Entertainment Computing 4 (2013) 131–142

Contents lists available at SciVerse ScienceDirect

Entertainment Computing

journal homepage: ees.elsevier.com/entcom

Building and mining a repository of design pattern instances: Practical and research benefits

Apostolos Ampatzoglou, Olia Michou, Ioannis Stamelos

Department of Informatics, Aristotle University, Aristotle University Campus, PO Address 54124, Thessaloniki, Greece

ARTICLE INFO

Article history:
Received 10 February 2012
Revised 15 June 2012
Accepted 1 October 2012
Available online 12 October 2012

Keywords:
Software engineering
Computer games
Design patterns
Repository

ABSTRACT

Design patterns are well-known design solutions that are reported to produce substantial benefits with respect to software quality. However, to our knowledge there are no scientific efforts on gathering information on software projects that use design patterns. This paper introduces a web repository of design patterns instances that have been used in open source projects. The usefulness of such a repository lies in the provision of a base of knowledge, where developers can identify reusable components and researchers can find a mined data set. Currently, 141 open source projects have been considered and more than 4500 pattern instances have been found and recorded in the database of the repository. The evaluation of the repository has been performed from an academic and a practical point of view. The results suggest that the repository can be useful for both experienced and inexperienced users. However, the benefits of using the repository are more significant for inexperienced users.

1. Introduction

A repository is a shared database about engineered artefacts, such as software, documents and information systems [10]. In the field of software, several repositories of open-source projects that can be used as means of knowledge and experience sharing are available. Such repositories (e.g. Sourceforge, CodeHouse), provide data to the fast developing open source research area [44] and to software engineering researchers in general. Even though such repositories contain a great amount of information, sometimes it is difficult to browse, due to scalability problems, data integrity problems, etc. Reuse repositories and especially open source repositories are the basic infrastructure for software reuse, due to its significant economic impact. Nevertheless, they have introduced difficulties in finding and locating reusable software artefacts.

Additionally, design patterns are software engineering techniques that are considered to aid in software system development. In [22], the authors attempted to identify whether there are common problems among different system designs and if common solutions might be introduced. Moreover, a qualitative evaluation of the proposed solutions implies that the use of design patterns leads to adaptability and extensibility, as each design pattern supports independence of possible future changes [22]. The possibility of creating reusable components from design patterns is discussed in [7,26], where the authors introduced the idea of componentizing and reusing design patterns. Furthermore, in [3], the authors investigated the possibility of selecting design pattern instances as reuse granules, in white box reuse.

Despite the fact that during the last years the research on design patterns appears to flourish, no repositories concerning design pattern instances in software projects have been identified in the literature. In this paper we present a web repository (Percerons – PattErn Repository and Components Extracted from OpeN Source software) of object-oriented design pattern instances and a web tool which is used to mine it [18]. Currently, the repository contains more than 4500 pattern instances, extracted from 141 Java open source projects. Up to this point, we have investigated and recorded all stable computer games identified in a well known source code repository[1]. The selection of games was based on the fact that game development is an active topic in OSS communities [44] but game developers are more probable to write code without prior design activities [27]. Thus, a repository that will enhance the design process might prove useful. The benefits from such an attempt are expected to assist both practitioners and academics in the following ways:

- Practitioners
  - The re-users will be given a smaller search domain than forges, that will provide application specific components

E-mail addresses: apamp@csd.auth.gr (A. Ampatzoglou), stamelos@csd.auth.gr (I. Stamelos).

[1] www.sourceforge.net

1875-9521/$ - see front matter
http://dx.doi.org/10.1016/j.entcom.2012.10.002

- The components that are recorded in the database of the repository involve design pattern participants[2] and therefore, the rationale of their design is documented
- Similarly, some internal quality attributes of the extracted components, such as coupling, cohesion and complexity, are expected to improve compared to any other non-pattern component [2]

• Academics
- The software engineering researchers can easily obtain datasets on design patterns, without having to mine vast source code forges.

The user of Percerons is expected to have two possible motivations for accessing the search engine. (a) He/She is interested in implementing a game requirement, but does not know where to look. Percerons will provide a set of classes that implement the complete or a part of the desired functionality. (b) He/She is interested in retrieving several instances of a specific design patterns for any possible reason (education, research etc.). In order to validate the abovementioned benefits we conducted an experiment, in order to evaluate the usefulness of the repository from the practitioner's and researcher's point of view. In the next section, the research state of the art on reuse repositories is presented. Section 3, presents an overview on game development and game design. In Section 4, the structure of repository is presented. Section 5 deals with the empirical validation of the repository, from both an academic and a practical point of view. In Sections 6 and 7 discussion and threats to validity are provided. Finally, Section 8 presents conclusions resulting from this work and proposals for future research.

2. Reuse repositories

Current trends in software reuse indicate that the design and implementation of open source repositories follow a rather holistic approach, since not only programmers, but software managers and users are also influenced. Repositories should support the 'produce → manage → consume' cycle and its stakeholders [6,15]. In [15], a generic architecture for software repositories which contains modules that satisfy the needs of all involved parties is presented. Accordingly there is a production module (for component producers), a management module (for reuse managers), a consumption module (for component reusers or consumers) and an infrastructure module (providing common services to the other modules). More specifically, in case of the management module, the repository should provide the means to manage the reuse process at the organizational level by enabling reuse managers to observe specific quantifiable metrics associated with the reuse of the available components. Consequently, a set of numerous requirements should be satisfied for the repository to be considered as successful. Dominant role play search, retrieval and the emerging technology of semantic web that is used for component description are discussed in [14,32,40]. Moreover, requirements such as specification publication, user notification, version management, dependency management should also be included in the set.

As the main challenge that a component approach will meet is dealing with change, the substitutability of its parts can be achieved only if components are properly specified. Faceted classification as a representational method for classifying reusable artefacts is an interesting approach, according to which it organizes the search space in search-related facets [17,35]. A faceted classification provides a vocabulary for cataloguing reusable components and an organization mechanism for these components. The effectiveness can be assessed through various metrics including the well known precision (high precision means that a high fraction of retrieved instances are relevant) and recall (high recall means that only few relevant components are not retrieved), search effort, user satisfaction, and ease of use [20,31,36]. The role of the semantic technology is prominent as metadata are ideal for component description through the facets. In literature in the domain of software design, quality and reuse, several approaches have been found concerning software engineering repositories. On the other hand none of these approaches deals explicitly with design patterns.

Firstly, in [1] a widely accessible repository of software development artefacts (e.g. code, models and test cases) is proposed. The repository would enable software developers and researchers to assess software engineering tools and techniques. In [16], a repository of FLOSS data and analyses is proposed. This repository named OSSmole is a collaborative project which collects, shares and stores compatible data and analyses of FLOSS development for research purposes. It aims to mine FLOSS source code repositories and provide resulting data and summary analysis at open source products.

ROSE is another software engineering repository based on open source products. It is developed to aid students and educators during software engineering courses or lessons on open source software development [29]. Another type of software engineering repository is introduced in [24], where among others a proof-of-concept prototype metric repository is described. In [33,47], the importance of software repositories aiming at reuse is proved as the first introduces a knowledge-based repository scheme for storing and retrieving business components, whereas the latter explores component classification and retrieval methods with an overriding concern for automation. In [30], the authors introduced a benchmark repository of faulty and correct software that would enable unification of diverse experimental results regarding software testing techniques. Agora system described in [42] is a software prototype that allows automatic discovery and retrieval of software components from the web and their characterization with introspection. The object of this work is to create an automatically generated and indexed worldwide database of software products classified by component model. Agora combines introspection with Web search engines to reduce the costs of bringing software to, and finding components in, the software marketplace.

Finally, Maracatu [23] is another approach for retrieving software modules from CVS repositories. The Maracatu service has the following modules: (a) a CVS module for accessing the repositories for reusable components, (b) an Analyser module for analysing the code to determine if it is suitable for indexing, (c) a Download module for downloading (checking out) the source code, and (d) a Search module for searching using queries. This functionality is provided with an Eclipse plug-in.

3. Game development

We have chosen to validate the usefulness of the proposed repository in the domain of game development, because computer game design is one of the most modern and fast growing trends in computer science [37]. Additionally, it is a common sense that games play a very important role in modern societies concerning economy and lifestyle.

The game industry is now considered to be one of the most powerful in the business spectrum [21,50]. The fact that computer games currently rival television and films in both market size and cultural impact [21] can justify why game development is

[2] As pattern participant we mean every class that plays a specific role in a design pattern. A definition of the pattern participants is given in [25].

considered a placeholder in current lifestyle. Additionally, it is estimated that 90% of US households have rented or purchased at least one video game, and that young people of the country spend an average of twenty (20) minutes per day playing video games.

In [4], the authors suggest that the research on software engineering for computer games is rapidly increasing. In the same study it is suggested that the design phase of the development lifecycle is an interesting research topic, which needs further investigation. Additionally, [41] suggests that game communities could benefit from reuse opportunities that derive from the activity noticed in open-source games.

Until the middle of 1990s, game developers did not aim to produce reusable code since every program has been written from scratch in assembly language [38]. Later, reusable coding has proven to be one of the most important issues in game development because games became far more complex and their production process more time consuming. In order to alleviate this problem, frameworks and game engines have been created. A framework is a collection of classes that can be widely reused and integrated with other components [39,46]. Usually they implement mechanisms that occur in many games, such as input handling, file handling (texture, models, audio etc.), 3D rendering etc. Game engines are programs that provide developers the potential to design game levels, handle player and opposition behaviour, by using scripting languages and powerful GUIs. As it is easily understood, if frameworks and game engines are "well-structured", they can be maintained without extreme effort and be transformed so that they can fit as many game genres as possible. In order to achieve this expectation, game developers should use software engineering techniques and methodologies.

The evaluation of object-oriented design pattern application in computer games has been investigated in [2,34]. In [2], the authors have examined the way object-oriented design patterns can affect the structure and the maintainability of a game by analyzing existing systems. The results suggested that patterns can reduce complexity and coupling of a game, increase cohesion of the code but as a side effect the project size, i.e. lines of code, is increased. In addition to that in [34] there was an attempt of creating a game that was based on patterns. The results suggested that design patterns should be considered an efficient way of properly achieve abstractions and decoupling in games.

Furthermore, a different approach for game design patterns is presented in two papers. More specifically, the authors propose patterns on game mechanics and not on the design of game. In [13], the authors suggest that design patterns are proven to support the design, analysis and comparison of games as an alternative to the need of developing a common language, concepts and terminology for games. Since games are commonly big projects that require collaboration among staff with different expertises, patterns should be viewed as a tool to overcome communication differences in an effective and efficient way. Similarly, in [28], the authors propose several design patterns that describe mechanisms of augmented reality systems, which could be used at games.

4. Repository design

The proposed design pattern repository (Percerons) is a software engineering repository that was established so as to provide a body of knowledge that would help developers to identify and reuse "common solutions" on "common problems". Additionally, it aims to help software engineer researchers by giving them instant access to a mined data set that consists of multiple instances of design patterns.

4.1. Repository scheme and data description

Since the effectiveness of the repository depends on the ease that the developers and researchers can satisfy their requirements for code reuse and pattern retrieval, the design of the repository proves to be critical. The repository consists of patterns found in various categories of open-source software. A complete description of a design pattern should include characteristics of its constituent elements. The abstraction hierarchy is based on [47] where the authors propose a classification and coding (C&C) scheme and knowledge-based business repository. Fig. 1 shows the abstraction hierarchy of the design pattern. Each pattern has one or more roles where each role is assigned to one class and each class has a set of methods and attributes.

In order to reduce the initial large set of patterns, structured identifiers could be used. Consequently, a description of each project should be included. The description should include the name, version, programming language, production phase and subcategory of the software. For example, if we consider games as software category, subcategory is the game genre (e.g. arcade games). The above structured identifiers represent the structured information at the pattern level. Moreover, as naming a design pattern lets software engineers to design at a higher level of abstraction [22], a name is also assigned to each pattern. The element role is considered a descriptor facet for the unstructured information regarding design patterns. Role is considered as the role that a class plays at the particular pattern and can be used for scientific purposes. Names and roles are according to [22]. The roles assigned in each design pattern are presented in Appendix A. Methods and attributes will be used as the final selection criteria, when the user will have to decide among design patterns that fulfill equally the other searching criteria.

4.2. Method for storing data in the repository

The method for populating the data of the repository consists of eight steps, as described below:

- Identify a number of projects that fulfill certain selection criteria, for each category
- Perform pattern detection for every selected project. Pattern detection has been performed with two tools [43,45]
- For every pattern identify all pattern participating classes
- Tabulate data
- Insert data on the database of the repository
- Reverse engineer the Java binary files in order to extract the methods and the properties of the class
- Link the data of the repository to source code and method/properties information.

Source code repositories provide a great variety of software projects. From these projects we have selected those that fulfilled some criteria, in order for our pattern extraction and storing mechanism to work. More specifically, due to limitation of the pattern detection tools that we have used, the software under study must be written in Java. Additionally, the software that is used for the identification of all the classes that participate in a design pattern has to use a Java .jar file. Consequently, we have only selected projects which are written in Java and provide an executable .jar file. Additionally, concerning games, the results that have been obtained were pre-processed and game engines have been excluded from the selected set.

Finally, the database of the repository consisted of 141 open source Java games. All the abovementioned software has been investigated for occurrences of 10 design patterns and more than 4500 pattern instances have been identified and recorded. The

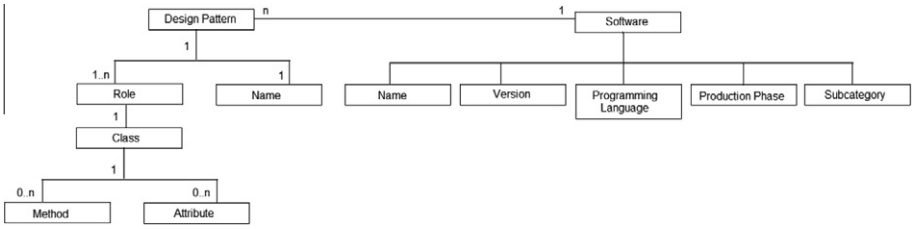

Fig. 1. Abstraction hierarchy of a Design Pattern.

patterns that have been considered are Factory Method, Prototype, Singleton, Adapter, Composite, Decorator, Observer, State, Strategy, Template Method and Visitor. Information on design pattern definitions and structure can be found in [22].

4.3. Retrieving data from the repository

The web page interface provides the user with several retrieve options:

(a) Find a Component from Search Engine
(b) Find a Pattern
(c) Find a Project

Option (a) allows the user to provide the search engine with three keywords, software category and subcategory, and browse it in order to identify class names that contain any of the keywords. If software category and software subcategory variables are not blank, the search space is limited to projects that fulfill the aforementioned criteria. Option (a) is more probable to be used by developers who are interested in identifying a class that provides certain functionality and simultaneously is pattern participant.

On the other hand, (b) and (c) options are more probable to be used by researchers who might want to find pattern instances on open source projects or to identify all the patterns that are implemented in open source projects. Using Percerons is expected to increase and enhance empirical studies concerning patterns and open-source, in the sense that it provides a large mined dataset that is easily accessible. Indicatory subjects of such research attempts are patterns application frequency [5], class participant change proneness [8,12,19] and the effect of design pattern is software quality [2,48].

5. Empirical evaluation method

This section deals with the evaluation of the proposed repository. In order to validate the usefulness of the Percerons project we have conducted an experiment. The experiment evaluates the repository from two perspectives, namely (a) research perspective and (b) development perspective. The experiment of this study has been conducted according to the guidelines described in [9,49]. The steps to be followed are: experiment definition, experiment planning, experiment operation and experiment interpretation. In the next sections, an overview of the steps followed in our formal experiment is presented.

5.1. Experiment definition

According to [9], the experiment definition involves experiment *motivation*, *purpose*, *object*, *perspective*, *domain* and *scope*. The motivation of our research is the enhancement of the design process. The purpose of the experiment is to evaluate Percerons from a practitioner's and a researcher's perspective. Therefore, the following research questions have arisen. The first research question (RQ_1) that the experiment attempts to answer can be described by the following scenario: "A developer wants to find a piece of code that implements a specific requirement. He comes up with one or more keywords that describe the desired functionality and according to them the repository retrieves a list of candidate sets of classes. At which degree is the objective correctly performed?" The research questions of this study are summarized below:

RQ1: Does the use of Percerons result in greater correctness of an objective?

RQ_2: Does the use of Percerons result in decreasing the time needed for completing an objective?

RQ_3: Does the use of Percerons acquire less effort for an objective?

RQ_4: Does the use of Percerons succeed in greater user satisfaction?

The objects of the study are two repositories, one source code repository where a user might identify non-pattern related sets of classes and another repository where recommended sets of classes involve design pattern participating classes. Information on the subjects and projects, i.e. domain and scope of the experiment, are thoroughly discussed later in the paper.

5.2. Experiment planning

The planning phase of the experiment is considered to be crucial to the overall experiment success and includes aspects such as design, criteria and measurements [9]. Considering the above, the experiments' hypotheses are defined, the experiment projects and subjects are described and the methods of comparison are presented.

Define Hypothesis – As mentioned earlier the paper aims at the construction and the evaluation of the design pattern repository. More specifically, several aspects are going to be investigated, namely (a) objective correctness, (b) objective completion time, (c) search effort and (d) user satisfaction. The precision and the recall of the repository are not investigated in this study. According to the experiment plan, eight (8) null hypotheses have been set for every research question. In total, thirty-two (32) hypotheses have investigated in order to explore the aforementioned research questions. In Table 1, we provide a sample of the tested hypotheses. More specifically, we present 16 hypotheses that deal with the correctness of the objective, the time needed for completing the objective, the search effort and user satisfaction from a developer's point of view. Similarly, sixteen (16) additional hypotheses have been set from a researcher's point of view, but they are omitted from the manuscript in order to improve its readability. In Table 1, let NDRS be a result set of the search engine other than the Percerons repository and DRS to be the result set of the search engine of Percerons.

More specifically, the hypothesis $H_{0(1)}$ to $H_{0(16)}$ refer to the research questions from a developer's point of view, taking into consideration the experience factor. More specifically, the sample has been split, w.r.t. experience according to COCOMO81 [11]. A more detailed description on this activity is presented later in the paper.

Table 1
Research hypotheses.

RQ_1	RQ_2	RQ_3	RQ_4
$H_{0(1)}$: the correctness of adapting code retrieved from NDRS is similar to the correctness of adapting the code retrieved from DRS	$H_{0(5)}$: the code adaptation time for the set of classes retrieved from NDRS is similar to the adaptation time for code retrieved from DRS	$H_{0(9)}$: NDRS and DRS have been obtained with similar effort from a developer's point of view	$H_{0(13)}$: the user satisfaction from NDRS is similar to user satisfaction from DRS, from a developer's point of view
$H_{0(2)}$: the correctness of adapting code retrieved from NDRS is similar to the correctness of adapting the code retrieved from DRS from a highly experienced developer's point of view	$H_{0(6)}$: the code adaptation time for the set of classes retrieved from NDRS is similar to the adaptation time for code retrieved from DRS from a highly experienced developer's point of view	$H_{0(10)}$: NDRS and DRS have been obtained with similar effort from a highly experienced developer's point of view	$H_{0(14)}$: the user satisfaction from NDRS is similar to user satisfaction from DRS, from a highly experienced developer's point of view
$H_{0(3)}$: the correctness of adapting code retrieved from NDRS is similar to the correctness of adapting the code retrieved from DRS from a developer with medium experience point of view	$H_{0(7)}$: the code adaptation time for the set of classes retrieved from NDRS is similar to the adaptation time for code retrieved from DRS from a developer with medium experience point of view	$H_{0(11)}$: NDRS and DRS have been obtained with similar effort from a developer with medium experience point of view	$H_{0(15)}$: the user satisfaction from NDRS is similar to user satisfaction from DRS, from a developer with medium experience point of view
$H_{0(4)}$: the correctness of adapting code retrieved from NDRS is similar to the correctness of adapting the code retrieved from DRS from a low experienced developer's point of view	$H_{0(8)}$: the code adaptation time for the set of classes retrieved from NDRS is similar to the adaptation time for code retrieved from DRS from a low experienced developer's point of view	$H_{0(12)}$: NDRS and DRS have been obtained with similar effort from a low experienced developer's point of view	$H_{0(16)}$: the user satisfaction from NDRS is similar to user satisfaction from DRS, from a low experienced developer's point of view

For example, hypothesis $H_{0(1)}$ is set and investigated in order to examine the benefits of using the pattern repository, w.r.t the correctness of a development objective (RQ_1), regardless of the developer's experience. On the other hand, hypotheses $H_{0(2)}$ and $H_{0(3)}$ also examines RQ_1 but from an experienced and an inexperienced developer's point of view, respectively.

Experiment Subjects and Projects – In the experiment, two different component search plans have been used, one NDRS and one DRS. Each plan aimed at the following four objectives (two development objectives and two research objectives):

> *O_1: Development Objective.* Identify a component that deals with a game that implements the behaviour of weapons
> *O_2: Development Objective.* Identify a component that deals with the attributes of hockey players. Hockey players are divided into two main categories (goalkeepers and field players). These categories share some attributes, but simultaneously each category has some different attributes
> *O_3: Research Objective.* Identify the average number of design pattern instances in stable board games
> *O_4: Research Objective.* Identify ten design pattern instances that have been added in the $(k+1)$ version of a game that did not exist in the (k) version of the game

Table 2
Experiment variables.

Occupation	Researcher/Developer
Experience	Research in Patterns/OO development (in years)
Experience Categorical	Low, Medium or High Experience
Objective Correctness	Correctness for meeting the objective
Objective Time	Time required for meeting the objective
Process	NDRS or DRS used for meeting the objective
Objective Effort	Effort of using the repository
Objective Satisfaction	Satisfaction from using the repository

A more detailed description of the objectives is provided in Appendix B. At this stage it is necessary to clarify that all objectives satisfy several constraints, in order for the experiment environment to be controlled. For instance, code development has been performed under the supervision of one author. Additionally, the subjects have been asked to identify a piece of code that is adapted on a pre-existing code, i.e. a main method. Thus, the subjects were not able to deflect from the experiment scenario. The NDRS procedure provided the subjects, the alternative to use any code search engine and forge they desired. Concerning development objectives, every subject who was not familiar to a code search engine was randomly assigned to one of the following engines, namely (a) Google©, (b) Merobase©, and (c) Koders©. The selection of (b) and (c) is based on the fact that these search engines are ranked first when searching for *software component search engine* and *source code search engine*, respectively in a well known web search engine, i.e. Google. At this point it is necessary to clarify that these engines do not provide pattern-based code. Concerning research objectives, the forge where a subject would search for projects that might involve pattern instances would be SourceForge©. Design pattern recognition for NDRS subjects were conducted through a reverse engineering tool was provided [45]. On the other hand, the DRS procedure allowed the corresponding subject to only use the Percerons search engines. All subjects were allowed to use Eclipse for compiling and completing the development objectives.

The experiment has been conducted with forty (40) professional developers and twenty (20) researchers. Each subject completed one objective with a NDRS or with a DRS process.

Methods of comparison – On the completion of the experiment a dataset with sixty (60) rows and eight (8) columns has been created. The dataset which has been created after gathering both researcher's and developer's questionnaires involve numerical, ordinal and categorical data. The variables that have been processed are mentioned in Table 2.

In order to test the hypotheses described in Table 1 we conducted independent sample *t*-tests and Mann–Whitney *U* tests. Independent sample *t*-test is a parametric test that is used to compare two independent sample means, that is, the comparison of two different methods. Mann–Whitney *U* test on the other hand, is non-parametric and is used to compare two independent sample means but in cases that the data are ordinal[3]. In order to explore the hypotheses $H_{0(1)}$–$H_{0(8)}$ and $H_{0(17)}$–$H_{0(24)}$, sixteen (16) independent sample *t*-test have been performed. Concerning the hypotheses $H_{0(9)}$–$H_{0(16)}$ and $H_{0(25)}$–$H_{0(32)}$ sixteen (16) Mann–Whitney *U* tests have been conducted.

[3] Parametric tests are used when data follow normal distribution, whereas non-parametric are used when data do not follow normal distribution.

Table 3
Descriptive statistics on subjects experience.

Group	Count	%
Researchers (process = ALL && experience = HIGH)	12	60.00
Researchers (process = ALL && experience = LOW)	8	40.00
Developers (process = ALL && experience = HIGH)	21	52.50
Developers (process = ALL && experience = LOW)	19	47.50
Researchers (process = NDRS && experience = HIGH)	6	60.00
Researchers (process = NDRS && experience = LOW)	4	40.00
Developers (process = NDRS && experience = HIGH)	10	52.63
Developers (process = NDRS && experience = LOW)	9	47.37
Researchers (process = DRS && experience = HIGH)	6	60.00
Researchers (process = DRS && experience = LOW)	4	40.00
Developers (process = DRS && experience = HIGH)	11	52.38
Developers (process = DRS && experience = LOW)	10	47.62
Researchers (process = ALL && experience = HIGH)	12	60.00
Researchers (process = ALL && experience = LOW)	8	40.00
Developers (process = ALL && experience = HIGH)	21	52.50
Developers (process = ALL && experience = LOW)	19	47.50
Researchers (process = NDRS && experience = HIGH)	6	60.00
Researchers (process = NDRS && experience = LOW)	4	40.00
	Avg. experience	Std. dev
Researchers	1.950	1.099
Developers	2.675	1.470

More specifically, in order to execute a test that explores $H_{0(1)}$ we created a two-column view of the dataset. The first column represents the *Objective Correctness* variable, the second column represents the *Process* variable, and the third represents the *Occupation* variable. The view has been created by filtering the sample in order to isolate cases that correspond to software developers (using *Occupation* variable). In the *t*-test the testing variable is *Correctness* and the grouping variable is *Process*. Concerning $H_{0(2)}$ we had to demarcate developers with respect to their level of experience. Consequently, we had to recode the variable *Experience* into a new variable named *Experience_Categorical*[4]. Finally, we filtered the dataset, in order to isolate the rows that correspond to experienced developers and we performed a *t*-test. The testing variable is *Correctness* and the grouping variable is *Process*. Similarly, we conducted independent *t*-tests for the hypotheses $H_{0(3)}$–$H_{0(8)}$ and $H_{0(17)}$–$H_{0(24)}$.

[4] Recoding the values of the variable Experience_Categorical was based on the categories described in [11]. However, we have chosen to create two experience levels, rather than three, because of the size of our dataset. Thus, developers/researchers with experience less than three years, are considered as inexperienced, whereas developers/researchers with three years of experience and more are considered as experienced.

Table 4
Descriptive statistics on subject performance/opinion on experiment objectives.

Group	Mean/Mode[a] value			
	Objective completion time	Objective Correctness	Objective completion effort	User satisfaction from tools
Total research objective	43.250	82.500	2	2
Total development objective	42.425	88.880	4	4
Researchers (process = ALL && experience = HIGH)	45.625	89.380	2	2
Researchers (process = ALL && experience = LOW)	41.667	77.920	3	–
Developers (process = ALL && experience = HIGH)	32.631	92.370	–	4
Developers (process = ALL && experience = LOW)	51.286	85.710	4	4
Researchers (process = NDRS && experience = HIGH)	57.500	91.250	–	2
Researchers (process = NDRS && experience = LOW)	48.333	66.670	–	2
Developers (process = NDRS && experience = HIGH)	33.888	90.000	1	4
Developers (process = NDRS && experience = LOW)	61.700	80.500	4	–
Researchers (process = DRS && experience = HIGH)	33.750	87.500	2	–
Researchers (process = DRS && experience = LOW)	35.000	89.170	–	–
Developers (process = DRS && experience = HIGH)	31.500	94.500	–	4
Developers (process = DRS && experience = LOW)	41.818	90.450	–	–
Total research objective	43.250	82.500	2	2
Total development objective	42.425	88.880	4	4
Researchers (process = ALL && experience = HIGH)	45.625	89.380	2	2
Researchers (process = ALL && experience = LOW)	41.667	77.920	3	–
Developers (process = ALL && experience = HIGH)	32.631	92.370	–	4
Developers (process = ALL && experience = LOW)	51.286	85.710	4	4

[a] The (–) symbol under the mode value, represents that at least two values are equally ranked as mode value.

Concerning Mann–Whitney *U* test, the $H_{0(9)}$ has been tested through the following procedure. Firstly, we created a two-column view of the dataset. The first column represented the *Effort* whereas the second column represented the *Process*. In the Mann–Whitney *U* test the testing variable was *Effort* and the grouping variable was *Process*. Similarly, we conducted independent *t*-tests for the rest hypotheses that involved ordinal variables.

Table 5
Cross tabulation for ordinal data.

	Cross tabulation						Pearson x^2 Asymp. Sig. (2-sided)
	NDRS		DRS		Total		
	Count	Exp. Count	Count	Exp. Count	Count	Exp. Count	
Effort Researcher Objective	Very Easy	0	1.5	3	1.5	3	0.387
	Easy	1	1	1	1	2	
	Neutral	4	3.5	3	3.5	7	
	Hard	2	2	2	2	4	
	Very Hard	3	2	1	2	4	
	Total	10	10	10	10	20	
Effort Developer Objective	Very Easy	3	3.3	4	3.7	7	0.852
	Easy	5	3.8	3	4.2	8	
	Neutral	2	2.8	4	3.2	6	
	Hard	3	3.3	4	3.7	7	
	Very Hard	6	5.7	6	6.3	12	
	Total	19	19	21	21	40	
Satisfaction Researcher Objective	Very Unsatisfied	1	0.5	0	0.5	1	0.090
	Unsatisfied	0	0.5	1	0.5	1	
	Neutral	6	3.5	1	3.5	7	
	Satisfied	1	2.5	4	2.5	5	
	Very Satisfied	2	3	4	3	6	
	Total	10	10	10	10	20	
Satisfaction Developer Objective	Very Unsatisfied	2	1.4	1	1.6	3	0.402
	Unsatisfied	5	3.3	2	3.7	7	
	Neutral	2	3.8	6	4.2	8	
	Satisfied	3	3.8	5	4.2	8	
	Very Satisfied	7	6.7	7	7.3	14	
	Total	19	19	21	21	40	

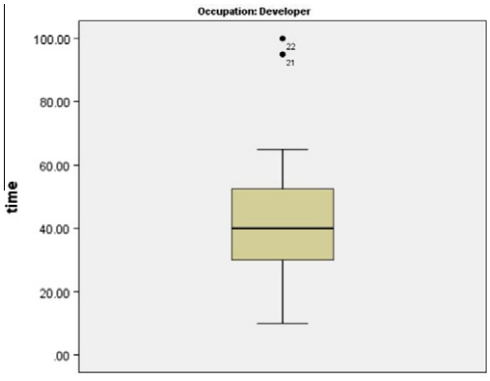

Fig. 2. Boxplot on developer objective completion time.

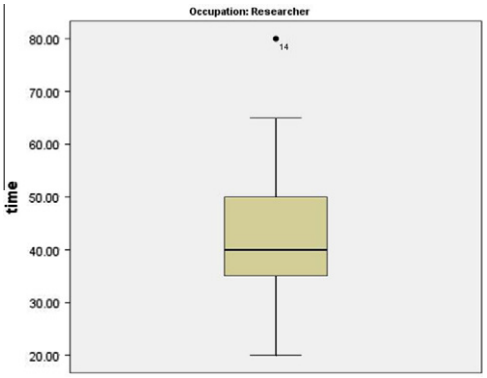

Fig. 3. Boxplot on research objective completion time.

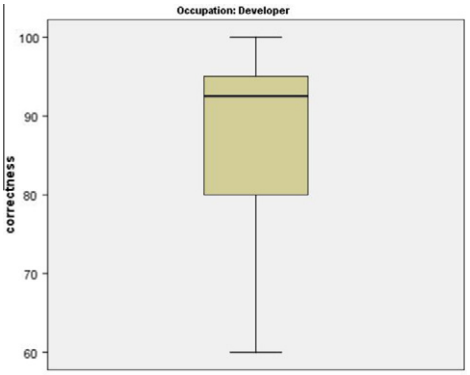

Fig. 4. Boxplot on developer objective correctness.

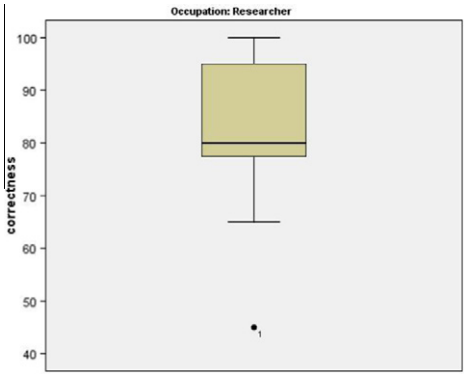

Fig. 5. Boxplot on research objective correctness.

5.3. Experiment operation

According to [49], the experiment operation phase is consisted of three steps: preparation, execution and analysis. The preparation phase of the experiment has included an interview with the experiment participants in order to evenly distribute them among groups w.r.t. their experience in patterns, in object-oriented programming and research experience. In this phase the values of the process variable was filled in the dataset.

During experiment execution, the researchers have collected and validated data concerning all time related variables. On the completion of the experiment, the researchers corrected the answers of the experiment participants (concerning correctness variable) and surveyed the participants in order to extract the values of the variables concerning satisfaction and ease of use. The guidelines used while evaluating objective correctness are presented in Appendix B.

5.4. Experiment interpretation

The interpretation of the experiment results is composed of interpretation context, extrapolation and impact [9]. The research interpretation is presented in Section 7 along with discussion on the study extrapolation and impact.

Table 6
Independent *t*-tests – Mann–Whitney *U* tests for developers.

		NDRS		DRS			
		N	Mean	*N*	Mean	*t*	Sig. (2-tailed)
All Developers	Time Developer Objective	19	48.526	21	36.904	1.97	0.06*
	Correctness Developer Objective	19	85.000	21	92.380	−2.52	0.02**
		N	Mean Rank	*N*	Mean Rank	*W*	Sig. (2-tailed)
	Effort Developer Objective	19	20.450	21	20.550	388.5	0.98
	Satisfaction Developer Objective	19	19.470	21	21.430	370.0	0.59
		N	Mean	*N*	Mean	*t*	Sig. (2-tailed)
Experience = LOW	Time Developer Objective	10	61.700	11	41.818	2.69	0.02**
	Correctness Developer Objective	10	80.500	11	90.450	−2.28	0.04**
		N	Mean Rank	*N*	Mean Rank	*W*	Sig. (2-tailed)
	Effort Developer Objective	10	11.750	11	10.320	113.5	0.58
	Satisfaction Developer Objective	10	9.900	11	12.000	99.0	0.43
		N	Mean	*N*	Mean	*t*	Sig. (2-tailed)
Experience = HIGH	Time Developer Objective	9	33.888	10	31.500	0.36	0.72
	Correctness Developer Objective	9	90.000	10	94.000	−1.32	0.20
		N	Mean Rank	*N*	Mean Rank	*W*	Sig. (2-tailed)
	Effort Developer Objective	9	9.330	10	10.600	84.00	0.61
	Satisfaction Developer Objective	9	10.060	10	9.950	99.50	0.97

* Significant at 0.10 level.
** Significant at 0.05 level.

6. Results

This section of the paper, reports on the results of statistically analyzing the dataset of the experiment. The analysis phase of our study has employed descriptive statistics, cross tabulation, Pearson x^2 tests, boxplots, independent sample *t*-tests and Mann–Whitney *U* tests. The section is divided into two sections, one concerning statistics that describe the dataset and the subjects' performance and another that deals with hypotheses testing.

6.1. Descriptive statistics

This section of the paper aims at presenting results that describe the characteristics of experiment subjects and the characteristics of the two subject groups, i.e. DRS and NDRS. Such results are presented in Table 3. Additionally, descriptive statistics on the experiment results are reported on Table 4.

In order to further analyze the results of Table 3, we performed two additional statistical methods, cross tabulation, for categorical variables and boxplots for numerical variables. The cross tabulation analysis, displays the joint distribution of two variables, as well as the Pearson x^2 test that ensures the distribution of the results. The corresponding results on subjects' effort and satisfaction are shown in Table 5. Furthermore, boxplots graphically represent measures of central tendency (mean and median) and measures of dispersion (range, inter-quartile range, minimum and maximum). The corresponding boxplots are depicted in Figs. 2–5.

6.2. Hypothesis testing

This section of the paper aims at presenting the results that derive from the investigation of the hypotheses which are defined in Table 1. In Table 6, we present the results on performing Mann–Whitney *U* tests (categorical variables) and independent sample *t*-tests (numerical variables) for the developer subjects. Whereas

Table 7
Independent *t*-tests – Mann–Whitney *U* tests for researchers.

		NDRS		DRS			
		N	Mean	*N*	Mean	*t*	Sig. (2-tailed)
All Researcher	Time Researcher Objective	10	52.000	10	34.500	3.59	0.02**
	Correctness Researcher Objective	10	76.500	10	88.500	−2.06	0.06*
		N	Mean Rank	*N*	Mean Rank	*W*	Sig. (2-tailed)
	Effort Researcher Objective	10	12.500	10	8.500	85.0	0.11
	Satisfaction Researcher Objective	10	8.400	10	12.6000	84.0	0.09*
		N	Mean	*N*	Mean	*t*	Sig. (2-tailed)
Experience = LOW	Time Researcher Objective	6	48.333	6	35.000	2.90	0.03**
	Correctness Researcher Objective	6	66.670	6	89.170	−3.48	0.01**
		N	Mean Rank	*N*	Mean Rank	*W*	Sig. (2-tailed)
	Effort Researcher Objective	6	7.170	6	5.830	35.00	0.51
	Satisfaction Researcher Objective	6	5.580	6	7.420	33.50	0.36
		N	Mean	*N*	Mean	*t*	Sig. (2-tailed)
Experience = HIGH	Time Researcher Objective	4	57.500	4	33.750	2.28	0.07*
	Correctness Researcher Objective	4	91.250	4	87.500	0.70	0.51
		N	Mean Rank	*N*	Mean Rank	*W*	Sig. (2-tailed)
	Effort Researcher Objective	4	5.750	4	3.250	22.00	0.05**
	Satisfaction Researcher Objective	4	3.500	4	5.750	22.50	0.05**

in Table 7, we present the corresponding results concerning researcher subjects. In both tables we marked the statistically significant differences at 0.1 and 0.05 levels.

7. Discussion

In this section we discuss the findings of our work with respect to the research questions stated in Section 5.1. More specifically, Section 7.1 deals with the effect of using a pattern repository on the correctness of research and development objectives. In Section 7.2 we discuss the effect of using a pattern repository on objective completion time. Finally, Section 7.3 deals with the effect of using the pattern repository on objective effort and user satisfaction.

7.1. Effect of pattern repository on objective correctness

From Tables 6 and 7, we observe that when developers or researchers use Percerons, they tend to achieve higher objective correctness than those who are using other search engines.

More specifically, the results of our study suggested that developers that use the repository, produce code with fewer defects, in a statistically significant rate, than developers who are attempting to identify code without having prior knowledge of its structure. Additionally, the use of the repository appears to have a weaker effect on experienced developers who understand code structure easier. This is depicted in Table 6, where it is suggested that experienced developers reduce their error rates by 4.2%, whereas inexperienced developers who are using Percerons produced about 12.5% less errors when using the repository. This result is intuitively correct since inexperienced developers need more guidance than more experienced developers, who are familiar with identifying and adapting code written by other programmers.

On the other hand, researchers do not seem to be statistically significantly affected by the use of the repository w.r.t. objective correctness. Inexperienced researchers appear to produce better results when using Percerons and this enhancement is statistically significant. On the other hand, experienced researchers appeared marginal worse results when they used the repository, but this is not statistically significant.

7.2. Effect of pattern repository on objective completion time

Concerning objective completion time, both researchers and practitioners appear to complete the required objective quicker when they are using Percerons rather than mining forges. The gain in development objectives completion time is about 25% for both experienced and inexperienced researchers. On the other hand the gain for experienced developers is about 4%, whereas the gain for inexperienced developers is 32%. However, all differences in mean objective completion time, minor or major, are statistically significant, except from the difference concerning the experienced users. This may occur because an experienced developer has a better understanding on the source code structure. Thus, the benefit that he/she can gain from the use of the design pattern repository is not important.

7.3. Effect of pattern repository on objectives effort/ user satisfaction

Both developers and researchers that used Percerons suggested that they needed less effort to complete the objectives that they have been assigned, than those who were using other repositories. Similarly, Percerons users evaluated their effort at a lower rate than the NDRS users. However, the above results were not statistically significant, with the exception of the experienced researchers who proved/seemed statistically significantly satisfied and regarded that they needed statistically significant less effort with the use of Percerons.

8. Threats to validity

In this section of the paper we discuss the threats to validity of our paper. In any empirical study there are two kind of threat, namely threats to internal validity and threats to external validity. Threats to internal validity deal with problems that occur during the operation of the study and slightly alter the results. In this study we identified two internal threats to validity. Firstly, the use of a tool in order to identify design pattern instances from open source code may lead to some false-positive results. Additionally, some variables such as effort, correctness and satisfaction are not objective. Thus, the assessment of these features may introduce internal threats to the validity of our study. Performing the empirical method on different subjects may provide different results. Similarly, evaluating the correctness of objective by different evaluators might alter the results as well. However, we believe that the subject group consists of developers and researchers that can adequately assess their effort and satisfaction and that evaluators are experienced enough to objectively mark the objectives that they received.

Threats to external validity correspond to possible problems when someone attempts to generalize the results outside the scope of the study. In our study, the results cannot be generalized to all 23 GoF patterns, but only to the 11 that we have examined. Additionally, the results cannot be straightforwardly valid for reuse of closed source software, for games written in programming languages other than Java and for open-source domains, other than games.

9. Conclusions

In this paper we introduce and validate a design pattern repository, populated by pattern instance mined from open source games. Up to this point we have registered more than 4500 pattern instances with more than 20,000 pattern participating classes. It is believed that the repository can be helpful for both practitioners and academics, since it provide a mined dataset for code reuse and research. In order to evaluate the usefulness of the repository we conducted an experiment with researchers and practitioners with varying experience on design patterns.

The results of the experiment suggest that inexperienced developers are more likely to benefit from using the repository. More specifically, inexperienced developers needed statistically significantly less time in order to complete the required objectives and completed them with fewer errors than inexperienced developers who were trying to reuse code from search engines. The main reason for that is that a design pattern based repository holds information on the rationale of the reuse candidates. This fact provides information on how the classes communicate and on how client on this subsystem can be added. This extra information appears to be very important to inexperienced developers who might have problems in understanding the structure of code. At this point it is necessary to clarify that inexperienced developers are expected to use Percerons motivated by trying to find a code portion that implements a certain requirement. The designer although inexperienced and probably not aware of the benefits of patterns, will gain several benefits from using pattern-based components.

On the other hand, more experienced developers have not been majorly effected w.r.t. to objective correctness, because they can manually retrieve such information from source code. However, in cases when the aforementioned information is available the

time needed for completing the objectives, is statistically decreased. Concerning researchers, the most important findings concerned objective execution time. More specifically, both experienced and inexperienced researchers achieved better scores in completing the objectives quicker when using the pattern based repository. Additionally, the user satisfaction of experience researchers that used the repository has been found to be significantly higher from the user satisfaction of researchers that were not given a mined dataset in order to perform the objectives.

Appendix A

A.1. Design pattern roles

Design pattern	Roles
Strategy–State	Strategy/State
	Concrete Strategies/States (referenced as subclasses)
Adapter	Client
	Adaptee/Receiver
	Adapter/Concrete Command
Composite	Client
	Component
	Composite
	Leaf
Decorator	Component
	Decorator
	Concrete Component/Decorator
Factory Method	Product
	Creator
	Concrete Product/Creator
Observer	Observer
	Subject
	Concrete Subject/Observer
Prototype	Prototype
	Concrete Prototype
	Client
Proxy	Proxy
	Subject
	Real Subject
Singleton	Singleton
Template Method	Abstract Class
	Concrete Class
Visitor	Visitor
	Element
	Concrete Visitor/Element

Appendix B

B.1. Description and evaluation of objectives

O_1: Development Objective. Identify a component that deals with a game that implements the behaviour of weapons.

During this objective the subjects have been asked to identify a component that will handle weapons in a first person shooter game. In order to fasten the time needed to complete the task, we have omitted any activities related to graphics. The final outcome of the end system will be to print what weapon is in use. In order to test the fit of the component we provided the subjects with a main function that should be used.

```
Weapon w = new Shotgun();
w.reloadNow();
w.fire();
w = new Laser();
w.reloadNow();
w.fire();
```

The output of the execution of the scenario should be:

```
A shoutgun has been reloaded
You fire with a shotgun
A laser has been reloaded
You fire with a laser weapon
```

Evaluation guidelines
Retrieve some of the classes Weapon, Shotgun and Laser
Correct the identified classes in order to fit the main function
Add functionality or missing classes
Produce a code that compiles without errors
Provide the expected final output

O_2: Development Objective. Identify a component that deals with the attributes of hockey players. Hockey players are divided into two main categories (goalkeepers and field players). These categories share some attributes, but simultaneously each category has some different attributes.

During this objective the subjects have been asked to identify a component that will handle players' attributes for a hockey game. The final outcome of the end system will be to print the attributes of all players of a team. In order to test the fit of the component we provided the subjects with a main function that should be used.

```
Team t = new Team();
Player p = new Player(); PlayerAttributes pa = new
GoalkeaperAttributes ();
pa.createRandomPlayerAttributes();
p.setAttributes(pa);
p = new Player(); PlayerAttributes pa = new Field-
PlayerAttributes ();
pa. createRandomPlayerAttributes();
p.setAttributes(pa);
for (int i=0; i<t.playerSize();i++) {
System.out.println(''Player average attributes: ''
+ t.get(i).getTotalAttributesAverage());
}
```

The output of the execution of the scenario should be:

```
Player Average Attributes: 78.5 as goalkeeper
Player Average Attributes: 55.5 as field player
```

Evaluation guidelines

Retrieve some of the classes Player, GoalkeaperAttributes, FieldPlayerAttributes, Team and PlayerAttributes
Correct the identified classes in order to fit the main function
Add functionality or missing classes
Produce a code that compiles without errors
Provide the expected final output

O_3: Research Objective. Identify the average number of design pattern instances in stable board games.

Objective 1										
Game name	Number of Adapter instances	Number of Composite instances	Number of Singleton instances	Number of State/ Strategy instances	Number of Decorator instances	Number of Factory method instances	Number of Observer instances	Number of Prototype instances	Number of Proxy/ Proxy2 instances	Number of Template method instances
JAVARISK2	14	0	2	3	0	2	1	0	0	0
JSETTLERS	7	0	2	8	0	1	2	0	1	1
JOSE	31	0	17	25	1	2	3	26	0	23
JSOKO	4	0	3	8	1	0	0	1	0	2
JSUDOKU VER 0.3.1	5	0	3	18	0	2	1	1	0	0
AVG	12.20	0.00	5.40	12.40	0.40	1.40	1.40	5.60	0.20	5.20

The subjects have to fill a table like the one below:
Evaluation guidelines
Identify five stable board games
Retrieve and count the patterns instances for each board game
Calculate the average number of patterns instances
Document the results in the given table
O_4: Research Objective. Identify ten design pattern instances that have been added in the (k + 1) version of a game that did not exist in the (k) version of the game.

The subjects have to fill a table like the one below:
Evaluation guidelines

Game name	JSUDOKU			
Game version (n−1)	0.3.1			
Game version (n)	1.0.45		Objective 2	
Id	Pattern name	Class-1	Class-2	Method-1
1	ADAPTER	com.eriksilkensen.sudoku.SudokuControl	com.eriksilkensen.sudoku.SudokuView	public void saveGame(java.io.File) throws java.io.IOException;
2	ADAPTER	com.eriksilkensen.sudoku.BoardInventoryHelper	com.eriksilkensen.sudoku.InventoryBoardFactory	public com.eriksilkensen.sudoku.Board createBoard(com.eriksilkensen.sudoku.BoardDifficulties)
3	SINGLETON	com.eriksilkensen.sudoku.BoardDifficulties		private static javax.swing.ImageIcon faceAahIcon;
4	OBSERVER	com.eriksilkensen.util.Observer	com.eriksilkensen.net.Client	protected void setUpNetworking() throws com.eriksilkensen.net.NotConnectedException;
5	OBSERVER	com.eriksilkensen.sudoku.NetworkDialog	com.eriksilkensen.util.Observer	public void setArg(java.lang.Object);
6	OBSERVER	com.eriksilkensen.util.Observer	com.eriksilkensen.sudoku.SudokuModel	public void gameWon();
7	STATE–STRATEGY	com.eriksilkensen.sudoku.BoardInventoryHelper	com.eriksilkensen.sudoku.BoardFactory	public void supplyInventory(int);
8	STATE–STRATEGY	com.eriksilkensen.sudoku.Solver	com.eriksilkensen.sudoku.SolvingStrategy	public void changeStrategy(com.eriksilkensen.sudoku.SolvingStrategy);
9	STATE–STRATEGY	com.eriksilkensen.sudoku.SudokuModel	com.eriksilkensen.sudoku.BoardFactory	public void notifyObservers();
10	TEMPLATE	com.eriksilkensen.sudoku.GeneratingBoardFactory		public com.eriksilkensen.sudoku.Board leaveClues(int, com.eriksilkensen.sudoku.Board);

Identify games with multiple versions
Retrieve and document all pattern instances of version (k)
Retrieve and document all pattern instances of version (k + 1)
Identify differences in the two documents
Document the additional patterns in the given table

References

[1] R.T. Alexander, J.M. Bieman, R.B. France, A software engineering research repository, Special Interest Group on Software Engineering Lecture Notes (SIGSOFT'04 Lecture Notes), Association of Computing Machinery 29 (5) (2004) 1–4.

[2] A. Ampatzoglou, A. Chatzigeorgiou, Evaluation of object-oriented design patterns in game development, Information and Software Technology 49 (5) (2007) 445–454.

[3] A. Ampatzoglou, A. Kritikos, G. Kakarontzas, I. Stamelos, An empirical investigation on the reusability of design patterns and software packages, Journal of Systems and Software 84 (2011) 2265–2283.

[4] A. Ampatzoglou, I. Stamelos, Software engineering research for computer games: a systematic review, Information and Software Technology 52 (9) (2010) 888–901.

[5] A. Ampatzoglou, S. Charalampidou, K. Savva, I. Stamelos, An empirical study on design pattern employment in open-source software, in: Proceedings of the 5th International Conference on Evaluation of Novel Approaches in Software Engineering (ENASE '10), 22–24 July 2010, Athens, Greece, 2010, pp. 275–284.

[6] H. Apperly, Configuration Management and Component Libraries in Component-Based Software Engineering: Putting the Pieces Together, Addison-Wesley Longman Publishing, 2001. pp. 513–526.

[7] K. Arnout, B. Meyer, Pattern componentization: the factory example, Innovations in Systems and Software Engineering 2 (2) (2006) 65–79.
[8] L. Aversano, G. Canfora, L. Cerulo, C. D. Grosso and M. Di Penta, An empirical study on the evolution of design patterns, in: Foundations of Software Engineering (FSE' 07), 3–7 September 2007, Dubrovnik, Croatia, Association of Computing Machinery, 2007, pp. 385–394, .
[9] V.R. Basili, R.W. Selby, D.H. Hutchens, Experimentation in software engineering, IEEE Transaction on Software Engineering 12 (7) (1996) 733–743.
[10] P. A. Bernstein, U. Dayal, An overview of repository technology, in: Proceedings of the 20th Conference on Very Large Data Bases (VLDB'94), September 1994, Santiago, Chile, Association of Computing Machinery, 1994, 12–15, pp. 705–713.
[11] S. Bibi, I. Stamelos, L. Angelis, Bayesian belief networks as a software productivity estimation tool, in: 1st Balkan Conference in Informatics, Thessaloniki, Greece, November 2003.
[12] J. M. Bieman, G. Straw, H. Wang, P. W. Munger, R. T. Alexander, Design patterns and change proneness: an examination of five evolving systems, in: Proceedings of the 9th International Symposium on Software Metrics (METRICS'03), 03–05 September 2003, Sydney, Australia, IEEE Computer Society, 2003, pp. 40–49.
[13] S. Bjork, J. Holopainen, Patterns in Game Design, Game Development Series, Charles River Media, 2004.
[14] W. Brown, Large-Scale Component-Based Development, Prentice Hall, 2000.
[15] V. Burégio, E. Almeida, D. Ludrédio, S. Meira, A reuse repository system: from specification to deployment, in: Proceedings of the 10th International Conference on Software Reuse (ICSR'2008), 25–29 May 2008, Springer, Beijing, China, 2008, pp. 88–99.
[16] M. Conclin, J. Howison, K. Crowston, Collaboration using OSSmole: a repository of FLOSS data and analyses, in: International Workshop on Mining Software Repositories (MSR'05), St. Louis, Missouri, 17 May, 2005, Association of Computing Machinery, 2005, pp. 1–5.
[17] E. Damiani, M.G. Fugini, C. Bellettini, Corrigenda: a hierarchy-aware approach to faceted classification of object-oriented components, Transactions on Software Engineering and Methodology 8 (3) (1999) 425–472.
[18] Percerons Component Search Engine, Available from: <http://www.percerons.com>, 2011.
[19] M. Di Penta, L. Cerulo, Y. G. Gueheneuc, G. Antoniol, An empirical study of the relationships between design pattern roles and class change proneness, in: Proceedings of the IEEE International Conference on Software Maintenance (ICSM'08), 28 September – 04 October 2008, Beijing, China, IEEE, 2008, pp. 217–226.
[20] W.B. Frakes, T.P. Pole, An empirical study of representational methods for reusable software components, Transactions on Software Engineering 20 (8) (1994) 617–630.
[21] T. Fullerton, Play-centric games education, IEEE Computer 39 (6) (2006) 36–42.
[22] E. Gamma, R. Helms, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, Reading, MA, 1995.
[23] V. Garcia, D. Lucrédio, F. Durão, E. Santos, E. de Almeida, R. de Mattos Fortes, S. de Lemos Meira, From specification to experimentation: a software component search engine architecture, in: Proceedings of the 9th International Symposium on Component-Based Software Engineering (CBSE'06), June 29 – July 1 2006, Vasteras, Sweden, Springer, 2006, pp. 82–97.
[24] W. Harrison, A flexible method for maintaining software metrics data: a universal metrics repository, The Journal of Systems and Software 72 (2) (2004) 225–234.
[25] N.B. Harrison, P. Avgeriou, How do architecture patterns and tactics interact? A model and annotation, Journal of Systems and Software 83 (10) (2010) 1735–1758.
[26] B. Meyer, K. Arnout, Componentization: the visitor example, Computer, IEEE Computer Society 39 (7) (2006) 23–30.
[27] M. McShaffry, Game Coding Complete, Paraglyph Press, Arizona, USA, 2003.
[28] A. McWilliams, T. Reicher, G. Klinker, B. Bruegge, Design Patterns for Augmented Reality Systems, in: Proceedings of the 2004 International Workshop Exploring the Design and Engineering of Mixed Reality Systems (MIXER' 04), 13 January 2004, Funchal, Madeira, 2004, pp. 1–8.
[29] A. Meneely, L. Williams, E. F. Gehringer, Rose: a repository of education-friendly open-source projects, in: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '08), 30 June – 02 July 2008, Madrid, Spain, Association of Computing Machinery, 2008, pp. 7–11.
[30] J. Miller, M. Roper, M. Wood, A. Brooks, Towards a benchmark for the evaluation of software testing techniques, Information and Software Technology 37 (1) (1995) 5–13.
[31] R. Mili, A. Mili, R.T. Mittermeir, Storing and retrieving components: a refinement based system, Transactions on Software Engineering 23 (7) (1997) 445–460.
[32] A. Mili, R. Mili, R. Mittermeir, A survey of software reuse libraries, Annals of Software Engineering 5 (1998) 349–414.
[33] H. Mili, E. Ah-Ki, R. Godin-Hamid Mcheick, An experiment in software component retrieval, Information and Software Technology 45 (10) (2003) 633–649.
[34] D. Z. Nguyen, S. B. Wong, Design Patterns for Games, in: Special Interest Group on Computer Science Education (SIGCSE'02), 27 February – 2 March 2002, Cincinnati, Kentucky, Association of Computing Machinery, 2002, pp. 126–130.
[35] R. Prieto-Diaz, Implementing faceted classification for software reuse, Communications 34 (5) (1991) 89–97.
[36] R. Prieto-Diaz, P. Freeman, Classifying software for reusability, Software 4 (1) (1987) 6–16.
[37] T. M. Rhyne, P. Doenges, B. Hibbard, H. Pfister, N. Robins, The impact of Computer Games on scientific & information visualization: if you can't beat them, join them, in: 11th Visualization Conference, 8–13 October 2000, Salt Lake City, Utah, USA, IEEE Computer Society, 2000, pp. 519–521.
[38] A. Rollings, D. Morris, Game Architecture and Design, New Riders, Indianapolis, 2003.
[39] R. Rucker, Software Engineering and Computer Games, Addison Wesley, Essex, United Kingdom, 2003.
[40] J. Sametinger, Software Engineering with Reusable Components, Springer-Verlag, 1997.
[41] W. Scacchi, Free and open source development practices in the game community, Software Magazine 21 (1) (2004) 59–66.
[42] R. Seacord, S. Hissam, K. Wallnau, Agora: a search engine for software Components, Internet Computing 2 (6) (1998) 62–70.
[43] N. Shi, R. Olson, Reverse engineering of design patterns from java source code, in: 21st International Conference on Automated Software Engineering, September 2006, Tokyo, Japan, IEEE/ACM, 2006.
[44] S. K. Sowe, L. Angelis, I. Stamelos, Y. Manolopoulos, Using repository of repositories (RoRs) to study the growth of F/OSS projects: a meta-analysis research approach, in: Open Source Software Conference, 11–14 June 2007, Limerick, Ireland, Springer, 2007, pp. 11–14.
[45] N. Tsantalis, A. Chatzigeorgiou, G. Stefanidis, S.T. Halkidis, Design patterns detection using similarity scoring, IEEE Transactions on Software Engineering 32 (11) (2006) 896–909.
[46] L. Valente, A. Conci, Guff: A game development tool, in: XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI' 05), 9–12 October 2005, Natal, Brazil, 2005, pp. 1–10.
[47] P. Vitharana, F. Zahedi, H. Jain, Knowledge-based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis, Transaction on Software Engineering 29 (7) (2003) 649–664.
[48] M. Vokac, Defect frequency and design patterns: an empirical study of industrial code, IEEE Transactions on Software Engineering 30 (12) (2004) 904–917.
[49] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, A. Wesslen, Experimentation in Software Engineering, Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.
[50] M. Zyda, Educating the next generation of game developers, IEEE Computer 39 (6) (2006) 30–34.