rate (in LOC) is found to be exponential [15] while in others linear [17]. In other words, all studies agree that systems continuously change and grow (a phenomenon called 'perpetual development' in the study by Israeli and Feitelson [20]). An agreement is also observed between previous studies and the current one for the 2nd and 8th law. Concerning increasing complexity, in 3 out of the 5 previous works that examined this law and reached conclusive results, it had not been confirmed, as in the case of PHP projects. Concerning the decline of growth rate at the pace predicted by the 8th law, four previous studies (out of the five that reached conclusive results for C/C++/Java projects) found that the actual growth rate attenuates at a slower pace, as it has also been found in this study for PHP projects.

6. Implications for researchers and practitioners

Although the research question that has been set, regarding the validity of Lehman's laws of evolution for PHP web applications, entails a theoretical perspective and thus the results are not directly exploitable, we can identify the following implications.

With respect to software practitioners and managers:

- In the context of the investigation of Lehman's laws of evolution the employed measures can be used to assess the evolution of other products and examine whether any striking deviations from Lehman's observations are valid for their projects. Since most laws are not directly quantifiable, software maintainers could employ the same methodology with respect to the applied trend tests and indicators that have been analyzed for each law.
- Especially with respect to the evolution of quality vs. the increase of size contrasting the results for their own projects to those of the examined applications could highlight issues that warrant attention. For example, it should be regarded as a warning if their own PHP web projects do not success in allowing continuous changes combined with a non-increasing complexity, since this trend has been observed both for small and large open-source projects in this study. If, for example, a development team observes that complexity is constantly increasing, whereas large and complicated PHP systems manage to keep complexity stable or even reduce it over time, then, quality assurance should focus on ways to address the increasing complexity.
- The results suggesting that PHP web applications conform to a lifecycle model where continuous and steady development takes places (a finding confirmed by other studies as well), imply that development teams should opt for agile development practices, where constant change is embraced, rather than models assuming elaborate and preconceived specifications and planning [20].
- The results indicating that PHP web applications continuously change and grow, a finding shared by all other studies as well, imply that project managers should anticipate increased future needs for resources to maintain and sustain the existing systems.

With respect to software engineering researchers:

- Based on the findings indicating that PHP web applications do not suffer from software ageing, researchers can focus on the reasons that drive this improved behavior of PHP projects and investigate whether this is due to the language, the domain or the practices in web application development.
- Researchers are encouraged to investigate whether the same trends are valid for the evolution of systems written in other scripting languages so as to investigate whether similar maintenance patterns can be attributed to the nature of the employed languages (i.e. scripting vs. compiled).
- Finally, for the specific group of research efforts that investigate the validity of Lehman's laws, empirical findings that suggest that: a) several laws are consistently not confirmed (e.g. Law VIII), or that b) some laws occasionally lead to inconclusive results (e.g. Laws IV and VII) or that c) some laws are

Table 15
Primary measures employed for the investigation of laws in previous studies.

Ref.	I	II	III	IV	V	VI	VII	VIII
Godfrey & Tu*	SLOC			SLOC		SLOC		SLOC
Robles et al.	SLOC			SLOC		SLOC		SLOC
Mens et al.	File changes	LOC, additions/modifications, #defects, CC				Several size measures including LOC		
Xie et al.	Cumulative #changes, type of changes	CC, function calls, coupling	# functions	Changes per day, handled functions/total functions	#modules, new functions	LOC, #functions, #definitions	#defects, defect density, complexity measures	#functions
Israeli & Feitelson	#Source files	CC	#files	Percentage of handled files	Releases per month, intervals between releases	#system calls, #configuration options	Maintainability Index	No quantitative approach
Businge et al.	Cumulative number of added/deleted dependencies		#dependencies		Percentage of handled files, percentage of added dependencies	unique dependencies	Indicator of balance between abstractness and stability	
Neamtiu et al.	Cumulative changes	Calls per function CC coupling	#modules #functions	Changes per day change rate growth rate	Net module growth #new functions #changes	LOC #modules #definitions	#defects defect density calls per function CC coupling	#modules LOC #functions
Kaur et al.	#functions and #classes	CBO, RFC, WMC, DIT, LOCH	#functions and #classes	No quantitative approach	#functions and #classes	LOC, #functions and #classes	CC	No quantitative approach
This study	Days between releases	CC	#functions	Maintenance effort and #commits	#functions	LOC	CA, CE, DIT, CR, MI, bug-related commits	#functions

*CC: cyclomatic complexity.
*SLOC: source lines of code (uncommented lines of code).
*CBO: coupling between objects.
*RFC: Response for class - #methods being invoked in response to the message received by an object of that class.
*WMC: weighted methods per class - the sum of the complexities of its methods.
*DIT: depth of inheritance tree.
*LOCH: lack of cohesion.
*CA: coupling afferent (#unique incoming dependencies for a software artifact).
*CE: coupling efferent (#unique outgoing dependencies for a software artifact).
*CR: comment ratio.
*MI: Maintainability Index.

Table 16
Validity of Lehman's laws according to various studies.

Ref.	Year	Prog.Lang.	#Projects	I	II	III	IV	V	VI	VII	VIII
Godfrey & Tu*	2000	C	1	✔			×		✔		×
Robles et al.	2005	C,C++, Java	19	✔			×		✔		×
Mens et al.	2008	Java	1	✔	×				✔		
Xie et al.	2009	C	7	✔	✔	✔	~	×	✔	×	×
Israeli & Feitelson	2010	C	1	✔	×	✔	✔	~	✔	×	✔
Businge et al.	2010	Java	21	✔		✔		×	✔	~	
Neamtiu et al.	2013	C	9	✔	×	×	×	×	✔	×	×
Kaur et al.	2014	C++	2	✔	✔	✔	~	✔	✔	✔	~
This study	2015	PHP	30	✔	×	✔**	✔**	✔**	✔	~	×

* The results in a later work by Godfrey & Tu [16] confirmed the validity of the same laws on 4 projects.
** These laws have not been statistically validated. The conclusion in these cases is based on a visual interpretation of the evolution for the projects where the null hypothesis (absence of trend) could not be rejected.

quantified by divergent approaches (e.g. Law IV), imply that the rules might need to be examined in the context of contemporary software development and possible be revisited.

7. Threats to validity

The investigation of the validity of Lehman's laws is by definition threatened by the subjectivity in the interpretation of each law and the selection of appropriate metrics to quantify its evolution. The fact that the employed measures might not reflect accurately the phenomenon under investigation poses a threat to the relation between theory and observation, i.e. to construct validity [48]. In addition, for several laws there might be additional measures that can be used to quantify the corresponding evolutionary trend, which are either not available (such as the effort spent in an open-source project) or unreliable if collected automatically (such as the number of issues). For example, law VII on the evolution of quality, can be quantified by numerous internal and external quality indicators, as it becomes evident from the multitude of metrics employed by previous studies shown in Table 15. To mitigate this threat, for most of the laws we relied on measures that have been used in previous studies as well. Moreover, to emphasize this inherent limitation in the quantification approach we explicitly stated the relevant concerns along with the approach for each law.

The conclusions derived from any empirical study that is based on a set of examined software systems are subject to external validity threats. In our case, this threat limits our possibility to

generalize our findings regarding the validity of Lehman's laws in PHP applications beyond the 30 examined projects and to other programming languages. In other words it is not granted that the selected projects are representative of the entire PHP web application landscape. As it is always the case, further replication studies would be extremely valuable. The emphasis on PHP was placed on purpose, since the goal of this study was to investigate patterns of evolution in web applications built upon a scripting language. In this regard, further studies could extend the analysis to other primarily scripting languages such as Python, Perl and Ruby.

Finally, since the presented empirical study relies heavily on the interpretation of statistical test results (mainly trend tests) threats to statistical conclusion validity may arise. The conclusions about the identified trends are based on the number of projects that exhibited statistically significant trends. For example, in the 2nd law we consider that the normalized complexity exhibits a trend because a decreasing trend has been observed in 12 out of the 18 projects with a statistically significant result. Such a finding might imply low statistical power. In other words, although the trend test for each project is correctly applied by analyzing the relevant assumptions, one has to aggregate the findings for all projects to reason about the validity of the law. To facilitate the interpretation of the results we have provided all data which have led to the confirmation of confutation of each law.

8. Conclusions

The evolution of software projects relying on scripting languages such as PHP has received limited attention, despite the fact that PHP forms the basis upon which a huge number of web applications are developed. Driven by the widely spread but undocumented claims that scripting languages are not suitable for regularly maintained software projects we have performed an empirical study on the evolution of 30 PHP web applications.

The main goal was to examine the validity of the eight laws of software evolution as stated by M. M. Lehman. These laws have been extensively studied in the context of software evolution for projects developed in compiled languages such as C and C++ and in an non-web related context. The results confirm the validity of continuing growth and changes for the evolution of the examined PHP applications. However, for the examined projects we have not confirmed the 2nd law on increasing complexity and the 8th law on the rapid decrease of the growth rate. Although the root causes for this trend require further investigation it is reasonable to assume that this phenomenon could be attributed either to the programming language or to the practices in web application development.

One interesting line of further research would be to compare the evolution of web applications against that of "conventional" desktop systems, in order to investigate whether there are differences in the trends of quality, work rate, complexity and size. Such evidence would be helpful in determining whether development practices for web applications adhere to the principles of building large-scale, multi-person, multi-version software systems or whether the benefits is the result of their architecture, which is often strictly dictated by the platforms being used.

References

[1] R.P. Loui, In praise of scripting: real programming pragmatism, Computer 41 (7) (Jul. 2008) 22–26.

[2] L. Prechelt, Are scripting languages any good? A Validation Of Perl, Python, Rexx, And Tcl Against C, C++, and Java, Advances in Computers, 57, Elsevier, 2003, pp. 205–270.

[3] J.K. Ousterhout, Scripting: higher level programming for the 21st Century, Computer 31 (3) (Mar. 1998) 23–30.

[4] S. Hanenberg, S. Kleinschmager, R. Robbes, É. Tanter, A. Stefik, An empirical study on the impact of static typing on software maintainability, Empir. Softw. Eng. 19 (5) (Dec. 2013) 1335–1382.

[5] "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities." [Online]. Available: http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext. [Accessed: 15-Mar-2015].

[6] P. Kyriakakis, A. Chatzigeorgiou, Maintenance patterns of large-scale PHP Web Applications, in: Proceedings of 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2014, pp. 381–390.

[7] M.M. Lehman, Laws of software evolution revisited, in: C. Montangero (Ed.), Software Process Technology, Springer, Berlin Heidelberg, 1996, pp. 108–124.

[8] H. Kagdi, M.L. Collard, J.I. Maletic, A survey and taxonomy of approaches for mining software repositories in the context of software evolution, J. Softw. Maint. Evol. Res. Pract. 19 (2) (Mar. 2007) 77–131.

[9] M.W. Godfrey, D.M. German, The past, present, and future of software evolution, in: Proceedings of Frontiers of Software Maintenance, 2008. FoSM 2008, 2008, pp. 129–138.

[10] M.M. Lehman, Programs, Cities, Students: Limits To Growth?, Imperial College of Science and Technology, University of London, 1974.

[11] M. Lehman, Laws of program evolution-rules and tools for programming management, in: Proceedings Infotech State of the Art Conference, Why Software Projects Fail?, 1978, pp. 11/1–11/25.

[12] M.M. Lehman, Programs, life cycles, and laws of software evolution, Proc. IEEE 68 (9) (Sep. 1980) 1060–1076.

[13] N.H. Madhavji, J. Fernandez-Ramil, D. Perry, Software Evolution and Feedback: Theory and Practice, John Wiley & Sons, 2006.

[14] I. Herraiz, D. Rodriguez, G. Robles, J.M. Gonzalez-Barahona, The evolution of the laws of software evolution: a discussion based on a systematic literature review, ACM Comput. Surv. 46 (2) (Dec. 2013) 28:1–28:28.

[15] M.W. Godfrey, Q. Tu, Evolution in open source software: a case study, in: Proceedings of the International Conference on Software Maintenance (ICSM'00), Washington, DC, USA, 2000, p. 131.

[16] M. Godfrey, Q. Tu, Growth, evolution, and structural change in open source software, in: Proceedings of the 4th International Workshop On Principles Of Software Evolution, New York, NY, USA, 2001, pp. 103–106.

[17] G. Robles, J.J. Amor, J.M. Gonzalez-Barahona, I. Herraiz, Evolution and growth in large libre software projects, in: Proceedings of Eighth International Workshop on Principles of Software Evolution, 2005, pp. 165–174.

[18] T. Mens, J. Fernandez-Ramil, S. Degrandsart, The evolution of Eclipse, in: Proceedings of IEEE International Conference on Software Maintenance, 2008. ICSM 2008, 2008, pp. 386–395.

[19] G. Xie, J. Chen, I. Neamtiu, Towards a better understanding of software evolution: an empirical study on open source software, in: Proceedings of IEEE International Conference on Software Maintenance, ICSM 2009, 2009, pp. 51–60.

[20] A. Israeli, D.G. Feitelson, The Linux kernel as a case study in software evolution, J. Syst. Softw. 83 (3) (Mar. 2010) 485–501.

[21] J. Businge, A. Serebrenik, M. van den Brand, An empirical study of the evolution of eclipse third-party plug-ins, in: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), New York, NY, USA, 2010, pp. 63–72.

[22] I. Neamtiu, G. Xie, J. Chen, Towards a better understanding of software evolution: an empirical study on open-source software, J. Softw. Evol. Process 25 (3) (Mar. 2013) 193–218.

[23] T. Kaur, N. Ratti, P. Kaur, Applicability of lehman laws on open source evolution: a case study, Int. J. Comput. Appl. 93 (18) (May 2014) 40–46.

[24] V.R. Basili, Software Modeling and Measurement: The Goal/Question/Metric Paradigm, University of Maryland at College Park, College Park, MD, USA, 1992.

[25] J. Fernandez-Ramil, A. Lozano, M. Wermelinger, A. Capiluppi, Empirical studies of open source evolution, Software Evolution, Springer, Berlin Heidelberg, 2008, pp. 263–288.

[26] H.B. Mann, Nonparametric tests against trend, Econometrica 13 (3) (Jul. 1945) 245–259.

[27] J. Durbin, G.S. Watson, Testing for serial correlation in least squares regression: I, Biometrika 37 (3/4) (Dec. 1950) 409–428.

[28] T.S. Breusch, A.R. Pagan, A simple test for heteroscedasticity and random coefficient variation, Econometrica 47 (5) (Sep. 1979) 1287–1294.

[29] S.S. Shapiro, M.B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52 (3/4) (Dec. 1965) 591–611.

[30] H. Theil, A rank-invariant method of linear and polynomial regression analysis, in: B. Raj, J. Koerts (Eds.), Henri Theil's Contributions to Economics and Econometrics, Springer, Netherlands, 1992, pp. 345–381.

[31] W.M. Turski, Reference model for smooth growth of software systems, IEEE Trans. Softw. Eng. 22 (8) (Aug. 1996) 599–600.

[32] I. Sommerville, Software Engineering, 9 ed., Addison-Wesley, Boston, 2010.

[33] T.J. McCabe, A complexity measure, IEEE Trans. Softw. Eng. SE-2 (4) (Dec. 1976) 308–320.

[34] M.M. Lehman, Software's future: managing evolution, IEEE Softw. 15 (1) (Jan. 1998) 40–44.

[35] M.M. Lehman, D.E. Perry, J.F. Ramil, On evidence supporting the FEAST hypothesis and the laws of software evolution, in: Proceedings of the Fifth International Software Metrics Symposium. Metrics 1998., 1998, pp. 84–88.

[36] S. Ali, O. Maqbool, Monitoring software evolution using multiple types of changes, in: Proceedings of International Conference on Emerging Technologies. ICET 2009, 2009, pp. 410–415.

[37] D.L. Parnas, Software aging, in: Proceedings of the 16th International Conference on Software Engineering, Los Alamitos, CA, USA, 1994, pp. 279–287.

[38] Z. Li, P. Avgeriou, P. Liang, A systematic mapping study on technical debt and its management, J. Syst. Softw. 101 (Mar. 2015) 193–220.
[39] C. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of Software Engineering, 2 ed., N.J: Prentice Hall, Upper Saddle River, 2002.
[40] R. Harrison, S. Counsell, R. Nithi, Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems, J. Syst. Softw. 52 (2–3) (Jun. 2000) 173–179.
[41] A.J. Riel, Object-Oriented Design Heuristics, 1 ed., Addison-Wesley Professional, Reading, Mass, 1996.
[42] K.K. Aggarwal, Y. Singh, J.K. Chhabra, An integrated measure of software maintainability, in: Proceedings of the Annual Reliability and Maintainability Symposium, 2002., 2002, pp. 235–241.
[43] P. Oman, J. Hagemeister, Metrics for assessing a software system's maintainability, in: Proceedings of Conference on Software Maintenance, 1992., 1992, pp. 337–344.
[44] B. Ray, D. Posnett, V. Filkov, P. Devanbu, A Large scale study of programming languages and code quality in github, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, New York, NY, USA, 2014, pp. 155–165.
[45] W.M. Turski, The reference model for smooth growth of software systems revisited, IEEE Trans. Softw. Eng. 28 (8) (Aug. 2002) 814–815.
[46] D.J. Sheskin, D. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, Second Edition, 2 ed., Chapman and Hall/CRC, Boca Raton, 2000.
[47] P. Kruchten, R.L. Nord, I. Ozkaya, Technical debt: from metaphor to theory and practice, IEEE Softw. 29 (6) (Nov. 2012) 18–21.
[48] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Science & Business Media, 2012.