An Empirical Study of the Evolution of Eclipse Third-party Plug-ins

John Businge*
Mbarara University of Science and Technology
P.O. Box 1410
Mbarara, Uganda
johnxu21@yahoo.com

Alexander Serebrenik
Eindhoven University of Technology
P.O. Box 513, 5600 MB
Eindhoven, The Netherlands
a.serebrenik@tue.nl

Mark van den Brand
Eindhoven University of Technology
P.O. Box 513, 5600 MB
Eindhoven, The Netherlands
m.g.j.v.d.brand@tue.nl

ABSTRACT

Since the inception of Lehman's software evolution laws in the early 1970s, they have attracted significant attention from the research community. However, to our knowledge, no study of applicability of these laws on the software systems that exhibit constrained evolution process has been carried out so far. In this paper we take a first step in this direction and investigate the constrained evolution of 21 Eclipse third–party plug-ins. We investigate the trends followed by the plug-ins dependencies on Eclipse over time. The study spans 6 years of the evolution of Eclipse evolving from release 3.0 to release 3.5. Our findings confirm the laws of continuing change, self regulation and continuing growth when metrics related to dependencies between the plug-ins and the Eclipse Architecture are considered. Unlike this, the conservation of familiarity law was not confirmed and the results for the declining quality law were inconclusive.

Keywords

Evolution, Eclipse, Third-party Plug-ins, Dependencies on Eclipse, Metrics

1. INTRODUCTION

Component frameworks simplify the work of software developers because of their standard development structure with their reusable components in form of API. A third-party plug-in of a component framework is a software system that on the one hand is built to extend the capabilities of a framework and on the other hand is built to reuse the some of functionality provided by the framework [20]. The ease of development and maintenance of software systems built on a component framework lead to an impressive increase on the number of third-party plug-ins: for instance, at the moment of this writing Eclipse Marketplace `http://marketplace.eclipse.org/` lists more than 1000 third-party plug-ins for Eclipse. However, as the framework evolves due to the need to improve its functionality and quality, the changes frequently affect the API (dependencies) on which the third-party plug-in relies. As a result the plug-in breaks when ported to the new version of the framework. Hence, the evolution of a plug-in constrained by two independent, and potentially conflicting, processes: 1) it has to evolve to keep up with the changes introduced in the framework (framework-based evolution); 2) it has to evolve in response to the specific requirements and desired qualities of the stake holders (general evolution) [22].

A number of studies on the evolution of software systems have been carried out in the past years [21, 4, 9, 17]. However, the research community has given little attention to potentially conflicting evolution processes in cases where the software systems are built on component frameworks. Therefore, our goal consists in understanding *applicability of traditional results on software evolution to constrained evolution of Eclipse plug-ins*. Specifically, in this paper we consider Lehman's laws of software evolution [12, 11] as being representative for traditional studies of software evolution. To study constrained evolution we focus on dependencies of the third-party plug-ins on the Eclipse framework.

The reminder of the paper is organized as follows. In Section 2 we briefly explain the applications used in carrying out the study and state some definitions. In Section 3 we discuss Lehman's laws of software evolution according to the analyzed data. In Section 4 we discuss the possible threats to validity and the ways we countered them. Section 5 discusses some of the related work. Finally, Section 6 concludes the article.

2. ECLIPSE PLUG-IN ARCHITECTURE

The Eclipse plug-in architecture is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written primarily in Java and can be used to develop applications in Java and, by means of various plug-ins, languages such as C, C++, COBOL, Python, Perl and PHP [2]. Eclipse has evolved over nine year since its first release 1.0 in 2001 to release 3.5 in 2009 (the most recent release at the time of our data analysis). According to a recent press release [23], release 3.5 comprises 33 projects, 24 million LOC and 380 committers from 44 companies.

*This work has been carried out during the author's stay at Eindhoven University of Technology, The Netherlands

IWPSE-EVOL '10 , September 20-21, 2010 Antwerp, Belgium

Plug-in	Abbr	Yrs	# Vers	DownL[a]	Version		Date	SLOC	Solution Category
AnyEditTools	Any	6	17	–[d]	FV[b]	0.93	07-07-2004	1,842	Editor
					LV[c]	2.3.0	04-10-2009	10,237	
Acceleo	Acceleo	4	13	–	FV	1.1.0	01-09-2006	172,807	Modeling
					LV	2.6.0	01-06-2009	213,346	
ByteCodeOutline	BCO	6	15	–	FV	1.2.0	08-11-2004	2,769	Code management
					LV	2.2.12	20-07-2009	6,918	
CheckStyle	Check	6	16	40030	FV	3.3.1	22-01-2004	7,127	Source Code Analyzer
					LV	4.4.3	06-08-2009	23,449	
Ecelemma	Ecelemma	4	20	311	FV	0.1.1	25-08-2006	4,419	Testing
					LV	1.4.3	19-10-2009	8,708	
Extended VS Presentation	Extend	4	13	–	FV	1.3.0	01-08-2006	6,830	UI
					LV	1.5.3	24-05-2009	9,429	
Quantum Database Utility	Quantum	5	12	298	FV	2.4.5	23-02-2005	30,235	Database
					LV	3.3.8	06-11-2009	73,154	
TeXlipse	Tex	6	12	520	FV	1.0.0	16-03-2005	21,005	Documentation
					LV	1.4.0	04-04-2010	31,327	
PyDev	PyDev	5	52	2112	FV	1.0.6	24-04-2006	57,482	Editor/IDE
					LV	1.5.5	04-03-2010	205,568	
Java Hex Editor	Hex	4	31	54	FV	0.0.1	05-05-2006	1,795	Editor
					LV	0.3.1	28-06-2009	10,204	
Beyond CVS	Beyond	5	16	68	FV	0.5.1	11-11-2005	1,205	Code management
					LV	0.8.9	27-02-2010	7,906	
My TourBook	Tour	4	26	277	FV	0.8.4	03-01-2007	32,060	Rich Client Applications
					LV	10.3.0	25-03-2010	125,331	
Eclim (eclipse + vim)	Eclim	6	28	360	FV	1.1.0	26-12-2005	4,760	Intergration
					LV	1.5.5	23-02-2010	29,670	
Verilog editor	Verlog	7	22	141	FV	0.1.0	13-03-2004	1,251	Editor/IDE
					LV	0.7.1	09-04-2010	21,248	
PHPeclipse	PHP	6	13	1649	FV	1.0.9	28-06-2004	61,907	Editor/IDE
					LV	1.2.3	09-10-2009	171,647	
Eclipse Metrics	Metrics	6	13	72	FV	2.7.0	04-09-2004	4,157	Source Code Analyzer
					LV	3.14.0	28-10-2009	7,089	
GetText Editor	Gted	4	17	213	FV	0.1.0	06-01-2007	1,060	Editor
					LV	1.5.5	22-09-2009	4,602	
Green	Green	5	12	343	FV	2.4.0	12-10-2005	11,416	UML
					LV	3.4.0	22-10-2009	20,401	
ClearCase	Clear	7	15	208	FV	0.9.11	14-06-2004	6,803	SCM
					LV	2.2.1	04-03-2010	9,424	
JavaCC	JavaCC	6	15	207	FV	1.5.0	13-11-2005	15,292	Build & Deploy
					LV	15.16	04-10-2009	17,150	
SQL Explorer	SQL	6	17	956	FV	2.2.3	27-03-2005	19,360	Databases
					LV	3.5.1SR3	31-01-2010	33,785	

Table 1: Plug-in information

[a] Average weekly Downloads
[b] First Version
[c] Last Version
[d] Download statistics could not be obtained from the respective web pages.

The architecture is structured as follows: the internal plug-ins which are the building blocks of the framework and the third-party plug-ins which extend the capabilities of the framework and reuse the functionality provided by the internal plug-ins. Eclipse internal plug-ins have been extensively analyzed in [19]. Our focus in this paper is Eclipse third-party plug-ins.

2.1 Third-party Plug-ins

Third-party plug-ins are applications developed by independent software vendors built on top of Eclipse. Since Eclipse evolves its APIs, the third-party plug-ins should co-evolve with it to use the new functionality and/or to fix changes. From now and onwards the word ***plug-in*** will be used to refer to ***third-party plug-in***.

In this paper we study dependencies of the plug-ins on Eclipse SDK. We say that a plug-in class `A.java` depends on Eclipse if it imports at least one class from `org.eclipse.`

- ***Deps:*** This word will often be used to refer to *dependency on Eclipse*; all metrics with the word *Deps* will be used to study the Eclipse based evolution of the plug-ins.
- ***Tot:*** The word will often be used to refer to total; all metrics with the word *Tot* (except *Deps-Tot*) will be used to study general evolution of plug-ins (explained in Table 2).

Our investigation is based upon 21 Open Source Eclipse plug-ins written in Java [3, 1].

The selection of a plug-in is based on the following criteria. First, a plug-in should have dependencies on Eclipse. All the dependencies in a plug-in should indicate a particular Eclipse class/interface being imported and not a group of classes in a package, e.g., plug-ins that had dependencies that include `org.eclipse.foo.*` were excluded. The reason for the exclusion is that they would affect our results since from our method of data collection we cannot tell which specific class/interface from this package are being used by the plug-in. Second, at least one of the most recent versions of the plug-in should support Eclipse 3.5 and one of the earlier versions should support Eclipse 3.2 or earlier. We consider plug-in versions starting from Eclipse release 3.0 (released 25 June, 2004). Third, the plug-in should have at least 12 or more versions since its first release so that we fine-grained growth trends and reduce on the noise threat

Metric	Description	Tool
Deps-Tot	Total number dependencies on Eclipse in a plug-in	Windows Grep[a] Excel
Deps-Uniq	Unique dependencies on Eclipse in a plug-in	Excel
SLOC-Tot	Total SLOC in a plug-in	SLOCCount[b]
SLOC-Deps	SLOC in a plug-in considering only the .java files having Deps on Eclipse	SLOCCount
NOC-Tot	Total NOC(.java) in a plug-in	Windows Grep
NOC-Deps	NOC(.java) with dependencies on Eclipse	Windows Grep
Add-Deps	Added Deps in a new version of a plug-in	Excel
Del-Deps	Deleted Deps in a new version of a plug-in	Excel
Add-NOC-Tot	Total Added NOC(.java) in a new version of a plug-in	DependencyFinder[c]
Mod-NOC-Tot	Total Modified NOC(.java) in a new version of a plug-in	DependencyFinder
Del-NOC-Tot	Total Deleted NOC(.java) in a new version of a plug-in	DependencyFinder
Add-NOC-Deps	Added NOC(.java) having Deps in a new version of a plug-in	DependencyFinder
Mod-NOC-Deps	Modified NOC(.java) having Deps in a new version of a plug-in	DependencyFinder
Del-NOC-Deps	Deleted NOC(.java) having Deps in a new version of a plug-in	DependencyFinder
D_n	Normalized distance from the main sequence	JDepend[d]

Table 2: Metrics and the corresponding Tools

[a] http://www.wingrep.com/
[b] http://www.dwheeler.com/sloccount/
[c] http://sourceforge.net/projects/depfind/
[d] http://clarkware.com/software/JDepend.html

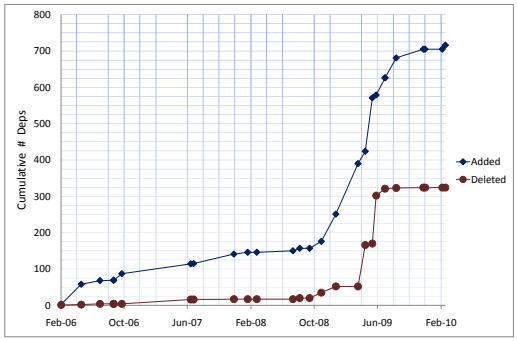

Figure 1: Cumulative Added and Deleted Deps to Eclim

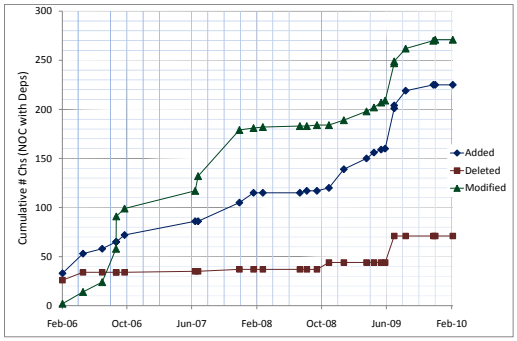

Figure 2: Cumulative changes of classes with Deps in Eclim

in the models. Although the Eclipse plug-in architecture has got over 1,000 plug-ins, only 21 of them satisfied our selection criteria.

The following criteria was used to extract the metric values with he help of the mentioned tools:

- ***Dependencies:*** We used windows grep to extract for us the dependencies from the plug-ins. Thereafter we used Excel to sort out the Deps-Tot and Deps-Uniq and also made sure that the dependencies did not include the jolly characters `org.eclipse.foo.*`.
- ***Changed dependencies:*** an excel function was written to extract the changes.
- ***Changed classes:*** we wrote a script to such the output of the the tool DependencyFinder

We wrote excel functions to determine for us changed dependencies: Add-Deps, Del-Deps. For changed classes, the tool DependencyFinder produced an xml file for the class changes and thereafter we wrote our own script to collect the values for the different changes.

As seen in Table 1 the studied plug-ins belong to a wide range of software domains, sizes and ages. For each of the plug-in we indicate the plug-in name, abbreviation, the age (in years), number of versions, weekly download statistics, first version (version, date of release, SLOC), last version (version, date of release, SLOC), and the solution category.

- ***First Version:*** the first release we consider is the first release having both sources and binaries and in addition its Java source code contains dependencies on Eclipse. For example, *PyDev* has got 88 versions with the first version (0.1) going as far as 05-08-2003 but the versions with with Eclipse SDK dependencies in the Java code starts from version 1.0.6.
- ***Solution Category:*** is reported according to the assignment on the Eclipse Marketplace site [1].

3. LAWS OF SOFTWARE EVOLUTION

Lehman's laws describe the evolution of *E-type* software systems [14]. Similar to the study of "The Evolution of Eclipse" in [17], we consider the Eclipse third-party plug-ins as *E-type* systems. Indeed, the plug-ins are actively used and hence, external pressure demands them to change over time. In our study, we investigate whether the constrained evolution of Eclipse plug-ins follows Lehman's laws of software evolution [12, 11]. This being our first study on the laws and due to space limitations, we focus on 5 out of the 8 laws: continuing change, self regulation, conservation of familiarity, continuing growth and declining quality. The remaining 3 will be considered a follow-up study.

3.1 Continuing Change

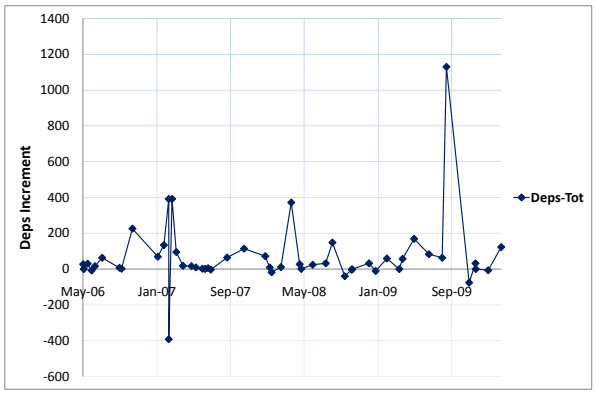

Figure 3: Incremental Deps-Tot growth of PyDev

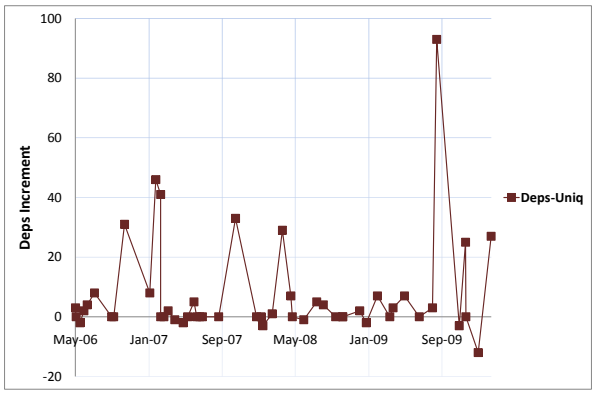

Figure 4: Incremental Deps-Uniq growth of PyDev

The law states an *E-type* system must be continually adapted else it becomes progressively less satisfactory. All our plug-ins are actively used as can be seen from the number of versions and weekly downloads in Table 1. Previous studies have mainly used number of modules handled (additions, modifications and deletions) in each release e.g. [4] employs the count of additions and changes in a portfolio, [21] employs cumulative number of changes. In our study, we employ two different kinds of metrics: cumulative number of Add-Deps and Del-Deps in a plug-in over time and the cumulative number of class changes (Add-NOC-Deps, Mod-NOC-Deps, and Del-NOC-Deps) in a plug-in over time (see Table 2 for detailed description of the metrics). Figures 1 and 2 present respectively the graphs for the plug-in *Eclim* for the two kind of metrics considered. From our observations, the rest of the plug-ins exhibit more or less similar trends. From the observed trends of the number of additions, deletions and modifications in the plug-ins, we conclude that the Eclipse based evolution of the plug-ins conforms to Lehman's law of continuing change.

3.2 Self Regulation

The law states that, the program evolution process is self regulating, i.e., the system will adjust its size throughout its lifetime. Evidence of this law according to Lehman [12, 15] is that empirical growth curves show a ripple superimposed on a steady growth trend. The ripples demonstrate the interaction between the conflicting forces of desired growth and bounded resources.

To verify this law, we analyzed the incremental growth of the dependencies on Eclipse for each of the plug-ins. Figure 3 and 4 present the graphs for the plug-in *PyDev*. We can observe that these ripples exist with the positive adjustments occurring more frequent than the negative adjustments, a trend shared by the rest of the plug-ins. The same behavior is observed when we consider the classes i.e., NOC-Tot and NOC-Deps. The plots for the rest of the plug-ins be found at [5]. We therefore conclude that the Eclipse based evolution of the plug-ins conforms to Lehman's law of self-regulation.

3.3 Conservation of Familiarity

The law suggests that, in general, the incremental growth and long term growth rate of *E-type* system tend to remain constant or to decline[12]. Lehman states in [12] that, the *rate of change and growth* of the system can be slowed down as it ages. A recent study by Xie et al. [21] suggests that both growth rate and change rate are neither invariant nor decreasing but the authors do not verify this claim statistically. The conclusion was based on visual inspection of the observed trends on the graphs. Barry et al. [4] considered only growth rate and found it decreasing and the result was statistically significant. In our study, we statistically test both *change* and *growth rate* and present results both for Eclipse based evolution and for general evolution of the plug-in. For *change rate* we use as a metric percentage of *handled files* (additions, deletions, modifications) in the previous version of a plug-in as the *dependent* variable. For *growth rate* we use two metrics: percentage of *added Deps* and percentage of *added NOC* in the previous version of a plug-in as the *dependent* variables respectively. The *independent* variable in all the cases is the plug-in age measured as the number of weeks since the first release considered.

A series of hypotheses are formulated to determine whether the evolution with respect to /evolution conforms to the stated law. We use both linear and quadratic models for the regression lines to determine the relationship between the dependent and the independent variables. We employ both linear and quadratic regression t-test to verify the statistical significance of the relationships. For each plug-in, we first test relationship of the dependent and the independent variables with the linear model. Second, for plug-ins that cannot be explained by the linear model ($p - value > 0.1$), we employ the quadratic model: if the relationship can be explained by the simplest model, the linear model, then further analysis with a nonlinear model will not produce a significant change in the result. For the choice of $p - value$ ($= 0.1$), we follow [13].

$$Y_t = \alpha + \beta * AGE_t + \varepsilon_t \quad (1)$$

$$Y_t = \alpha + \beta_1 * AGE_t + \beta_2 * AGE_t^2 + \varepsilon_t \quad (2)$$

Equations 1 and 2 represent the linear and quadratic model respectively. The common terms in both models: Y_t represents the particular metric (dependent variable) used to evaluate the law for a time period (weeks)t, α is the Y_t intercept or constant term, AGE_t is the variable for plug-in age that varies with time period (weeks)t and ε is the error term of the model. For the linear model, β represents the coefficient of the independent variable. For the quadratic model, AGE_t^2 is the term added to allow quadratic relationship, β_1 is the same as β and β_2 is is the coefficient of the

Plug-in		Adj R^2	β	Std err	t	$P > \|t\|$	Support	$P - val$[b]
Any	Tot	0.593	**-0.427**	0.098	-4.356	**0.001**	**YES**	0.993
	Deps	0.603	**-0.506**	0.114	-4.442	**0.001**	**YES**	0.993
Acceleo	Tot	0.324	**-0.155**	0.065	-2.408	**0.039**	**YES**	0.804
	Deps	0.390	**-0.198**	0.075	-2.717	**0.024**	**YES**	0.596
BCO	Tot	0.192	**-0.112**	0.055	-2.020	**0.066**	**YES**	0.176
	Deps	0.148	**-0.082**	0.046	-1.804	**0.096**	**YES**	0.389
Check	Tot	-0.071	-0.021	0.078	-0.272	0.790	NO	0.061
	Deps	-0.070	-0.025	0.089	-0.281	0.783	NO	0.056
Eclemma	Tot	0.022	-0.121	0.103	-1.177	0.256	NO	0.062
	Deps	0.026	-0.152	0.126	-1.207	0.245	NO	0.069
Extend[a]	Tot	0.116	-0.128	0.082	-1.564	0.149	NO	0.998
	Deps	0.116	-0.128	0.082	-1.564	0.149	NO	0.998
PyDev	Tot	-0.018	0.004	0.009	0.371	0.712	NO	0.037
	Deps	-0.021	-0.002	0.008	-0.181	0.857	NO	0.115
Hex	Tot	0.118	**-0.090**	0.041	-2.209	**0.036**	**YES**	0.799
	Deps	0.109	**-0.113**	0.053	-2.130	**0.042**	**YES**	0.281
Beyond	Tot	0.760	**-1.053**	0.162	-6.491	**0.000**	**YES**	0.979
	Deps	0.763	**-1.053**	0.161	-6.550	**0.000**	**YES**	0.986
Eclim	Tot	0.336	**-0.192**	0.052	-3.696	**0.001**	**YES**	0.153
	Deps	0.408	**-0.266**	0.062	-4.269	**0.000**	**YES**	0.338
Verilog	Tot	-0.029	-0.075	0.111	-0.676	0.507	NO	0.007
	Deps	0.057	-0.099	0.068	-1.465	0.160	NO	0.014
Gted	Tot	0.133	**-0.577**	0.317	-1.818	**0.091**	**YES**	0.257
	Deps	0.107	-0.847	0.506	-1.672	0.117	NO	0.081
Green	Tot	0.267	**-0.111**	0.052	-2.154	**0.060**	**YES**	0.971
	Deps	0.393	**-0.138**	0.051	-2.732	**0.023**	**YES**	0.941
Clear	Tot	0.548	**-0.090**	0.022	-4.092	**0.001**	**YES**	0.283
	Deps	0.529	**-0.093**	0.024	-3.950	**0.002**	**YES**	0.269
SQL	Tot	-0.021	-0.115	0.138	-0.831	0.420	NO	0.018
	Deps	-0.028	-0.080	0.104	-0.768	0.455	NO	0.022

Table 3: Summary of Conservation of Familiarity (NOC ***change*** rate) results $p < 0.1$ and $\beta \leq 0$ in bold

[a] All the classes have Deps
[b] $p - value$ of the Kolmogorov-Smirnov test for normality

Plug-in		Adj R^2	β_2	β_1	Std err	t_2	t_1	$P > \|t_2\|$	$P > \|t_1\|$	Support	$P - val$
Extend	Tot	0.332	0.002	-0.927	0.001	2.057	-2.348	**0.070**	0.043	NO	0.392
	Deps	0.332	0.002	-0.927	0.001	2.057	-2.348	**0.070**	0.043	NO	0.392
PyDev	Tot	0.032	0.0002	-0.058	0.034	1.863	-1.718	**0.069**	0.093	NO	0.018
	Deps	0.020	0.0001	-0.052	0.030	1.697	-1.730	**0.096**	0.090	NO	0.027

Table 4: Summary of Conservation of Familiarity (NOC ***change*** rate) quadratic model

Plug-in	Adj R^2	β	Std err	t	$P > \|t\|$	Support	$P - val$[a]
Any	0.69	**-0.153**	0.037	-5.869	**0.000**	**YES**	0.962
Acceleo	-0.077	0.015	0.032	0.464	0.653	NO	0.637
BCO	0.129	-0.039	0.023	-1.713	0.112	NO	0.287
Check	-0.060	-0.026	0.057	-0.459	0.654	NO	0.017
Eclemma	-0.061	-0.007	0.040	-0.172	0.865	NO	0.297
Extend	-0.060	-0.009	0.014	-0.614	0.553	NO	0.114
PyDev	-0.021	-0.0003	0.004	-0.081	0.936	NO	0.000
Hex	-0.035	-0.002	0.010	-0.163	0.872	NO	0.013
Beyond	0.732	**-0.307**	0.051	-6.039	**0.000**	**YES**	0.982
Eclim	-0.006	-0.033	0.036	-0.921	0.366	NO	0.064
Verilog	-0.032	-0.008	0.012	-0.637	0.532	NO	0.023
Gted	0.110	-0.150	0.089	-1.690	0.113	NO	0.266
Green	-0.105	0.004	0.018	0.220	0.830	NO	0.867
Tour	0.027	-0.013	0.011	-1.272	0.217	NO	0.027
SQL	-0.036	-0.023	0.033	-0.691	0.501	NO	0.035

Table 5: Summary of Conservation of Familiarity (Deps growth rate) results $p < 0.1$ and $\beta \leq 0$ in bold

[a] $p - value$ of the Kolmogorov-Smirnov test for normality

Plug-in	Adj R^2	β_2	β_1	Std err	t_2	t_1	$P > \|t_2\|$	$P > \|t_1\|$	Support	$P - val$
Hex	0.133	0.0003	-0.084	4.268	2.532	-2.489	**0.017**	0.019	NO	0.180

Table 6: Summary of Conservation of Familiarity (Deps ***growth*** rate) quadratic model

AGE_t^2.

3.3.1 Hypotheses

We present a series of hypotheses for change rate and growth rate: Hypotheses *H2*, *H4* and *H5* test Eclipse based evolution while Hypotheses *H1* and *H3* test evolution.

Change Rate:

Plug-in		Adj R^2	β	Std err	t	$P > \|t\|$	Support	$P - val$[a]
Any	Tot	0.466	**-0.165**	0.045	-3.638	**0.003**	**YES**	0.920
	Deps	0.390	**-0.157**	0.050	-3.154	**0.008**	**YES**	0.635
Acceleo	Tot	-0.097	-0.006	0.053	-0.177	0.863	NO	0.794
	Deps	-0.074	-0.013	0.065	-0.493	0.633	NO	0.910
BCO	Tot	0.238	**-0.036**	0.016	-2.251	**0.044**	**YES**	0.282
	Deps	0.063	-0.021	0.009	-1.367	0.197	NO	0.257
Check	Tot	-0.066	-0.012	0.030	-0.372	0.716	NO	0.056
	Deps	-0.065	-0.013	0.034	-0.380	0.710	NO	0.000
Eclemma	Tot	-0.031	-0.013	0.018	-0.695	0.497	NO	0.075
	Deps	-0.021	-0.017	0.021	-0.811	0.429	NO	0.092
Extend	Tot	0.295	**-0.070**	0.029	-2.367	**0.039**	**YES**	0.995
	Deps	0.295	**-0.070**	0.029	-2.367	**0.039**	**YES**	0.995
PyDev	Tot	-0.015	-0.002	0.004	-0.525	0.602	NO	0.000
	Deps	0.017	-0.003	0.006	-0.424	0.673	NO	0.000
Hex	Tot	0.126	**-0.022**	0.010	-2.273	**0.031**	**YES**	0.153
	Deps	0.124	**-0.035**	0.016	-2.263	**0.032**	**YES**	0.111
Beyond	Tot	0.750	**-0.405**	0.064	-6.320	**0.000**	**YES**	0.999
	Deps	0.751	**-0.106**	0.064	-6.345	**0.000**	**YES**	1.000
Eclim	Tot	0.246	**-0.035**	0.012	-3.029	**0.006**	**YES**	0.768
	Deps	0.270	**-0.045**	0.014	-3.202	**0.004**	**YES**	0.901
Verilog	Tot	-0.054	0.010	0.075	0.140	0.890	NO	0.011
	Deps	0.016	-0.018	0.016	-1.140	0.268	NO	0.008
Gted	Tot	0.054	-0.277	0.203	-1.364	0.194	NO	0.076
	Deps	0.046	-0.512	0.390	-1.312	0.211	NO	0.078
Green	Tot	0.047	0.025	0.020	1.221	0.253	NO	0.843
	Deps	-0.107	0.003	0.019	0.182	0.860	NO	0.753
Clear	Tot	0.331	**-0.017**	0.006	-2.728	**0.018**	**YES**	0.560
	Deps	0.345	**-0.017**	0.006	-2.802	**0.016**	**YES**	0.487
SQL	Tot	-0.036	-0.029	0.042	-0.696	0.498	NO	0.028
	Deps	-0.038	-0.030	0.044	-0.675	0.511	NO	0.012

Table 7: Summary of Conservation of Familiarity (NOC ***growth*** rate) results $p < 0.1$ and $\beta \leq 0$ in bold

[a] $p - value$ of the Kolmogorov-Smirnov test for normality

- ***(Handled NOC-Tot)*** $H1_0$*:* $\beta > 0$ *(change rate increases over time)* $H1_a$: $\beta \leq 0$ (change rate is invariant or declines over time)
- ***(Handled NOC-Deps)*** $H2_0$*:* $\beta > 0$ *(change rate increases over time)* $H2_a$: $\beta \leq 0$ (change rate is invariant or declines over time)

Growth Rate:

- ***(Deps)*** $H3_0$*:* $\beta > 0$ *(growth rate increases over time)* $H3_a$: $\beta \leq 0$ (growth rate is invariant or declines over time)
- ***(Handled NOC-Tot)*** $H4_0$*:* $\beta > 0$ *(growth rate increases over time)* $H4_a$: $\beta \leq 0$ (growth rate is invariant or declines over time)
- ***(Handled NOC-Deps)*** $H5_0$*:* $\beta > 0$ *(growth rate increases over time)* $H5_a$: $\beta \leq 0$ (growth rate is invariant or declines over time)

3.3.2 Results

The results used to verify the stated hypotheses are presented in Tables 3–7. The metrics used to analyze this law were extracted from both binaries and the corresponding sources; for six of the plug-ins we were not able to obtain some of the sources that correspond to the binaries. We therefore excluded them and only present 15 of 21 plug-ins. Tables 3, 5 and 7 present the results for the linear model and Tables 4 and 6 present results for the quadratic model. Due to space constraints only entries with interesting observations are presented for the quadratic model.

In the column headers of we use the following notation: for the common terms, $AdjR^2$ represents the adjusted R-square; used to explain the variation explained by the dependent variable, Std err represents the standard error of the slope. For the terms in linear model tables, t is the t-score test statistic and $P > |t|$ represents the $p - value$ that corresponds to t with certain degrees of freedom (number of versions in Table 1). For terms in the quadratic model tables, t_1 and t_2 are the t-score test statistics for β_1 and β_2 respectively and $P > |t_2|$ and $P > |t_1|$ are the resulting $p - values$ respectively.

To ensure that the results of the significance tests are meaningful, we carried out the Kolmogorov-Smirnov normality test on the error component, ε, in the models (last column). At a given significance level, $(100 - \alpha)\%$, the test compares the distribution followed by the error component and the normal distribution. The null hypothesis is that the two distributions are different and the alternative hypothesis is that they are the same. If the result of the test is less than α, then their is a $(100 - \alpha)\%$ chance that the two distributions are different. From our results, considering a threshold, $\alpha = 0.01$, only a total of six entries in all the tables fail the normality test.

Tables 3 and 4 present the *change rate–general evolution* (Tot) and *change rate–Eclipse based evolution* (Deps) for the plug-ins. For hypothesis $H1$ we have 9 of the 15 plug-ins supporting the law and for $H2$ we have 8 of the 15 supporting the law. However, out of the 9 and 8 plug-ins in Tot and Deps respectively, only 2 plug-ins, *Any* and *Beyond*, express a strong co-efficient of variation ($AdjR^2 > 0.5$). In Table 4, plug-ins *PyDev* and *Extend* show a statistically significant result ($p - value < 0.1$) but their result in Table 3 are not statistically significant. This means that the *change rate* of these two plug-ins is more accurately modeled by the quadratic model. A more interesting observation is that

as opposed to exhibiting an invariant or declining *change rate* as the law stipulates, an increasing trend is observed ($\beta_2 > 0$). Further investigation of these 2 plug-ins, show that the growth of both the NOC-Tot and NOC-Deps follows a super-linear trend. This result corresponds to Koch's observation that projects with a super-linear growth are more active in number of revisions than those with linear and more active than those with sub-linear [10]. A chaotic change rate was observed for rest of the plug-ins.

Tables 5–7 present the growth rate–general evolution (Tot) and growth rate–Eclipse based evolution (Deps) for the plug-ins. For hypothesis $H3$ we have 2 of the 15 plug-ins supporting the law which also have strong co-efficient of variation ($\approx AdjR^2 \geq 0.7$). Plug-ins BCO and Gted are nearly statistically significant at the chosen significant level 0.1. For hypothesis $H4$ we have 7 of the 15 plug-ins in support and hypothesis $H5$ we have 6 of the 15 in support; only *Beyond* has got a strong coefficient of variation. The large significant difference in the results of *Hex* in Table 5 and 6 imply that the metric can be better modeled by the quadratic model. The results in Table 6 show an increasing trend of the growth rate of Deps ($\beta_2 > 0$). Further investigation on this plug-in revealed that the metric Deps exhibits a superlinear growth trend. This corresponds to Godfrey and Tu's findings that Linux exhibits a super-linear growth trend [8].

From the analysis of the hypotheses, we cannot reject or accept any of them since some plug-ins are in support of the law and others are not. We therefore, conclude that the law is not confirmed by the plug-in considered.

3.4 Continuing Growth

The law states that, the functional capability of E-type systems must be continually increased to maintain user satisfaction over the system lifetime[12]. Despite the fact that the law talks about functional capability, most of the research that has been carried out so far have considered metrics that measure size of the program: e.g., number of sub-projects, number of features, number plug-ins, NOC, NOF[17], number of modules [4, 21], LOC [21, 8, 17], number of definitions [21] were considered. One of the main reasons for this choice of metrics is that it is not easy to measure the functional capability. In addition to the metrics that have been used to measure size and growth over time for the programs studied so far, we also study Deps-Tot, Deps-Uniq, SLOC-Tot, SLOC-Deps, NOC-Tot and NOC-Deps (cf. 2). After analysis of the results of these metrics, we have observed that there is an increasing trend for all the chosen metrics on all the studied plug-ins. Due to space constraints, we present a few graphs, Figure 5 and 6, to show our results. The rest of the metrics show general increase in the trends for all the plug-ins and be found at [5]. From the analysis of the selected plug-ins we conclude that the law is confirmed on the two evolutionary processes considered.

3.5 Declining Quality

The law states that the quality of *E-type* systems will appear to be declining unless they are rigorously adapted, as required, to take into account changes in the operational environment Lehman [12]. Lehman continues to state that quality is a function of many aspects; these aspects need to be quantified to be controllable. Subject of being observed and measured in a consistent way, associated measures of quality can be defined for a system, project or organization and their value may then be tracked over *releases* or *units of time* and analyzed to determine whether levels and trends are required or desired. Some of the metrics Lehman suggests include fault rates, rate of fixes in previous releases and many others, to observe the fitted trend line (or other models) whether it increases, decreases or remains steady over time.

Like the law of conservation of familiarity in Section 3.3, we use empirical results from the plug-ins to analyze whether the plug-ins' Eclipse based evolution follows the law. We employ the same methodology to evaluate the conformity of this law. The metrics used for the independent variable are obtained from Martin's [16] normalized distance from the main sequence, denoted by D_n. D_n is an indicator of the package's balance between abstractness and stability. A low value (closer to zero) of average and standard deviation of the package's D_n in a software system is an indicator of high quality. To test the conformance of this law we follow [18] and investigate the growth trends of averages ($\bar{D_n}$) and standard deviations ($\sigma(D_n)$) of the different versions of the plug-ins over time. The method used in determining our average D_n is borrowed from [18] in the study of formalizing D_n-based architecture assessment of Java Open Source software. To ensure the validity of our results, we follow [18] and also select plug-ins having at least thirty packages (not including third party packages). Only 8 of the 21 plug-ins satisfied this requirement.

3.5.1 Hypotheses

We state the hypotheses that will be used to test the evolution.

- $H6_0$: $\beta \leq 0$ *($\bar{D_n}$'s of the different packages in the different plug-in versions will not increase over time)* $H6_a$: $\beta > 0$ *($\bar{D_n}$'s of the different packages in the different plug-in versions will increase over time)*
- $H7_0$: $\beta \leq 0$ *$\sigma(D_n)$'s of the different packages in the different plug-in versions will not increase over time)* $H7_a$: $\beta > 0$ *($\sigma(D_n)$'s of the different packages in the different plug-in versions will increase over time)*

3.5.2 Results

Tables 8 and 9 show the results for the linear and quadratic models analyzed. From the Table 8 considering the *MEAN*, we have four of the eight plug-ins in support of the law. In Table 9 *Tex* is in support of the law. This means that the $\bar{D_n}$'s are better modeled by the quadratic model. This increases the number of plug-ins in support of the law to 5 out of 8. Much as the coefficients of AGE, β's, appear to be small from the tables, the values are sufficient since the values of the dependent variable are large (up to 300-500 weeks) and all the Y_t intercepts are > 0. In addition, the result of the model(s) is in the range, $0 \leq Y_t \leq 1$.

For *SQL* we have an exceptional case where the $\bar{D_n}$'s are significantly decreasing and the $\sigma(D_n)$'s are significantly increasing. On inspection of the raw data we observed that in the last seven versions of the plug-in introduced many packages that had $D_n = 0$, this in turn lowered the averages. Therefore the increase in $\sigma(D_n)$'s tells us that the quality the old packages has not changed. *PyDev* exhibits a statistically significant increase in the $\bar{D_n}$'s and the $\sigma(D_n)$'s are

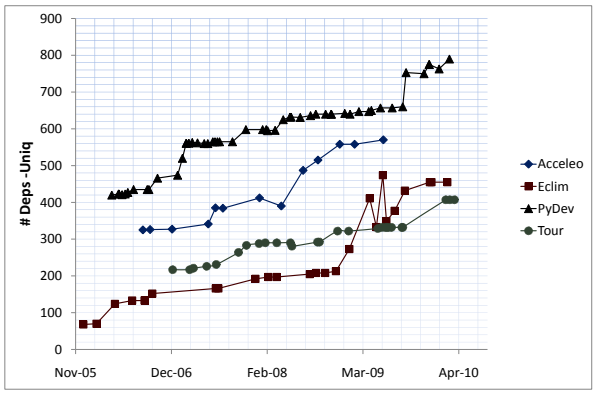

Figure 5: Growth Trend of Deps-Uniq for selected plug-ins

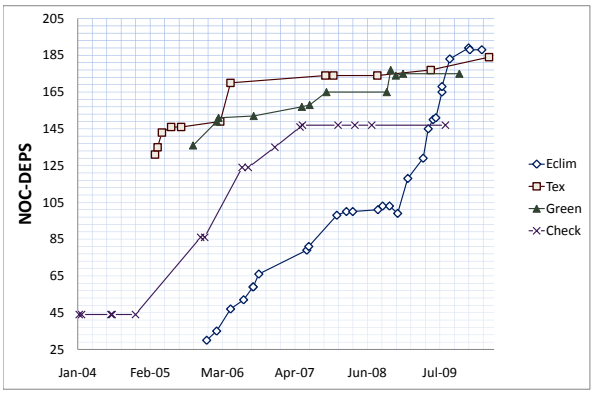

Figure 6: Growth Trend of NOC-Deps for selected plug-ins

Plug-in		Adj R^2	β	Std err	t	$P > \|t\|$	Support	$P - val^a$
Acceleo	MEAN	0.275	**3.9E-05**	1.7E-05	2.355	**0.038**	**YES**	0.331
	STDV	-0.032	2.7E-05	3.4E-05	0.789	0.447	NO	0.159
Quantum	MEAN	0.902	-3.7E-05	3.6E-06	-10.126	0.000	NO	0.641
	STDV	0.822	-4.9E-05	6.9E-06	-7.199	0.000	NO	0.244
Tex	MEAN	-0.024	-1.2E-05	1.4E-05	-0.860	0.410	NO	0.602
	STDV	0.905	**3.6E-05**	3.5E-06	10.313	**0.000**	**YES**	0.717
PyDev	MEAN	0.307	**7.8E-05**	1.6E-05	4.813	**0.000**	**YES**	0.168
	STDV	0.013	-8.0E-06	6.2E-06	-1.292	0.203	NO	0.071
Eclim	MEAN	0.124	**7.8E-05**	3.6E-05	2.199	**0.037**	**YES**	0.020
	STDV	0.786	**1.5E-04**	1.5E-05	10.022	**0.000**	**YES**	0.997
PhP	MEAN	0.027	-1.3E-05	1.1E-05	-1.157	0.272	NO	0.250
	STDV	0.031	-8.6E-06	7.3E-06	-1.177	0.264	NO	0.541
Metrics	MEAN	0.774	**1.9E-04**	2.6E-05	7.368	**0.000**	**YES**	0.781
	STDV	0.018	**2.5E-05**	2.2E-05	1.128	0.278	NO	0.648
SQL	MEAN	0.516	-7.4E-05	1.9E-05	-3.994	0.000	NO	0.547
	STDV	0.430	**5.8E-05**	1.7E-05	3.397	**0.005**	**YES**	0.939

Table 8: Growth trends of the $\bar{D_n}$'s and $\sigma(D_n)$'s of the plug-ins that passed the selection criteria
[a] $p - value$ of the Kolmogorov-Smirnov test for normality

Plug-in		Adj R^2	β_2	β_1	Std err	t_2	t_1	$P > \|t_2\|$	$P > \|t_1\|$	Support	$P - val^a$
Acceleo	MEAN	0.278	2.5E-07	-2.1E-05	2.4E-07	1.583	-1.290	0.145	0.226	NO	0.886
	STDV	0.092	7.2E-07	-1.5E-04	4.6E-07	1.583	-1.290	0.145	0.226	NO	0.866
Tex	MEAN	0.893	**3.4E-07**	-1.5E-04	3.7E-08	9.329	-9.207	**0.000**	0.000	**YES**	0.990
	STDV	0.895	-1.8E-09	**3.7E-05**	3.1E-08	-0.057	2.789	0.956	**0.021**	**YES**[b]	0.985

Table 9: Growth trends of the $\bar{D_n}$'s and $\sigma(D_n)$'s of plug-ins for the quadratic model
[a] $p - value$ of the Kolmogorov-Smirnov test for normality
[b] Coefficient of AGE_t^2, β_2, is to small and therefore ignored

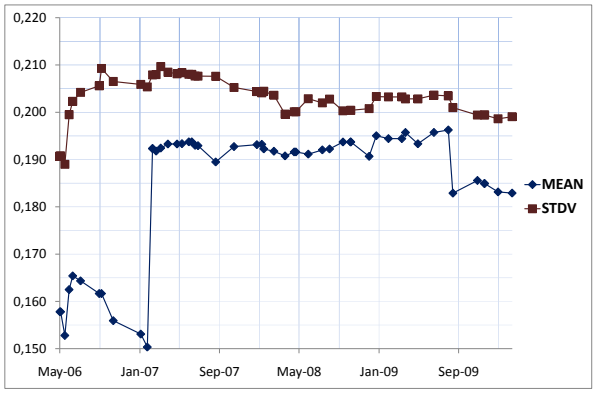

Figure 7: Growth Trend of $\bar{D_n}$'s and $\sigma(D_n)$'s for *PyDev* plug-in

close to significantly decreasing. The reason for this behavior can be obtained by observing the growth trend of D_n's, Figure 7. The trend is staged; where it is more or less stable in the early stages, then a sharp increase, then it becomes stable for a long period, after it goes down and finally it becomes stable again. *Quantum* exhibits a increasing quality as seen from both growth trends of $\bar{D_n}$'s and $\sigma(D_n)$'s decreasing over time. Since the law states that unless the systems being evolved are rigorously adapted, the quality is supposed to decline. It is possible that this might be the case with *Quantum* plug-in but we leave further investigation for the future study when we increase the number of plug-ins being studied. Since we only studied 8 plug-ins our conclusions will be threatened by external validity, therefore our findings for this law are inconclusive.

4. THREATS TO VALIDITY

Like any other empirical analysis, our study may have been affected by validity threats. We categorized the possible threats into construct, internal and external validity.

4.1 Construct Validity

This validity relies on the assumption that the independent and the dependent variables accurately model the intended characteristic we are studying. In our situation, for example, whether the time series of the chosen metrics accurately model the characteristics claimed in the laws we have studied. To mitigate this threat, we used a number of metrics for each of the studied laws. Despite the fact that a number of studies, including this one, carried out on the law of continuing growth use metrics that relate to size, the law talks about functional content. A study that validates whether a software systems' size and functional content exhibit same evolution patterns needs to be carried out.

4.2 Internal Validity

The key question internal of validity in our study is whether the observed changes in the dependent variable can be attributed to the changes in the independent variable and not other possible causes. This threat seems to be comparably lesser in our study since we tried to mitigate it during the plug-in inclusion/exclusion in our study by the selection criteria we set in Section 2.1. For the two laws conservation of familiarity and declining quality, we further strengthened the significance test results by making sure that the error, ε, component in the models follows a normal distribution by carrying out the Kolmogorov-Smirnov normality tests.

4.3 External Validity

This threat refers to the degree to which our conclusions from our findings can be generalized. In our study we observed two threats that relate to external validity. First, as we already pointed out in Section 3.5 our findings were inconclusive since the sample size reduced from 21 to only 8 plug-ins. Second, we are studying the Eclipse based evolution of Eclipse and its plug-ins but we only consider Open Source plug-ins.

5. RELATED WORK

Most of the studies that are related to our work have already been implicitly introduced in Section 3 while discussing the different laws we studied. Studies of Xie et al. [21] and Barry et al. [4] are yet to be discussed and are closely related to our study.

Xie et al. [21] investigate all Lehman's evolution laws by conducting an empirical study on the evolution of seven long-lived (5–15 years) Open Source programs written in the C language. We employ similar methods for verifying the laws of continuing change, self regulation, conservation of familiarity, and declining quality. In both studies, the laws of continuing change and self regulation were validated and the laws of conservation of familiarity and declining quality were not validated. In the study of conservation of familiarity, the authors base their conclusion on visual inspection of the growth trends of the metrics they employed. We based our conclusion on quantitative analysis using statistics. For the law of declining quality, their analysis was based on observing the sign of the slope and the correlation coefficient between the independent and the dependent variable. In our study, we employ hypothesis tests on the models we stated which we believe is a better way of studying growth of a time series variables since in addition to sign of slope and correlation coefficient, it incorporates other terms that strengthens the reliability of the results.

Barry et al. [4] studied Lehman's laws on 23 application systems for a large retail company that had evolved over 20 years. In the data analysis, they stated hypotheses for each law and employed a time series regression to test each of the hypotheses. Two of the laws; conservation of familiarity and declining quality, both studies (our study and their study) employed similar methods to analyze the data. The first difference between the studies is that, they found support of these two laws which we did not. A summary of their findings was reported but it is not clear if all or the majority of the 23 applications studied conform to the law. If this is the case, which we could not establish with our applications, one reason for the difference could be their applications were drawn from the same domain (same development conventions are applied to all applications since they are developed from the same company). Second, the authors analyze growth rate in verifying the conservation of familiarity law, whereas we analyze both growth and change rate. Third, the two studies employ different metrics in analyzing the declining quality law.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we took the first step to study the fine grained evolution, of software systems built from component frameworks, by investigating their evolutionary trends using Lehman's laws of software evolution. Our study was based on 21 third-party plug-ins of the Eclipse component framework, from Eclipse 3.0 to 3.5. We investigated the empirical evidence of 5 of the 8 Lehmans' evolution laws. Our findings confirm the laws of continuing change, self regulation and continuing growth when metrics related to dependencies between the plug-ins and the Eclipse architecture are considered. We could not validate the conservation of familiarity law and the results for the declining quality law were inconclusive. Our overall observation is that the trends observed for constrained evolution are similar to those presented by earlier researches on the general evolution of software systems.

The results obtained so far and presented above are encouraging. However, a broader study covering additional laws and additional plug-in architectures is required to verify to what degree can the constrained evolution of plug-ins be described by means of the Lehman's laws.

Furthermore, Eclipse distinguishes between public APIs provided by the architecture ("good" APIs) and APIs labeled as internal ("bad" APIs). Use of internal APIs is discouraged by Eclipse because they are still evolving and unlike the public APIs neither binary nor contract compatibility of can be guaranteed for the internal APIs [6, 7]. However, sometimes third-party plug-in developers are forced to rely upon the internal APIs due to unique functionality provided by them. In these case the developers are confronted by the need to fix the incompatibilities brought about by the changes of the internal APIs in a new Eclipse release. Therefore, as an additional direction of future work we consider distinguishing between plug-ins depending solely on the "good" APIs as opposed to those dependent on the "bad" APIs and studying whether this difference between the two groups of plug-ins influences applicability of the Lehman's laws.

7. REFERENCES

[1] Eclipse Marketplace. http://marketplace.eclipse.org/, Consulted on June 01, 2010.

[2] Eclipse Wiki. http://en.wikipedia.org/wiki/eclipse_(software), Consulted on June 01, 2010.

[3] Sourceforge. http://sourceforge.net/, Consulted on June 01, 2010.

[4] E. J. Barry, C. F. Kemerer, and S. Slaughter:. How software process automation affects software evolution: a longitudinal empirical. *J. Softw. Maint. and Evol.: Res. and Pract.*, 19(1):1–31, February 2007.

[5] J. Businge. http://www.win.tue.nl/∼jbusinge/.

[6] J. des Rivières. Evolving Java-based APIs, October, 2007. Technical report, http://wiki.eclipse.org/Evolving_Java-based_APIs, Consulted on June 01, 2010.

[7] J. des Rivières. How to use the Eclipse API, May, 2001. Technical report, http://www.eclipse.org/articles/article.php?file=Article-API-Use/index.html, Consulted on June 01, 2010.

[8] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *Proc. Int'l Conf. Software Maintenance (ICSM)*, pages 131–142. IEEE Computer Society Press, March 2000.

[9] A. Israeli and D. G. Feitelson. The Linux kenel as a case study in software evolution. *The Journal of System and Software*, 83(3):485–501, March 2010.

[10] S. Koch. Software evolution in open source projects–a large-scale investigation. *J. Softw. Maint. and Evol.: Res. and Pract.*, 19(6):361–382, November 2007.

[11] M. M. Lehman. Evolution and Conservation in the large Program Life Cycle. *J. of Sys. and Software*, 1(3):213–221, 1980.

[12] M. M. Lehman. Rules and tools for software evolution planning and management. *Annals of Software Engineering*, 11(1):15–44, 2001.

[13] M. M. Lehman. Efficient Feature Selection via Analysis of Relevance and Redundancy. *J. of Machine Learning Research*, 5:1205–1224, 2004.

[14] M. M. Lehman and L. A. Belady. *Program evolution: processes of software change*. Academic Press, London, 1985.

[15] M. M. Lehman, D. E. Perry, and J. F. Ramil. On evidence supporting the feast hypothesis and laws of software evolution. In *Proc. Metrics'98*, pages 84–88. IEEE Computer Society Press, 1998.

[16] R. Martin. OO design: An analysis of dependencies, 1994. http://www.objectmentor.com/resources/articles/ood-metrc.pdf. Consulted on June 01, 2010.

[17] T. Mens, J. Fernández-Ramil, and S. Degrandsart. The evolution of Eclipse. In *Proc. 24th Int'l Conf. on Software Maintenance*, pages 386–395, October 2008.

[18] A. Serebrenik, S. A. Roubtsov, and M. van den Brand. D_n-based architecture assessment of Java open source software. In *ICPC2009*, pages 198–207. IEEE Computer Society, 2009.

[19] M. Wermelinger and Y. Yu. Analyzing the Evolution of Eclipse Plugins. In *Proc. Int'l Conf. Mining software repositories (MSR'08)*, pages 133–136. ACM, 2008.

[20] Wikipedia. Plug-in (computing), 2010. [Online; accessed 22-July-2010].

[21] G. Xie, J. Chen, and I. Neamtiu. Towards a better understanding of software evolution: An empirical study on open source software. In *Proc. 25th IEEE Int'l Conf. on Soft. Maint. (ICSM2009)*, pages 51–60, September 2007.

[22] Z. Xing and E. Stroulia. Api-evolution support with Diff-catchup. *IEEE Tran. Soft. Eng*, 33(12):818–836, 2007.

[23] C. Yang. Press Release: Eclipse Galileo, June, 2009. Technical report, http://www.eclipse.org/org/press-release/20090624_galileo.php, Consulted on June 01, 2010.