A Case Study of Software Product Line for Business Applications Changeability Prediction

Roško, Zdravko; Strahonja, Vjeran

Source / Izvornik: **Journal of Information and Organizational Sciences, 2014, 38, 145 - 160**

Journal article, Published version
Rad u časopisu, Objavljena verzija rada (izdavačev PDF)

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:543359

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-10-08**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

UDC 004.41:001.87
Original Scientific Paper

A Case Study of Software Product Line for Business Applications Changeability Prediction

Zdravko Roško *zrosko@gmail.hr*
Adriacom Software d.o.o.
Vodice, Croatia

Vjeran Strahonja *vjeran.strahonja@foi.hr*
Faculty of Organization and Informatics
University of Zagreb, Varaždin, Croatia

Abstract

The changeability, a sub-characteristic of maintainability, refers to the level of effort which is required to do modifications to a software product line (SPL) application component. Assuming dependencies between SPL application components and reference architecture implementation (a platform), this paper empirically investigates the relationship between 7 design metrics and changeability of 46 server components of a product line for business applications. In addition, we investigated the usefulness of Platform Responsibility (PR) metric as an indicator of product line component changeability. The results show that most of the design metrics are strongly related to the changeability of server component and also indicate statistically significant correlation between Maintainability Index (MI) and PR metric. The assessment is based on a case study of the implementation of the product line for business applications in a financial institution. The results show that PR metric can be used as good predictor of changeability in the software product line environment.

Keywords: Software product lines, changeability, maintainability index, metrics, reuse, reference architecture, platform responsibility.

1. Motivation

Software maintenance is the most expensive activity that consumes about 50 - 70 percent of development cost [23]. As software development processes in its life cycle through the phases of requirements analysis, design, implementation, testing and maintenance, the complexity and the cost of the software increases [28]. In spite of this fact, software organizations pay much more attention to software development than to maintenance. Most of methods and approaches are devoted to the development of new business applications, setting aside the maintenance of existing ones. For programmers, maintenance is "less attractive" than development; in fact, many existing business applications use old programming environments, file systems, etc., whereas they prefer working with new, visual environments. However, the same applications evolve and will continue to evolve along years and, to their regret, programmers devote 61% of their professional life to maintenance, and only 39% to new development [34].

There were many attempts to find ways to minimize maintenance cost by introducing better development approaches that can minimize the costly effects of change, simplify understanding of source code, facilitate early detection of faults, etc. One of the most successful approaches is Software Product Lines (SPL) approach, a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [12]. This approach and its techniques make a system better

maintainable as stated in [15]: "The same design techniques that lead to good reuse also lead to extensibility and maintainability over time".

Business applications are traditionally developed as standalone systems, each having specific architecture. As opposed to traditional approaches that focus on one application, the product line approach means a fundamental shift of focus from the individual system to the product line, i.e. a set of applications that rely on a common product line platform. A software platform is a set of software subsystems and interfaces that form a common structure from which a set of derivative products can be efficiently developed and produced [27]. A common rule of thumb found in literature is that a product lines approach will pay off only after the development of the software product platform and an initial set of products in the family. The relevant literature also claims that there is a significant reduction in costs associated with managing the evolution of the products when a product line approach is followed [24]. Due to the fact that any change in the platform can be relatively easily propagated to all of the product line members, the advantages of using a platform-based approach are even more significant.

The level of effort needed to maintain a software product line is related to the technical quality of the source code. Many software metrics have been proposed as indicators for technical quality of source code [18], [43]. Oman et al. proposed the Maintainability Index [39], [13] which attempts to objectively determine the maintainability of software system based upon on the characteristics of the source code. In the ISO/IEC 9126 standard (replaced by ISO/IEC 25010:2011), maintainability is seen as one of the 6 main characteristics of software product quality. IEEE (1990) defines maintainability as “The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment” [19]. The maintainability is further decomposed into the sub characteristics of analyzability, changeability, stability and testability [22]. Changeability characterizes the amount of effort to change a system (ISO/IEC 9126). In the context of software product lines, where many applications rely on a common platform, the technical quality of source code has its specifics comparing it with an ordinary system, since it is an important determinant for software product lines changeability. Changeability, the subject of this paper, is a key success factor in application areas such as business systems, in which applications are evolving at a rapid pace.

In [32] we have proposed Platform Responsibility (PR), a product line reference architecture coupling metric, to address the product component changeability prediction. In this paper, we further discuss the various issues arising when trying to assess the changeability of software product line components.

Object of the study. Object of this research study is the software product line server side application components and their changeability characteristics.

Purpose. The purpose of the study is to investigate the relationship between a number of design metrics and changeability of software product line components. Specifically, we would like to investigate metrics that can be used as good indicator of product line components changeability.

Perspective. This study targets two perspectives, one from the point of view of the researcher and the second of a developer, i.e. the researcher or developer would like to find out if there are any systematic differences in the changeability based on the design metrics of the individual product line component.

Quality focus. The main objective of this research is to determine whether there is a significant correlation between MI and PR metrics. In case there is a significant correlation between them, we will investigate the usefulness of PR metric as a predictor of product line components changeability, instead of using MI metric since it is too generic and yet not adapted to the product line environment specifics.

Context. The context of the experiment is a software product line for business applications in a financial institution. As a case for a survey we took 46 server side software components used by 9 different business applications within the product line.

2. Related Work

Changeability is related concept to maintainability and it is generally considered as its sub characteristic. Due to the fact that there are a number of different dimensions of maintainability, there exists a great deal of inconsistency in terminology. Matinlassi et al. proposed three maintainability abstraction levels; system, architecture and component [25]. The focus of our study is on component changeability, a sub characteristic of major importance for maintainability [36][8]. Several empirical studies have been carried out to investigate the maintainability of software product line artifacts [6], [9], [4], [25], [35] but a review of the literature fails to note significant research related to the changeability of SPL components in the context of external and internal dependencies.

There are different approaches used for the assessment of changeability.

The assumption that changeability is a sub characteristic of maintainability consequently opens the door to application of maintainability metrics to the assessment of changeability. Generally, maintainability metrics are based code level metrics, on change impact analysis or on design metrics.

We are using the last approach.

Riaz et al. found in their study that the most successful maintainability metrics are based on size, complexity, and coupling [31].

Ingram and Riddle [21] used six metrics: LOC (Lines of Code), DIT (Depth of Inheritance Tree), WMC (Weighted Methods per Class), CBO (Coupling Between Objects) and McCabe's Cyclomatic complexity (CC) to demonstrate a correlation between software size/complexity and change proneness. In their study change tendency was measured as the number of files changed for each revision. The result of the study suggests that classes with the highest CBO were the most likely to change.

Chaumun et al. used experiment which showed a high correlation, across systems and across changes, between changeability and the access to a class by other classes through method invocation or variable access. This relationship refers to the so called afferent coupling (Ca), the number of classes in other packages that depend upon classes within the package. The more a class is used through invocation of its methods and outside references to its variables, the larger the impact of a change to such a class. On the other hand, no result could support the hypothesis that the depth of the inheritance tree has some influence on changeability [3].

Wilkie and Kitchenham [41] tested the usefulness of the CBO (Coupling Between Objects) metric in predicting the classes that are likely to be affected by a change. Their object of study was a multimedia conferencing system which consists of 25.000 lines of code. The metrics used in their study were: CBO, WMC (Weighted Methods per Class) and number of functions per class. The research results show that the CBO metric is useful in identifying the most change prone classes. Also the same metric does not identify the classes likely to experience ripple effect changes.

Aldekoa [17] extended the Maintainability Index where the maintainability index of each features is measured. The metric is based on the average of the McCabe's Cyclomatic Complexity value [26] which directly measures the number of linearly independent paths through a program's source code.

Tizzei et al. [35] tested positive and negative change impact of component and aspect based design on Product Line Architecture (PLA) stability. They concluded that the combination of aspects and components supports the design of high cohesive and loosely coupled PLAs and improve modularity.

André van der Hoek et al. [17] developed a class of closely related metrics that specifically target product line architectures such as metrics base on *Provided Service Utilization (PSU)* and *Required Service Utilization (RSU)*. The metrics explicitly take into account the context in which individual architectural elements are placed and are based on the concept of service utilization. However, those metrics are based on concept of service that is defined as any public accessible resource but do not consider dependencies on external third party components and its influence on the quality of the application components.

In our study, we focus on the changeability assessment based on the product line component dependencies. The dependency may exist between product line components and internally owned (e.g. SPL platform) or externally owned (e.g. Spring) components. We presume that changeability of a component is better and more under control of internal development when the number of external dependencies that exist for a component is lower. This assumption is along the lines of the recommendations of good modular design, which seeks to achieve a high degree of internal cohesion, and the less external communication (coupling).

These external and internal dependencies among components and between components and the product line platform can serve to assess the impact of change as changes can propagate from one component to other components through the dependencies.

3. Experimental Design

Here, we provide some background on the product line for business applications investigated in this study. Furthermore, we describe goals, hypotheses, dependent and independent variables. The objects of the study are, as stated above, not selected randomly; they are the server side application components of the software product line for business applications in a financial institution.

3.1. System Investigated

The source of data we have collected for this study was a product line for business applications in a financial institution. The data we have collected include two cumulative versions of the 9 applications and its corresponding server components. First version included 43 server components, while the second version included 46 components, having 3 additional components added to the product line. The selection of the product line was influenced by its technical complexity and the fact that the author has been involved in its development. The product line is a Java-based group of 9 applications based on the shared platform. It is a closed-source system built with several external components which include: Apache POI, which is used for reading and writing Microsoft Office files, iText for reading and writing PDF files, Apache Shiro for user authentication with Active Directory, Apache uploads, for uploading files to a server, Aktiviti for process engine and Jasperreports used for the generation of business reports.

To study the product line component changeability, the maintenance data were limited to the Java source code which was collected from Subversion Edge source code repository [14]. In the repository the changes made can be viewed at the source code level hence they can be identified and manually assigned to the different program components. Maintenance tasks carried out on non-java application artifacts were out of the scope of this study. This study analyzes only the Java code.

3.2. Goals, Variables and Hypotheses

The main goal of this study is to determine the design metrics that can be used as good indicators of product lines component changeability. In order to determine that, we collected historical maintenance data from a product line for business applications. The objective of this study is to determine the differences in changeability for software product lines components using the product line platform as a base. Motivation for this study is a need to understand the differences in changeability among components within the product line. One objective of introducing software product line is to provide the environment for better changeability. In order to support the better changeability, it is important to understand what differences can be expected within the product line, and explain them in order to improve the product line changeability.

3.3. Dependent Variables

In this paper, Platform Responsibility (PR) [32] and Maintainability Index (MI) [39] are both used as the dependent variables to quantify product line components changeability. Our goal here is to show that both, PR and MI measure the changeability of components, but from different perspectives. The objective of the research is to verify the hypothesis that PR is more suitable dependent variable for studying the relationship between design metric and component changeability in software product line environment.

Figure 1 shows the elements the PR metric is calculated from. When introduced at [32], PR metric is analyzed within the Distance framework of measurement theory [30] and framework based on desirable properties which serves guidance provided to define proper measures for specific problem [10]. These frameworks ensure that the metrics developed using these guidelines are tested to be valid and that they can be used as measurement instruments [32].

PR is a combination of three coupling metrics: D3 - number of distinct references outside the platform that depend upon classes within the platform, D4 - the number of distinct references inside the component that depend upon classes within environment (e.g. Java RTE), D5 - number of distinct references inside the component that depend upon classes within external components. It measures the "level of responsibility" of a reference architecture implementation (a platform) to communicate to the external components needed by application component in order to provide business logic to an application. The more the component delegates a communication to the external components the more it is protected from frequent changes to the external third party components. The three coupling metrics are combined and used to calculate the PR value, stated by equation 1.

$$\boldsymbol{PR} = \left(1 - \frac{D4 + D5}{D3 + D4 + D5}\right) * 100 \tag{1}$$

The range for this metric is from 0 to 100. The larger the PR, the more maintainable is the product line component. Components with a PR less than 50 are more difficult to maintain than components with PR between 50 and 100 which have reasonable maintainability.

We can calculate a total PR for product line platform by taking in account all of the components through the following equation:

$$\boldsymbol{PR} = \sum_{n=1}^{n} \left(1 - \frac{D4_i + D5_i}{D3_i + D4_i + D5_i}\right) * 100 \tag{2}$$

where n is the number of application components using the reference architecture implementation (a platform).

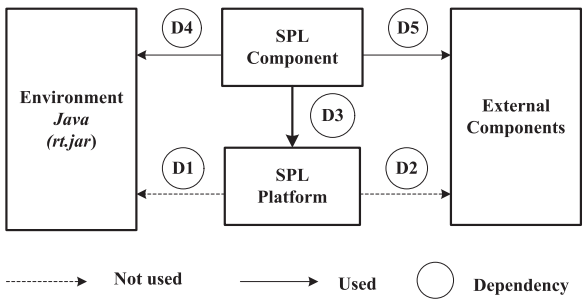

Figure 1. Platform Responsibility Metrics [32]

MI, our second dependent variable, is a combination of widely-used and commonly-available measures [4, 28, 33, 39, 40, 42]. MI is a complex calculation involving a number of different metrics: Cumulative Halstead Effort of all the parts of a class, Number of methods in class, Total Cyclomatic Complexity of all the methods in the class, Total Number of Java Statements in class [38]. The metrics are combined into parts and then used to calculate the MI value, stated by equation 2.

$$\textbf{\textit{EffortPart}} = 3.42 * log(HEFF/NOMT)$$
$$\textbf{\textit{CyclomaticPart}} = 0.23 * log(TCC/NOMT)$$
$$\textbf{\textit{LinesPart}} = 16.2 * log(NOS/NOMT)$$

$$MI = 171 - effortPart - cyclomaticPart - linesPart \quad (3)$$

The larger the MI, the more maintainable is the product line component. Components with a MI less than 65 are difficult to maintain, components between 65 and 85 have reasonable maintainability and those with MI above 85 have excellent maintainability. Since MI measurement is not a trivial task, we used JHawk 5.1 tool to measure MI for each product line component.

Maintainability	MI Score
Highly maintainable	>85
Moderately maintainable	>65 and ≤ 85
Difficult to maintain	≤ 65

Table 1. Maintainability rules for MI [13]

The empirically derived coefficients from actual usage data for MI are shown Table 1. MI is a complexity metric to indicate the code that is difficult to maintain. MI metric is used by individual programmers while maintaining a code to determine if the maintainability of given software is improving or diminishing. Although several variants of the MI equations have evolved over time, in this study we use the version originally proposed by Oman and Hagemeister. Since many studies have shown that the MI model was often overly sensitive to the comment metric, we use the version, where the comment equation portion was omitted to limit the contribution of comments in MI.

3.4. Independent Variables

The selection of the independent variables includes 7 object oriented design metrics of size, complexity, coupling and inheritance. The definition of those object oriented design metrics is given in Table 2. The metrics selection is based on the previous research results which have indicated that ACC, ADIT, AMC, and AWMC have statistically significant effects on maintainability [42]. The selection is based on the metrics selection criteria, a set of criteria for choosing suitable metric set [1]. Also, the metric selection is based on distinct metrics characteristics which we have identified and used to omit the metrics which measure the same thing. Finally the metrics selection was limited by available tools we could have used for this case study. The goal of our metrics selection criteria was to avoid measuring too much or measuring too little and not gaining sufficient insight into the desired objective. These metrics are commonly used and have been validated [7].

Metric	Description
ABD	Average block depth
ACC	Average cyclomatic complexity
ADIT	Average depth of inheritance hierarchy
AMC	Average number of methods per class
ALCM	Average lines of code per method
AWMC	Average weighted methods per class
NMETH	Number of methods

Table 2. Definition of design metrics

Some metrics such as CC, DIT, MC, LCM, and WMC are originally defined at the class level. However, this study is performed at the product line component level. Therefore, those metrics may not be directly used as independent variables. In order to use them in this study, for each such metric, its mean among classes is calculated and used as an independent variable. Naming of those metrics is prefixed by an "A", for example, the average CC metric is named ACC, when used at the component rather than at the class level.

3.5. Hypotheses

The hypotheses that relate metrics to product line components maintainability are listed and described in Table 3. The relationship column (+/-) indicates the direction of correlation between each metric and changeability (PR), where "+" means positive and "–" means negative correlation. Different authors have measured correlation between source code metrics and maintainability [2, 16, 42]. To date various methods have been developed and introduced to measure maintainability, however, there are a software product lines specifics, since they heavily rely on the platform used by its components. It will be addressed by this study. This experiment introduces a new relation measurement, the correlation between PR and MI metrics, since they potentially measure the same thing, but from different perspective, one (MI) from a generic perspective, the other (PR) from the product line specific perspective.

Metric	+/-	Description
MI	+	Measure the same thing as PR
ABD	-	Component become more complex
ACC	-	Component become more complex
ADIT	-	Component coupling increases
AMC	-	Component become more complex
ALCM	-	Component become more complex
AWMC	-	Control flows are more complex
NMETH	-	Number of faults and difficulty increases

Table 3. The hypotheses

4. Execution

The experiment was based on data (Java source code) collected through the two major releases of the product line in a financial institution. To ensure the data validity, the Java source code metrics of collected data are measured twice, once using CodePro Analytix™ , tool [20], and again with JHawk tool [37], but just for the metrics which could be measured by both of the tools. These tools measure and report on key quality indicators in a body of Java source code. In cases the results from the tools were different, we used JHawk results as representative, since the JHawk tool was also used to measure the MI metrics which depends on some of the other measured metrics.

4.1. Sample

The product line reference architecture implementation (platform) together with 9 applications has all together 161.376 lines of Java source code without comments (LOC). The rest of the source code has been written by using Transact-SQL, XML, HTML, CSS, Java Script languages. In this study we analyze the Java source code used by server side business application components, consisting of 33.139 LOC, and of 27.252 Java statements (NOS). Table 4 presents the 46 components and its data that was collected from Subversion Edge source code repository.

Component	Metrics											
	D3	D4	D5	PR	MI	ABD	ACC	ADIT	AMC	ALCM	AWMC	NMETH
SC1	13	6	0	68,42	106,66	1,45	1,73	4,00	5,50	10,90	10,00	11
SC2	104	79	0	56,83	104,22	1,21	1,30	5,77	6,61	8,88	8,67	119
SC3	8	5	0	61,54	104,90	1,38	1,46	3,00	6,50	10,61	9,50	13
SC4	25	17	0	59,52	96,39	1,53	1,58	4,20	2,40	13,30	3,80	12
SC5	42	38	0	52,50	88,42	1,62	2,47	4,61	2,92	20,44	7,62	38
SC6	30	34	0	46,88	97,44	1,60	1,79	4,20	15,60	16,64	28,00	78
SC7	26	16	0	61,90	94,63	1,27	1,73	3,60	4,40	16,27	7,60	22
SC8	10	4	0	71,43	112,42	1,18	1,19	3,00	8,00	6,93	9,50	16
SC9	17	15	0	53,13	101,15	1,47	1,59	3,66	19,67	13,94	31,33	59
SC10	16	8	0	66,67	97,52	1,51	1,59	4,33	12,33	14,72	20,00	37
SC11	6	5	0	54,55	100,87	1,50	1,50	3,50	10,00	13,09	15,00	20
SC12	60	47	0	56,07	96,84	1,41	2,06	5,63	5,91	14,53	12,55	65
SC13	81	60	0	57,45	107,64	1,23	1,32	5,25	8,81	8,99	11,69	141
SC14	29	19	0	60,42	109,40	1,25	1,33	3,75	15,75	9,22	17,75	63
SC15	8	5	0	61,54	99,00	1,42	1,71	4,00	3,50	12,28	6,00	7
SC16	30	18	0	62,50	97,50	1,55	1,76	4,20	7,60	14,23	13,40	38
SC17	64	44	0	59,26	99,71	1,43	1,65	5,59	9,10	13,56	15,00	91
SC18	19	12	0	61,29	96,35	1,57	1,86	4,00	4,67	16,14	8,67	14
SC19	18	14	0	56,25	100,79	1,45	1,75	5,00	5,00	11,25	9,00	20
SC20	38	29	0	56,72	114,35	1,12	1,30	6,29	5,88	6,51	7,76	100
SC21	16	13	0	55,17	99,03	1,53	2,19	4,66	8,67	16,00	19,67	26
SC22	8	5	0	61,54	102,63	1,46	1,53	4,00	7,50	12,53	11,50	15
SC23	11	4	0	73,33	105,20	1,30	1,31	3,00	6,50	9,92	8,50	13
SC24	10	5	0	66,67	104,45	1,39	1,47	4,66	5,00	11,06	7,33	15
SC25	26	19	0	57,78	100,93	1,47	1,63	4,50	6,33	12,68	10,50	38
SC26	31	21	0	59,62	104,11	1,34	1,51	4,20	8,20	10,95	12,60	41
SC27	31	24	0	56,36	104,26	1,33	1,43	4,00	8,40	10,80	12,20	42
SC28	15	13	0	53,57	98,72	1,53	1,91	4,00	14,33	15,13	28,67	43
SC29	78	57	0	57,78	95,56	1,52	1,86	4,53	7,00	15,73	13,00	91
SC30	20	10	0	66,67	98,15	1,52	1,52	3,83	3,50	14,28	5,33	21
SC31	13	13	0	50,00	97,50	1,54	1,92	4,33	8,00	15,37	15,33	24
SC32	29	20	0	59,18	105,75	1,22	1,40	5,33	5,83	8,65	8,17	35
SC33	33	26	0	55,93	99,11	1,54	1,89	3,60	22,80	15,53	43,80	114
SC34	17	14	0	54,84	103,02	1,43	1,43	4,33	12,33	11,91	17,67	37
SC35	9	6	0	60,00	102,78	1,52	1,84	3,00	9,50	12,57	17,50	19
SC36	142	116	0	55,04	98,42	1,49	1,87	5,65	9,81	14,23	18,73	255
SC37	10	9	0	52,63	99,21	1,45	1,64	3,50	5,50	13,63	9,00	11
SC38	15	11	0	57,69	104,52	1,33	1,44	3,66	6,00	10,44	8,67	18
SC39	6	4	0	60,00	93,54	1,64	2,29	4,00	8,50	18,52	19,50	17
SC40	27	28	0	49,09	98,56	1,43	1,68	5,42	5,29	12,91	8,86	37
SC41	6	4	0	60,00	107,45	1,33	1,50	3,50	3,00	7,33	4,50	6
SC42	55	24	0	69,62	102,23	1,56	1,97	3,85	4,64	12,43	9,43	65
SC43	12	8	0	60,00	101,16	1,50	1,50	4,33	4,00	11,66	6,00	12
SC44	8	4	0	66,67	109,75	1,27	1,27	4,00	5,50	7,90	7,00	11
SC45	6	1	0	85,71	119,01	1,30	1,40	4,00	5,00	6,00	7,50	10
SC46	11	5	0	68,75	109,43	1,36	1,55	4,50	5,50	7,90	9,00	11
Total	1289	939	0	2758,49	4690,68	65,45	75,62	195,96	356,79	568,49	592,78	1991
Average	28	20	0	59,97	101,97	1,42	1,64	4,26	7,76	12,36	12,89	43

Table 4. Sample data (V2)

The data we use comprise two separate time series: the stream of release V2, and a corresponding stream of release V3. The data to be analyzed are historical data from 2012 and 2013 and comprise the data for all independent and two dependent variables that we observe in this study. The mentioned source code repository is used by developers to commit the

changes at least on monthly bases. Java source code is developed using Eclipse IDE for Java Developers which includes Subversion Edge client to work with Java source code stored in the repository. The data from repository were collected in two separate Eclipse workspaces, each for the corresponding release (V2, V3). Each workspace is used separately to collect the source code metrics. For measurement of the D3, D4 and D5 dependency metrics we have used CodePro Analytix™ Eclipse plug-in, and for measuring the rest of metrics we have used JHawk tool. The collected metrics for both time series (V2, V3) are entered into IBM SPSS® Statistics statistical software for further processing.

It can be observed that components vary in size in terms of the number of methods (NMETH), and also vary in coupling in terms of number of platform references (D3) and number of Java environment references (D4).

Table 5 provides a summary of the maintenance tasks and the impacts on the product line components between the two releases of the product line applications. It shows that most of these maintenance tasks were new components development.

Maintenance type	Components affected	Classes affected	LOC affected
Add	35	68	7314
Change	33	61	101
Delete	11	0	87

Table 5. The maintenance tasks and their impact on components

The components in Table 5 refers to the server side application units, each consisting of component interface (facade), zero or more business objects and data access objects. The results from the table show that most of these maintenance tasks were addition of the new functionalities. Different maintenance tasks and changes in the source code have different impact on the product line components. Some changes are localized and they do not affect other components (facade) of the product line whereas others affect a number of components (data access objects, business objects).

5. Analysis

This section presents the statistical analysis of the data we have gathered. In this study we focus on investigating the capability of Platform Responsibility (PR) metric to serve as indicator of application component changeability. As indicated earlier in Section 2 (**related work**) and in [11], [5], many studies have already investigated the OO and coupling (Ca, Ce) metrics for this purpose, however, the metrics to measure coupling between product line application components and their reference architecture implementation (platform) has not been used as a predictor of product components changeability. After the analysis, we assume that changeability of a product line application component is better assessed in terms of coupling metrics as coupling metrics measure the degree to which components depend on product line reference architecture implementation (platform), external third party components and to the Java environment. These dependencies may be used to assess the impact of change as changes of an external third party component may impact application component classes through the dependencies. A class belonging to a component that is coupled to other external components is sensitive to changes and as a result it becomes more difficult to maintain. Changeability of a class (and hence a product line component it belongs to) depends on the number of changes required and the impacts of these changes related to other dependent classes. More application component class dependencies on the platform component classes rather than on other third party or environment classes, suggests a better changeability. Therefore, the proposed metrics (PR) that capture the dependencies among application classes and other external components are expected to have the ability of predicting changeability of a software product line for business applications.

5.1. Descriptive Statistics

Table 4 shows the descriptive statistics for two accumulated versions of the server business components. The selection of these metrics is based on the fact that they measure different structural properties of a component: size, coupling, complexity, and inheritance, and since they refine classical object-oriented (Chidamber and Kemerer metrics) [3], which are well established and based on sound measurement theory.

Column "Skewness" is a measure of the asymmetry that shows whether the data distribution is skewed. Column "Kurtosis" is a measure of the "peakedness" that shows whether the data are peaked or flat relative to a normal distribution.

The low mean for NMETH and AWMC indicate that there are a small number of components which are having very high number of methods. Their distribution and the distribution of AMC metrics show high variations across the product line, which may reflect the lack of development experience of the programmers involved in those components. Figure 3 and 4 show the PR and MI metric frequency distribution. Both of the metrics distributions form a symmetrical, bell-shaped pattern, which approximates a normal distribution of the data. Second, the mean, mod and median for both metrics are equal and are located at the center of the distribution. Third, most of the values are clustered around the center of the distribution.

Metric	Minimum value	Maximum value	Mean value	Standard deviation	Skewness	Kurtosis
PR	46.87	85.71	59.95	6.88	1.17	2.79
MI	88.42	119.01	101.95	5.82	0.58	0.62
ABD	1.11	1.7	1.41	0.12	-0.38	-0.58
ACC	1.19	2.47	1.63	0.27	0.89	0.71
ADIT	3	6.29	4.29	0.78	0.54	-0.16
AMC	2.4	22.8	7.75	4.19	1.70	3.18
ALCM	6	20.44	12.26	3.24	0.06	-0.29
AWMC	3.8	44.2	12.86	7.68	1.99	4.79
NMETH	6	255	45.09	48.26	2.45	7.17

Table 6. Descriptive statistics of the server components

Figure 2 visualizes the overall level of MI and PR metrics for 46 server components under study. Roughly, we can say that there is an implicit relationship between MI and PR, i.e., a component with a higher MI level is more likely to have a higher PR.

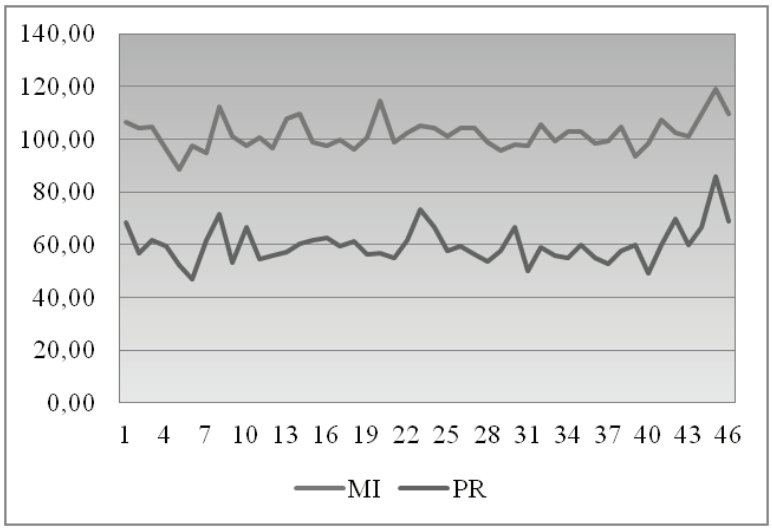

Figure 2. Relationship of MI to PR metric

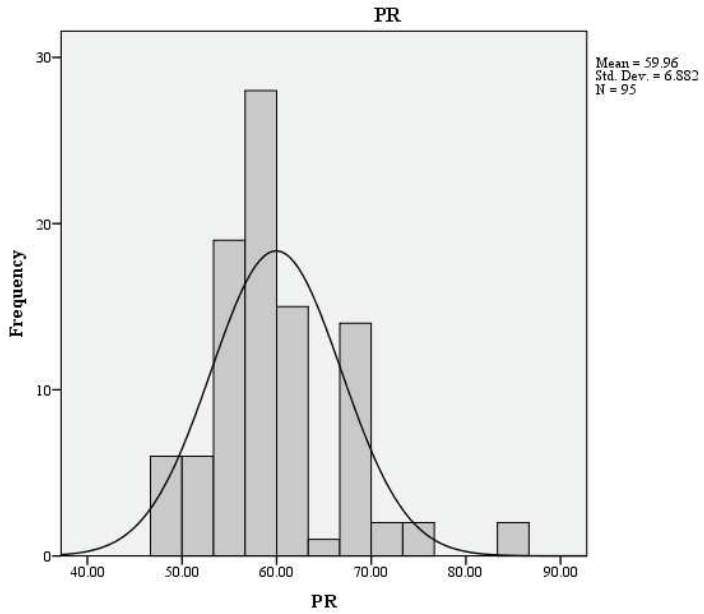

Figure 3. PR metric histogram

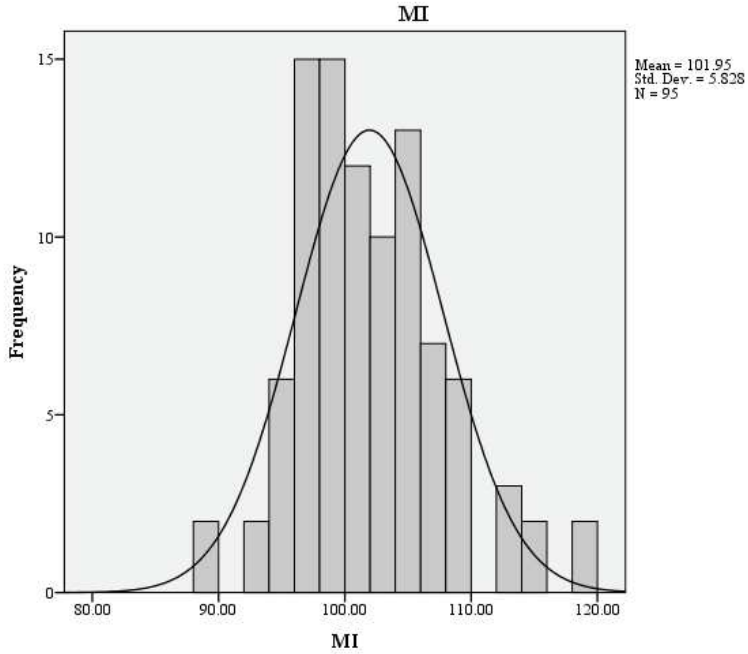

Figure 4. MI metric histogram

Figure 5, the scatter plot, shows the relationship between PR and MI variables. The scatter plot shows an upward trend, a positive correlation, in which a direct relationship exists between PR and MI variables. That means that an increase in PR is related to an increase in MI, and a decrease in PR is related to decrease in MI. The figure also shows that the homoscedasticity assumption is met, because the variability of the PR variable, pretty much remains relatively constant from one MI value to the next. The two outliers (PR=86, MI=118 and PR=86, MI=119) shown at the upper right corner at figure 5, are due to the rare event of components which interface is designed but their business logic was never implemented. These components are candidates to be dropped from the product line and could be excluded from the analysis.

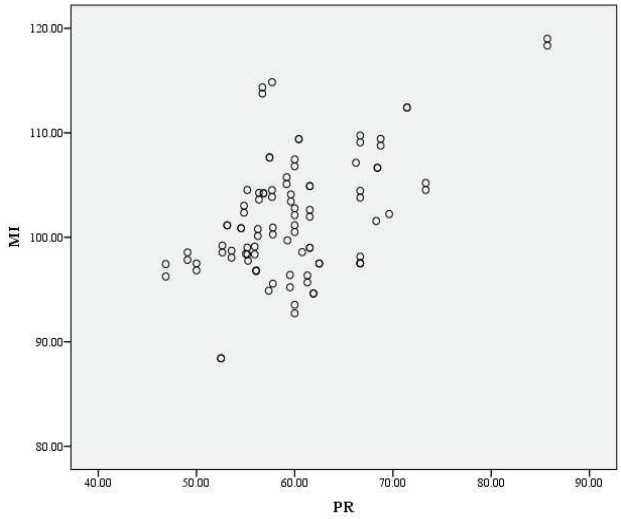

Figure 5. MI and PR metric data distribution

5.2. Hypothesis Testing

Correlation technique was used to analyze the relationship between design metrics and both PR and MI metrics, and also between MI and PR metrics them self. Since our goal was to prove that PR metric can be used as a changeability predictor, we needed to find out if there is a positive correlation between MI, widely-used measure, and PR, newly proposed measure. The significance of the correlation was tested at 99% confidence level (i.e. p-level ≤ 0.01) and at 95% confidence level (i.e. p-level ≤ 0.05). The results obtained by applying this analysis are given in table 7 and 8, where ** and * values indicate statistically significant correlations.

Among the 7 design metrics used in this study, all were found to have significantly correlated negative effect on changeability (PR). ALCM (Average Lines of Code per Method) shows the highest negative correlation followed by ACC and AWMC. It is most probably the result of tendency that class methods with more lines of code than average, are using more of the external components than average methods, which makes the component less changeable. Therefore, the hypotheses related to those metrics are supported.

Table 8 shows the correlations between design metrics and MI, where only 3 metrics (ABD, ACC, and ALCM) were found to have statistically significant negative effect on maintainability (MI). ALCM (average lines of code per method) shows the highest negative correlation followed by ABD and ACC. The correlation between Maintenance Index (MI) and Platform Responsibility (PR) is moderate, r = **0.526**** which we consider strong in the context of the software product lines environment. The correlation indicates that PR can be used for the same purpose as MI in case we are predicting product line component changeability. Therefore, as the values of these metrics increase, the changeability of the components and hence the product line decreases.

Metric	Pearson correlation coefficient (r)
ABD	-.296**
ACC	-.353**
ADIT	-.304**
AMC	-.313**
ALCM	-.477**
AWMC	-.374**
NMETH	-.277**

**. Correlation is significant at the .01 level (2-tailed)

Table 7. PR and metrics correlations

Metric	Pearson correlation coefficient (r)
ABD	-.777**
ACC	-.751**
ADIT	.016
AMC	-.050
ALCM	-.941**
AWMC	-.239*
NMETH	-.072

**. Correlation is significant at the .01 level (2-tailed)
*. Correlation is significant at the .05 level (2-tailed)

Table 8. MI and metrics correlations

6. Interpretation

Based on the results of correlation analysis performed, this section discusses the implications of selected design metrics on the changeability of product line components. The implications can provide decision support for reference architecture designers in the process of significant architectural decision making at the relatively early phases of software product line development.

The results are interpreted with respect to the hypotheses stated in section 3.5. All hypotheses are tested using Pearson correlation. It can be concluded that there are significant differences among the components, depending on their design metrics characteristics. This is true for all the hypotheses. Furthermore, there is significant correlation between MI and PR metrics, and thus the exceptional usage of PR instead of MI in software product line environment can be suggested.

The design metrics used in this study, were found to have significantly correlated negative effect on changeability, which is in line with what was reported in [29].

The same metrics are reported to be used as successful predictors for source code maintainability, acording to systematic review [31].

A strong relation between import coupling metrics (efferent) and maintainability characteristics has been reported by Dagpinar et al. [16]. Import coupling considers interactions of the class or component that is using the functionality of other classes or the component.

Our results also suggest that metrics related to application size, complexity and coupling may be used as maintainability predictors.

The results of this study, where we used the PR dependency metrics, also measure the import coupling for server component which is using the functionality of the platform (D3), environment (D4) and external components (D5). The results are consistent with those of the aforementioned study and suggest that changeability of product line components depends on source code design characteristics.

There are limits of this study to generalize the results of our experiment to industrial practice. Threats to the conclusion validity of the results are that the number of samples is low, in particular the number of maintenance changes. However, we believe that our proposed work can be seen as a first step toward the needed empirical research on the relation between source code design metrics and external quality of product line components. The specific business environment, programming language, developers experience and technical environment are not representative of the population we want to generalize to, but the threats are reduced by making the experimental environment as realistic as possible.

7. Conclusion and Future Work

In this paper we have investigated the relationships between 7 design metrics and software product line component changeability, a sub-characteristic of maintainability, based on a

software product line implementation in a financial institution. The metrics used here, measure coupling, size, inheritance and complexity of a product line components used by the 9 applications from the product line. The Maintainability Index (MI) was used as the dependent variable together with recently proposed Platform Responsibility (PR) metric. Our goal was to find out if PR metric can be used instead of the MI metric in case the study is carried over within software product line environment. Pearson correlation analysis results indicate a statistically significant correlation between PR and MI metrics, and also between most of the individual metrics and component changeability represented by those metrics. We have also found the ability of the design metrics to predict components changeability, when design metrics are used together. The results of this research support the idea to use the PR metric as predictor of changeability in the software product line environment. The correlation between MI and PR is interesting because it is much easier to measure PR than MI metric. This indicates that PR metric may be used more often in the future as a predictor for product line component changeability. The major limitation of this study was the sample size and the specific technical environment which was used to develop the product line in a financial institution. This study contributes preliminary and novel empirical knowledge about the relationships between some design metrics and product line components changeability. In the future work we will employee classical linear regression to investigate the relationship between design metrics and changeability of software product line components. Also, the future work will include the analysis of influences of individual design metrics.

References

[1] Abreu, F.B., Carapu\cca, R.: Object-oriented software engineering: Measuring and controlling the development process. In: proceedings of the 4th International Conference on Software Quality. (1994)

[2] Aggarwal, K.K., Singh, Y., Chhabra, J.K.: An integrated measure of software maintainability. In: Reliability and maintainability symposium, 2002. Proceedings. Annual. pp. 235–241. IEEE (2002)

[3] Ajrnal Chaumun, M., Kabaili, H., Keller, R.K., Lustman, F., Saint-Denis, G.: Design properties and object-oriented software changeability. In: Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European. pp. 45–54. IEEE (2000)

[4] Aldekoa, G., Trujillo, S., Mendieta, G.S., Díaz, O.: Experience Measuring Maintainability in Software Product Lines. In: JISBD. pp. 173–182. Citeseer (2006)

[5] Ayalew, Y., Mguni, K.: An Assessment of Changeability of Open Source Software. Computer and Information Science. 6 (3), p68 (2013)

[6] Bagheri, E., Gasevic, D.: Assessing the maintainability of software product line feature models using structural metrics. Software Quality Journal. 19 (3), 579–612 (2011)

[7] Belsley, D.A., Kuh, E., Welsch, R.E.: Regression diagnostics: Identifying influential data and sources of collinearity. John Wiley & Sons (2005)

[8] Bengtsson, P., Bosch, J.: Architecture level prediction of software maintenance. In: Software Maintenance and Reengineering, 1999. Proceedings of the Third European Conference on. pp. 139–147. (1999)

[9] Briand, L.C., Bunse, C., Daly, J.W.: A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. Software Engineering, IEEE Transactions on. 27 (6), 513–530 (2001)

[10] Briand, L.C., Morasca, S., Basili, V.R.: Property-based software engineering measurement. Software Engineering, IEEE Transactions on. 22 (1), 68–86 (1996)

[11] Burrows, R., Garcia, A., Taïani, F.: Coupling metrics for aspect-oriented programming: A systematic review of maintainability studies. In: Evaluation of Novel Approaches to Software Engineering. pp. 277–290. Springer (2010)

[12] Clements, P., Northrop, L.: Software product lines: Practices and Patterns. Addison-Wesley Boston (2002)

[13] Coleman, D., Ash, D., Lowther, B., Oman, P.: Using metrics to evaluate software system maintainability. Computer. 27 (8), 44–49 (1994)

[14] CollabNet: Subversion Edge,Release: 1.3.0.

[15] Coplien, J., Hoffman, D., Weiss, D.: Commonality and variability in software engineering. Software, IEEE. 15 (6), 37–45 (1998)

[16] Dagpinar, M., Jahnke, J.H.: Predicting maintainability with object-oriented metrics-an empirical comparison. In: Proceedings of the 10th Working Conference on Reverse Engineering (WCRE). pp. 155–164. (2003)

[17] Van Der Hoek, A., Dincel, E., Medvidovic, N.: Using service utilization metrics to assess the structure of product line architectures. In: Software Metrics Symposium, 2003. Proceedings. Ninth International. pp. 298–308. IEEE (2003)

[18] Fenton, N.E., Pfleeger, S.L.: Software metrics: a rigorous and practical approach. PWS Publishing Co. (1998)

[19] Geraci, A., Katki, F., McMonegal, L., Meyer, B., Lane, J., Wilson, P., Radatz, J., Yee, M., Porteous, H., Springsteel, F.: IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries. IEEE Press (1991)

[20] Google, Inc: CodePro Analytix. Google, Inc (2011)

[21] Ingram, C., Riddle, S.: Linking software design metrics to component change-proneness. In: Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics. pp. 31–37. ACM (2011)

[22] International Organization for Standardization: ISO/IEC 9126-1: Software engineering - product quality - part 1: Quality model, (2001)

[23] Jalote, P.: A Concise Introduction to Software Engineering. Springer (2008)

[24] Kang, K.C., Sugumaran, V., Park, S.: Applied software product line engineering. CRC press (2009)

[25] Mari, M., Eila, N.: The impact of maintainability on component-based software systems. In: Euromicro Conference, 2003. Proceedings. 29th. pp. 25–32. IEEE (2003)

[26] McCabe, T.J.: A complexity measure. Software Engineering, IEEE Transactions on. (4), 308–320 (1976)

[27] Meyer, M.H.: The power of product platforms. Simon and Schuster (1997)

[28] Misra, S.C.: Modeling design/coding factors that drive maintainability of software systems. Software Quality Journal. 13 (3), 297–320 (2005)

[29] Misra, S.C.: Modeling design/coding factors that drive maintainability of software systems. Software Quality Journal. 13 (3), 297–320 (2005)

[30] Poels, G., Dedene, G.: DISTANCE: A framework for software measure construction. DTEW Research Report 9937. 1–47 (1999)

[31] Riaz, M., Mendes, E., Tempero, E.: A systematic review of software maintainability prediction and metrics. In: Proceedings of the 2009 3rd

International Symposium on Empirical Software Engineering and Measurement. pp. 367–377. IEEE Computer Society (2009)

[32] Rosko, Z.: Assesing the Responsibility of Software Product Line Platform Framework for Business Applications. Presented at the CECIIS-2013, (2013)

[33] Schach, S.R., Jin, B., Wright, D.R., Heller, G.Z., Offutt, A.J.: Maintainability of the Linux kernel. IEE Proceedings-Software. 149 (1), 18–23 (2002)

[34] Singer, J.: Practices of software maintenance. In: Software Maintenance, 1998. Proceedings., International Conference on. pp. 139–145. IEEE (1998)

[35] Tizzei, L.P., Dias, M., Rubira, C.M., Garcia, A., Lee, J.: Components meet aspects: Assessing design stability of a software product line. Information and Software Technology. 53 (2), 121–136 (2011)

[36] Vigder, M.: The evolution, maintenance, and management of component-based systems. Component-Based Software Engineering: Putting the Pieces Together. 527–539 (2001)

[37] Virtual Machinery: JHawk. (2013)

[38] Virtual Machinery: JHawk 5.1 Documentation-Metrics Guide, (2012)

[39] Welker, K.D., Oman, P.W.: Software maintainability metrics models in practice. Crosstalk, Journal of Defense Software Engineering. 8 (11), 19–23 (1995)

[40] Welker, K.D., Oman, P.W., Atkinson, G.G.: Development and application of an automated source code maintainability index. Journal of Software Maintenance: Research and Practice. 9 (3), 127–159 (1997)

[41] Wilkie, F.G., Kitchenham, B.A.: Coupling measures and change ripples in C++ application software. Journal of Systems and Software. 52 (2), 157–164 (2000)

[42] Zhou, Y., Xu, B.: Predicting the maintainability of open source software using design metrics. Wuhan University Journal of Natural Sciences. 13 (1), 14–20 (2008)

[43] Zuse, H.: A framework of software measurement. Walter de Gruyter (1998)