Architecting an extensible framework to support Gamification of Software Engineering Activities

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS by Research
in
Computer Science.

by

SRIPADA VENKATA SAI KRISHNA
201250904
saikrishna.sripada@research.iiit.ac.in

Software Engineering Research Centre
International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2016

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Architecting an extesible framework to support Gamification of Software Engineering Activities" by SRIPADA VENKATA SAI KRISHNA, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. Y. Raghu Reddy

To my parents

for their love, encouragement and support.

Acknowledgments

A Single conversation with a wise man is better than 10 *years of study*

Chinese Proverb

My journey in IIIT-Hyderabad has been a wonderful experience. It could not have been so without the support of many people. As I submit my MS thesis, I would like to extend my gratitude to all those people who helped me in successfully completion this journey. First of all, I want to thank my advisor Prof.Y. Raghu Reddy, for accepting me as a student and constantly guiding me. His guidance has helped me improve not only as a researcher but also as a person. I thank my fellow research mates at Software Engineering Research Center for stimulating discussions. In particular, I am grateful to Dr. Devi Prasad for the intial intellectual discussions on pursuing research in software engineering area. I am also thankful to Prof. Shatrunjay Rawat for his constant moral support during my stay at IIIT.

I could not have accomplished it without the support and understanding of my parents. I wish to thank my parents Satyavathi and Malleswara Rao for being my constant support and motivation. Last, but not the least, thanks to IIIT community for giving me an inspiring environment and loads of opportunities to grow.

Abstract

Nothing can stop the onward march of an idea whose time has come.

Victor Hugo

Software engineering activities such as code reviews, change management, knowledge management, issue tracking etc. tend to be heavily process oriented and are repetitive in nature. Serious games, simulations, augmented reality, gamification are seen as potential options to improve human motivation towards performing process oriented repetitive tasks. However, gamification differs from other techniques as the original intent of the activity is not overly impacted. The idea of making a user lose or win is optional while gamifying an application. All these factors add on building a healthy competitive environment which is suitable for software product development industry as well as academic community. Prior research has shown gamification helped in building interesting e-learning systems which impact the academic community by improving the learning experience of students. Gamification of such activities is done by composing the core activities with game design elements like *badges, points* etc. Gamification can increase developers' interest in performing such activities. While there are various frameworks/applications that assist in gamification, extending and configuring the frameworks to add any/all the desired game design elements has not been adequately addressed. Implying configurable rules within a rule engine instead of implementing business logic to award game design elements is still unaddressed.

Addressing the aforementioned issues in this thesis, an extensible architectural framework for gamification of software engineering activities is designed and developed. The framework supports building new game design elements and gamifying software engineering processes. Game design elements are implemented as rest-full services. Gamifying an application involves adding features without modifying the original look and feel of the application. The framework consists of three major components : 1) A base framework consisting of software packages required to support an SE activity. 2) Game design elements which are implemented as micro services using python's Flask micro framework and 3) Basic front end components namely table, matrix and list which support the UI of game design elements.

The validation of this work is done in three ways. 1) A Gamified instance of code review activity is created using the framework developed. 2) To evaluate the ease of use and extensibility of the framework, developers are asked to extend the framework and instantiate gamified applications from the framework and our results are encouraging - developers are able to quickly develop new services as

well as use the existing framework to instantiate new gamified instances for software engineering activities. 3) The impact of tool developed is quantitatively analyzed by calculating the usefulness of code review comments. The number of comments given using the gamified code review tool are significantly similar to the number of comments given when code review is performed using existing tools. Thus indicating that the tool developed has the required elements of a code review tool.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Begin at the beginning, the King said gravely, "and go on till you come to the end: then stop."

Lewis Carroll, Alice in Wonderland

In software Engineering (SE) curricula, processes oriented activities like bug hunting, code reviews and inspections, requirement elicitation, change management, knowledge management, issue tracking, are part of under graduate computer science course curriculum [79]. These activities are taught in a sophomore level or graduate level software engineering course and take huge amount of time and efforts and tend to be repetitive in nature. In the academic setup these are normally taught using assignments, case studies etc. However such activities are best learned in a real time project setup than as assignment or case studies in a single course.

Maintenance activities like code reviews take time & effort and require students to repeat the same set of tasks over a period of time. This can lead to loss of interest in the activity affecting a student's level of interest in learning since (s)he may not be solving the assignment. [85], [16], [95]. Apart from academia, even in industry widespread apathy among developers is noticed due to repetitive tasks. This may lead to loss of interest in the activity affecting the productivity of the developer. In-turn this may hinder the progress of the project itself.

Gamification is seen as a technique to motivate developers to perform the same task repetitively by incentivising them in different ways. For example, allow a developer to use a new gamified tool to perform code review activity. Within each of these gamified tools, the developer must be incentivised with different game design elements like badges / levels / points / rewards. This will improve the interest of a developer in performing repetitive tasks. Literature on gamification shows that gamification is useful in improving user motivation in performing repetitive tasks. The use of gamification in Education [83], Enterprise health-care [52], marketing [97], etc. is increasing over the past few years [74]. A case study from Deloitte [23] revealed 37% increase in the users returning to Deloitte Leadership Academy training program every week as a result of gamifying their training activity. Another case study from Cognizant 20-20 insights shows results from two gamified applications namely *The Bugs Premier League* and *3Curve Gamification Contest* [28]. The Bugs premier league is a contest to reduce

maintenance backlog count from 500+ to less than 50. This helped in increased defect fixing rate by more than 75%. The *3Curve Gamification Contest* is a contest for release deliverables encourage team members to showcase their excellence in code quality, features completion and performance improvements. 3Curve Gamification Contest showed that cost to achieve code quality reduced by 75% while automating the code quality checks and code coverage. Pedreira et al. [74] found out that *"points"* and *"Badges"* to be the most commonly used game design elements.

Current gamification frameworks such as *BadgeCraft* [8], *Badgelist* [9], *BadgeOS* [10] or *OpenBadges* [69] concentrate on one game design element, i.e badges. These projects have a huge code base and to extending the framework to include new game design elements which requires understanding of developer manuals, contributing guidelines and approvals of quality gate keepers of the respective projects. These issues contribute to make the process of extending these frameworks to include a new game design element tedious. Other frameworks like *credly* [31] is specific to learning management systems, *keas* [57] is specific to health care, which are not relevant to software engineering domain. Frameworks like Badgeville [11], Bigdoor [17] require a commercial license. All these observations indicate that extensibility is still a common issue among the frameworks currently available. It is a tedious task to include a new game design element in these frameworks. Beta versions of frameworks like *zurmo* [103] promise extensibility as part of future releases. Considering all these issues, there is a need for generic and open-source framework which is extensible to include game design elements applicable to multiple domains.

Within the scope of this thesis, a generic architectural framework is proposed and the framework implementation has begun with game design elements which are identified to be suitable to software engineering domain. A service based approach is taken while building this framework. Service based approach makes it possible to extend the framework and build new game design elements by following our implementation guidelines. Game design elements suitable for software engineering domain are also implemented as part of this thesis work. Detailed notes on how to gamify applications for SE activities is presented in later chapters of this thesis. Evaluation of the framework is done in three ways. (1) From the perspective of the authors of this framework: where the architecture should be implemented to include proposed functionality. Code review activity is gamified using the framework to validate the same. (2) From the perspective of developer : implementation time and efforts either to extend the framework or to build gamified applications should noticeably come down to be able to justify the notion of extensibility. Developers are asked to extend the framework by building new game design elements and instantiate gamified applications for SE activities using our framework. Results are shown in a later chapter. (3) From the perspective of end-user: the gamified application should be successful in retaining the players interest and persuade the player to perform a task repetitively. A study conducted over $180+$ participants to use our gamified instance and we present the quantitative and qualitative outcomes. As a part of the work done for this thesis, code review activity is performed within the course. The results presented are from the activity being performed in two different years. During the first time when the code reviews are introduced, quantitative analysis is performed to calculate the

impact of code reviews. Sentiment analysis is performed to understand the student's behaviour and team dynamics while performing code review activity. On the next year when this course is taught, gamification is introduced and text analysis is done to calculate the usefulness of comments given to quantitatively analyse the impact of gamification and adoptability of the tool. During this iteration, students are not asked to fix the code review comments, rather the focus is on performing code reviews in a gamified environment. Hence calculating source code metrics in this phase is out of scope of this work.

Rest of this chapter is as follows. Overview of SE activities and how they are practiced in industry and academia is presented in section 1.1. Background and Literature on Gamification in section 1.2. Service based architecture and an overview of proposed framework is discussed in section 1.3. Contributions and organisation of thesis is presented in sections 1.4 and 1.5 respectively.

1.1 Software Engineering in Undergraduate curricula and Industry

In a paper published by Reddy et al. [79] on software engineering curriculum in India and else where, the paper specifies the emphasis on teaching programming during first year of undergradute education [3] [56]. This paper also brings in the view that current under graduate curricula looks at software engineering as a course rather than a discipline. Teaching the practices of software engineering involves introducing the software development life cycle, activities and processes to be followed while developing software. At IIIT-Hyderabad, software engineering process activities are taught as part of two software engineering courses namely Structured Systems Analysis and Design (SSAD) and Software Engineering [1]. These courses aim at enabling students to work in teams on real time projects from start-ups and deliver software systems. As a part of these courses peer code reviews are introduced to improve the code quality written by students. Impact of code review activity in a classroom setup is already addressed [53] [54] [61]. A case study was conducted on how these processes are taught at IIIT-Hyderabad and the results show that performing such repetitive tasks can lead developers to apathy [89] [88].

A study from industry [7] revealed that while finding defects remains the main motivation for peer code review activity, it provides additional benefits such as knowledge transfer, increased team awareness, and creation of alternative solutions to problems. Apart from these benefits, even in industry it is observed that developers loose interest due to repetitive tasks. This may lead to loss of interest in the activity affecting the productivity of the developer. In-turn this may hinder the progress of the project itself. Gamification is seen as a technique to motivate developers [22] to perform the same task again and again by incentivising them in different ways every-time they perform the task. For example, when a code review activity is performed by a reviewer, if s(he) is incentivised with different game design elements like badges / levels / points / rewards every-time s(he) does code review, the reviewer will be interested in performing the task repetitively.

[1] https://serc.iiit.ac.in/resources/courses

1.2 Gamification

Gamification [33] [34] is the process of gamifying an application with the use of game design elements and game thinking in non-gaming contexts to improve the users engagement, motivation, and performance. Gamification uses elements from game design, but is not a game. A game can be a first person / multi-player character based and is played in a visually compelling graphics environment, where as the environment in case of gamification remains the same of that of a tool being used to perform a task. Gamification can encourage and engage users, where users can compete with one another or simply challenge themselves to improve their own performance. Hence, gamification is seen as a promising solution rather than serious games [93], simulation games [37] or augmented reality systems [18]. Stackoverflow [91] is a classic example of gamification, where users are rewarded for answering questions with correct and relevant answers.

Game design elements are tools and techniques required to gamify an application [2]. Game design elements play an important role in determining the success of a gamified application. Gamification design process is the process of designing and applying game design elements to create the rules and mode of operation for a gamified application. To gamify SE activities, platforms supporting gamification of applications should support invocation and usage of any / all the game design elements required to gamify SE activities. After a comprehensive study, we collated a generic set of game design elements applicable for software engineering [90] which are detailed in a later chapter.

Application of gamification in various domains has increased in recent years. The emerging technologies hype cycle published by Gartner in 2013 [44] shows gamification has reached the top of the wave. However, the next publication of Gartner in 2014 [45] predicted 80% percent of the gamified applications would fail to meet their business objectives, primarily due to poor design. Literature review of papers on gamification frameworks [68] [74] discuss the available frameworks and their limitations such as extensibility issues, being domain specific etc., to adopt and build gamified applications.

1.2.1 Issues in architecting gamified environments.

On-going work in the field of gamification converge into the following three aspects (1) improving and building domain specific game design elements, (2) building platforms with a subset of game design elements to support gamification, and (3) identifying the challenges in gamifying an application. However, designing an architectural framework that can be used to realize and implement gamified instances of such activities with various game design elements is a challenge. In Software Engineering, process oriented activities can be considered as good candidates for gamification. For example, if a person is involved in code review activity, every time s(he) will come back to review the code, s(he) will be awarded with different incentives like badges/ points/ rewards/ levels etc. Existing gamified applications for code review activity provide the same incentive while performing an activity multiple times. Frameworks which are described in a later chapter in this thesis and in the literature [68] [74], does not

also provide the required support for realizing multiple instances of gamified applications with different incentives.

For any framework, the base framework used to build tools that implement the game design elements may vary. Current tools like *getbadges* [46] and *gitpoints* [50] that support gamification rely on existing tools like *github* [49] as their base framework. As a result, these tools limit the gamification to github supported SE activities. For example, Requirements elicitation activity may require all the features of collaborative text editing supported by utilities such as *"wiki pages"* [65]. Code Inspections and Code Reviews may require the underlying "diff utility" [36] support. Hence building an extensible architectural framework that supports required base utilities and provides an easy means of including additional set of base utilities, where in the game design elements can vary independent of the underlying framework is a challenge.

1.3 Overview of proposed approach for gamification

A systematic study published in Information and Software technology journal [74] provides detailed mapping of software engineering processes and gamified applications. Existing frameworks that support gamification tend to be application specific or game design element(s) specific. Open source frameworks like *Mozilla OpenBadges [69]* provide programmable interface. These programmable interfaces require developer to understand and tweak the code base of the frameworks. Information on available frameworks that support gamification, various game design elements supported by each of these frameworks and existing gamified applications to perform software engineering activities is collated in chapter 4. Scope, Commonality and Variability analysis[29] is performed to identify the game design elements suitable for software engineering activities.

Game design elements are implemented as rest-full web services. The front end for the game design elements is designed as ember.js components and back-end for the service is developed using the python's flask micro framework. Design and implementation details of the proposed architectural framework are detailed in later chapters. Incentives will be allotted to the players while playing a game based on pre-defined set of rules. However the rules and the incentives will vary from application to application. A rule engine is developed such that it takes in the rules as parameter values from with in a form and realize an instance of rule engine with the data provided by the developer. Ember.js helps build re-usable components and flask encourages re-usability by providing *blueprints* feature. By following the proposed design, a new game design element can be plugged in without effecting the framework. Similarly any out-dated game design element can be deleted once the data is taken care of.

Developers both from academic and industrial community assisted in evaluating ease of usage of the framework by instantiating gamified applications for SE activities. Another set of developers, again from both academic and industry participated in implementing new game design elements as services following the guidelines provided with in the documents, to extend the architectural framework. The biographical data of these developers along with the time taken by developer to 1) understand the frame-

work, 2) Install the prerequisite software packages to run the framework, 3) Time taken by authors to explain the framework to developers 4) to implement a game design element as a service and finally 5) the time taken to instantiate a gamified application from any existing non gamified application is captured for evaluation purpose.

1.4 Research Contributions and scope of this research

With in the scope of this thesis, impact of gamification is presented based on the literature. Existing gamified applications does not support incentivising players in multiple ways. Extending frameworks supporting gamification to include a new game design element is a tedious process and requires significant effort of the developer. In the scope this thesis, game design elements suitable for SE domain are identified. The architectural framework is currently developed to support only those game design elements amenable for SE domain. Although this framework supports to instantiate a gamified application for any web application, the work is evaluated by restricting to generate gamified applications for SE activities. Hence evaluating the framework to realize gamified applications for domains other than SE is beyond the scope of this thesis. Currently, only a part of the framework is exposed to open-source community, hence reporting results depicting adaptability of the framework with-in open source community is also out of scope of this project. A service based approach is followed to realize gamified applications, issues such as hosting, performance, scalability are out of the scope of this thesis. Instead extensibility aspect is considered while evaluating the framework. User-experience with respect to visually compelling environments while gamifying an application is also out of scope of this work.

The major contributions of this thesis are as follows :

(I) Identifying the game design elements suitable for SE activities.
Scope, commonality & variability (SCV) [29] approach is followed while coming up with a set of game design elements that are suitable to build a product line of gamified applications suitable for SE domain. Existing gamified applications and frameworks are studied and a list of game design elements used in these are identified. SCV analysis on these game design elements has resulted in game design elements suitable software engineering domain. These game design elements are then classified into *"static and "dynamic"* categories depending on the game play data being available at run-time. This is further discussed in chapter 4.

(II) Designing and building an architectural framework.
A service based approach for designing and implementing an extensible architectural framework to support the game design elements required to gamify software engineering activities. Game design elements are designed as microservices, implemented using python based microservice framework *flask* [2]. The minimal code changes required to build a new gamified instance are listed while presenting the

[2]http://flask.pocoo.org/

implementation of framework in Chapter 5.

(III) Evaluation of the proposed framework.
To evaluate the extensibility of the framework, developers are asked to extend the framework and instantiate gamified applications from the framework. Results show that developers are able to quickly develop new services as well as use the existing framework to instantiate new gamified instances for software engineering activities. The evaluation of the framework is performed with a controlled set of subjects consisting of mostly undergrad and graduate students from IIIT. Although a couple of subjects are from the industry, they are not experienced with the technologies used to implement our framework.

A Gamified instance of code review activity is created using the framework developed. Quantitative analysis on the impact of tool developed is also performed by calculating the usefulness of code review comments. To do this, an activity was performed involving $180+$ students from a sophomore level software engineering class. The number of comments given using the gamified code review tool are significantly similar to the number of comments given when code review is performed using existing tools. The results shows that the gamified instance developed is at par with the existing tools with respect to the desired features and is easily adoptable to perform peer code reviews.

1.5 Organisation of thesis

The rest of the thesis is organised as follows :

Chapter 2 presents the existing literature on code reviews and gamification. We also talk about the background required to design and implement the architectural framework for gamifying software engineering activities.

Chapter 3 elaborates on the way code reviews are taught and practiced in undergraduate curriculum. The results of the case study conducted are also presented.

Chapter 4 presents the outcome of scope, commonality & variability (SCV) analysis by listing down the game design elements suitable for software engineering activities.

Chapter 5 introduces the service based approach followed while designing and building the architectural framework. We also present an implementation detail of a new service and gamified application.

Chapter 6 details the gamified instance on code review process and display the screen shots of the tool developed. We present our experimental results of how developers other than authors used and extended the framework by building new game design elements and instantiating a new gamified instance by following our guidelines.

Chapter 7 discusses about the summary of thesis, conclusion and future prospects in this direction of research related gamification and software engineering.

Chapter 2

Related Work and Background

Your connections to all the things around you literally define who you are.

Aaron O'Connell

2.1 Related work

This section presents the literature on continuous code reviews, impact of code inspections and code reviews on code quality, code smells, Gamification, game design elements and existing gamified applications.

2.1.1 Code reviews

Software industry has seen the benefits of code review processes [81]. For example, a study within Microsoft [7] revealed that while finding defects remains the main motivation for review, it provided additional benefits such as knowledge transfer, increased team awareness, and creation of alternative solutions to problems.

Educators in computer science have seen benefits of peer review [53] [54] [61]. Empirical studies on integrating formal code reviews with the course curriculum have shown two broad benefits [6]: (1) Peer code reviews teach students about constructive criticism and consensus building, and (2) Peer reviews provide opportunities for deeper learning and critical thinking. However, previous studies were based on class room projects may or may not be applicable to live projects. Other researchers are mining software repositories to gain insights on various questions like developer thinking, team dynamics, etc. [72].

Peer code review process acts as an effective strategy to catch bugs in early stages of product development [7] [72]. The process can also help assess coding standards [61] and improve software quality [64]. These studies on modern code reviews support the notion that modern code reviews also assist in performing bug-fixing tasks [7] eventually leading to less changes in code. Studies in this area also

suggest that higher code churn leads to ripple effect changes [14]. A previous study on mining software repositories explored the impact of code review coverage and code review participation on software quality [64]. Outcome of this study indicated a clear advantage of code reviews. In spite of such visible benefits, it is often perceived as boring and busy / time taking activity. Gamification of code review process has been under study since sometime now [87] to reduce the boredomin performing repetitive tasks.

$ContinuousDelivery$ is a set of practices and principles to release software faster and more frequently [58]. This notion combined with code reviews gives rise to continuous code review work flow. Instead of reviewing code related to the feature under review, the reviewer reviews the whole source code of the project till date. A previous study [15] concludes that there is a clear positive effect on the understandability and collective ownership of the code base with continuous code reviews.

2.1.2 Gamification

Gamification is the use of Game Design elements, Game Mechanics and Game Thinking in non-gaming contexts to improve the user's engagement, motivation, and performance. Pedreira et al. [74] found out that *"points" & "Badges"* are the most commonly used game design elements. Earlier Studies on gamification of software engineering activities reveal that gamification helped developers to perform activity better [86], [16], [95].

2.1.2.1 Serious Games Vs Gamification

A serious game has a set of rules and actions bound together in a cohesive manner for the purpose of training or educating its intended users (players). Players can score and compete with one another based on the given set of rules. The games can be designed for individuals or for teams, where the latter encourages collaborations and facilitates community learning.

On the other hand, Gamification uses elements from game design, but is not a game. Gamification can encourage and engage users, where users can compete with one another or simply challenge one's own self to improve own performance. Gamification can exist without a game. Also in gamification, there may not always be an element of competition. It depends on the game design elements used where as a game implies a winner (and a loser), with players competing between each other.

Military, Health care, Business process management systems [101] were early adopters of serious games. Building a serious game requires creation of a story and scenes for the game. Although serious games have higher potential on the impact of engaging users, the possibility of losing a game may make the user stop using the application, which is counter to the major motivation of building a game. Also the cost involved in building a serious game is high.

Gamification of an application or specific activities costs much lower compared to serious games because it does not require creation of stories or scenes. Additionally original intent of the activity is

not impacted. The idea of making a user lose or win is optional while gamifying an application. As a result, the risk of losing users' interest is minimal.

Many researchers have provided evidence in favor of gamification, however a recent study of women engagement on StackOverflow [98] showed gamification can be counterproductive, encourage problematic behaviour or discourage minorities. It is always good to look at the community as experts versus novice resources and try to improve the expertise of novice users/ developers rather than basing the community on age, sex, color, etc. In this way, the goal of gamification should be to improve the overall expertise and pull the target audience to perform the repetitive task without loosing interest. We present one such approach where the skills of participants always start from an initial base value, ideally '0' or 'None' for all the game design elements made available in our architectural framework. This may bring in added advantage to already expert users, but this is also needed for the novice programmers / developers / designers / test engineers to observe the community and learn to do things. These young and inexperienced players will continue to learn and be motivated by obtaining the perks of a gamified application without loosing interest while comparing with the experts only based on the technical terms.

Currently, there are both commercial and open-source platforms that assist in gamifying an application. Platforms like *bigdoor*[17] and *badgeville* [11] provide enterprise support but are available at a certain cost. Additionally, similar to open source platforms they do not provide all the game design elements. To build a generic platform that supports gamifying an application, there are challenges identified as part of earlier work done in enterprise systems [5]. We take the learning from this work while proposing a generic platform suitable to build a platform for gamifying SE activities.

2.1.2.2 Game Design Elements anf Game Mechanics

Game design elements are tools and techniques required to gamify an application [2]. They play an important role in determining the success of a gamified application in any domain. Gamification design process is the process of designing and applying game design elements to create the rules and modes of operation for a gamified application. For supporting gamification of software engineering activities, any platform supporting the gamification of applications should support invocation and usage of any/all the game design elements required to gamify those activities. Pedreira et al. [74] found out that *"points"* and *"Badges"* are the most commonly used game design elements.

Game design elements are expected to act as stimulus in promoting interest among users. However, we identified one game design element i.e. (*social network connection between users*) that can assist in retaining users if its not supported within a gamified application. For example, let's consider the instance of Quora and StackOverflow. StackOverflow, intended to be a purely technical Q&A site achieved the best quality without allowing its users to be able to follow similar users or users whom a person is interested in. While Quora also being a Q&A site, but with different goal of getting any question answered with real life experiences could afford to let users follow people. Many times the most accurate answer on Quora is not on the top list of answers for a question, whereas Stackoverflow has the most accurate answer which is verified to be a working solution (the person who raised the

question will mark the working answer as accepted apart from upvotes by other users) presented on top always. This pattern can be related with answers that attain ”upvotes” based on social influence rather than real value add to the question. Additionally, the gamified application should remain fair to all the users without being manipulated or misused for the personal benefit of one particular user of the game.

Game Mechanics are the agents, objects, elements and their relationships in the game. They define the game as a rule-based system, specifying what there is, how everything behaves, and how the player can interact with the game world. with in the scope of this thesis, we limit the notion of game mechanics to being a distinct set of rules that dictate the outcome of interactions within the system. They have an input, a process and an output.

2.1.2.3 Existing gamified applications

Earlier studies suggest that gamification improves adherence to coding conventions of the code written by developers [75]. It is common to find gamified versions of code review, bug identification tools, etc. on the web these days. However the setup and usage of these tools are still a concern. All gamified applications are not likely to have the same game design elements. Hence there should be a customizable set of game design elements to choose from and develop a gamified application. To gamify applications in such a dynamic scenario, a service based approach is preferred. In a service based approach, a set of game design elements are made available as RESTful web services. Any developer while gamifying an application can use these services with minimal development effort required.

Table 2.1 represents some of the existing gamified applications in software engineering domain to show the short comings of existing applications. The list is kept crisp to represent gamified applications available for the software engineering activities rather than listing all the gamified applications. The list consists of applications like *gitpoints and getbadges*, which are either dependent on github for their base framework offering support only to those SE activities github supports. A detailed study on existing gamified applications, gamification frameworks, and game design elements available in the frameworks is performed and Scope Commonality & Variability analysis is performed to find out the game design elements suitable for software engineering domain. The results are presented in next chapter.

2.2 Background

In this section, the concepts required to understand the design and implementation of architectural framework and the tools used in building the framework and services are presented.

Source code metrics

Metrics to measure the change in software quality using peer review process can broadly be classified into four major categories [64]: Product Metrics [77], Process metrics [63] , Human factor metrics [19] and Coverage metrics [100]. For the purpose of this study, emphasis is on developer expertise (Human

Table 2.1 Classification of SE activity and gamified Applications

Activity	Gamified application
software requirements	collabreview [75]
configuration management	Devhub [35]
Implementation	code combat [26], Treehouse [94], code hunt [27]
Code Inspection	codebrag, gitpoints, getbadges
Documentation	collabreview

factor metrics), proportion of changes reviewed (Coverage metrics), and size and complexity of code (Product metrics).

Linting tools

Linting tool such as pylint or Jlint [1] are run over source code to ensure the coding standards and style guidelines are followed. Advanced linting tools support identifying bugs and code smells also. PEP-8 standards [2] as a guideline are followed during the case study presented in next chapter and used PyLint to measure the code quality. PyLint is a python source code analyzer looking for bugs and signs of poor quality based on PEP-8 standards.

Sentiment analysis tool : clips

Sentiment analysis on the code review comments to determine whether students are highly positive or deeply negative through out the activity and check on seriousness of students during the review activity is performed. Existing case studies on performing sentiment analysis on commit comments [51] suggest that teams distributed tend to have higher polarity in their emotional content. Since our course is project oriented and students are grouped into teams while performing software engineering activities, we considered performing sentiment analysis on process data to understand the learning quotient of students across the duration of the course. With the help of this service, the gamificatoin experts or the researches, can monitor the outcome by varying different GDEs, and can choose the best GDEs for current application.

Finding sentiment among textual data is a solved problem and tools like RapidMiner [3] are available to use. *Pattern* from Clips is used to calculate the subjectivity and sentiment among the code review comments. The approach we followed to extract and analyse the comments is shown in the Figure 2.1.

[1] http://jlint.sourceforge.net/

[2] https://www.python.org/dev/peps/pep-0008

[3] https://rapidminer.com/

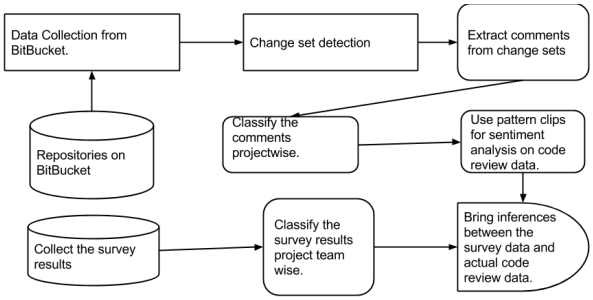

Figure 2.1 Approach followed for sentiment analysis of code review data.

Usefulness of text data : A machine learning approach

Usefulness of code review comments is measured by implementing a machine learning model. The model is implemented using python's sklearn[4] library. Various feature extraction and supervised learning algorithms were analyzed and tested before finalizing on the classification and prediction process shown in Figure 2.2. The method followed and results obtained are presented in evaluation chapter.6

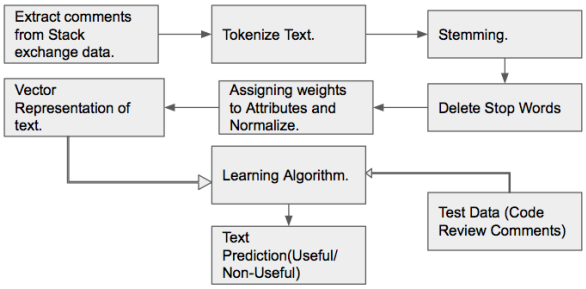

Figure 2.2 Text classification and prediction process

Service orientation and REST API

Service orientation facilitates the architecture to be implemented as a set of loosely coupled provider and consumer services. However, in a service oriented architecture the consumer will continuously poll requests to the provider of the services. This can result in unnecessary wastage of computational resources. Events and Hooks can overcome this problem by informing the consumer when to poll a request. Earlier studies [60] [102] [47] suggest this approach of communication between producer and consumer reduces resource utilization. We inherit features from this architectural style while designing the extensible architectural framework for gamifying SE activities.

flask - A micro framework

Flask [5] is a python micro framework and enables implementation of restful web services in python even by a novice python developer. As there are no rigid scaffolding rules and requirements, Flask

[4]http://scikit-learn.org/

[5]http://flask.pocoo.org/

provides more flexibility. Flask supports features like blueprints [6] to improve reuse of code base also. Flask is used to implement the required back-end for game design elements implemented as independent services.

Ember.js

Gamification of an application also deals with user interface to represent the game design elements developed. While building a framework to support extensibility and re-usability from the core, it is important to be able to re-use existing/newly implemented HTML/JS/CSS. However, all the tags required for a specific UI will always not be available. Game design elements can be represented using three major display components namely : list, table and matrix. But these tags / functions are neither available in HTML or javascript according the requirement. Every game design element provide data in key-value pairs or as an indexed list. There can be a custom third type of output like progress bar + current level. So front end is plug and play for first two categories, while third category will need custom UI (which can be developed either by developer or we can provide slowly) Ember components will let the developer create a HTML tag like functionality which once developed, will be available in browsers. Ember's implementation of components is very close to the Web Components specification[7]. Hence, the UI elements were implemented as different Ember components [8].

[6] http://flask.pocoo.org/docs/0.11/blueprints/

[7] https://www.w3.org/wiki/WebComponents/

[8] http://emberjs.com/api/classes/Ember.Component.html

Chapter 3

Code Reviews

Quality means doing it right when no one is looking.

Henry Ford

This chapter presents a case study done to understand how SE activities are taught in under graduate course curriculum and there by draw the motivation to try gamification approach while teaching software engineering.

3.1 A Case Study on Code reviews and code comprehension

3.1.1 Course Design

The study is done in an introductory software engineering sophomore level course Structured Systems Analysis and Design (SSAD) at IIIT Hyderabad[1]. Assignments are introduced to address the research questions listed and team-based projects from start-ups are allotted to student groups over a period of 16 weeks of the course. The projects were from start-ups based in the city of Hyderabad and hence involved real clients. These start-ups are from various domains including E-commerce, Health care, Data Analysis, etc. The projects were primarily web applications. As a part of the course, students are required to work on the project for a period of 12 weeks. Each student is expected to devote about 6-8 hours per week for the project. Since 2013, instead of offering a course project, students were offered projects from start-ups for simulating industry experience.

The course had 201 students with prior knowledge of Python who are enrolled during the year 2015 when the course is taught at IIIT-Hyderabad. The students were in Second year of Undergraduates majoring in computer science, and electronics and communication engineering. The students had prior exposure to C, Python and Web2py. There were 47 projects in various languages. The study was conducted on 33 out of 47 projects. The other 13 projects were not considered due to various reasons. For example, few projects required the students to work on HTML, some projects used proprietary

[1] http://www.iiit.ac.in

Table 3.1 Timeline indicating the research setup

PHASE 1	PHASE 2	PHASE 3	PHASE 4
Assignment 1 is released. The objective is to create a game in python using object oriented programming (OOP) principles.	Projects given by start-ups are introduced. Students are required to plan a minimum of three iterations using iterative development model.	Iteration II of project commences.	Iteration III (Final iteration) of the project.
Introduced PEP-8 standards to students. Analysed the initial coding standards of students.	Iteration 1 of the project commences.	Students are taught about code smells and code refactoring.	Delivery of project that meets the requirements.
Second assignment is released consisting of two parts. First part required writing Unit test cases for the first assignment.	Students performed peer-code review on code implemented during iterations	Teams identified most common code smells within a class and code smells between classes in their code base.	Maintainability index for python projects was calculated.
The latter part of the second assignment introduces measuring code quality using CK metrics.	Students identified the most common types of bugs among the code written by peers.	Survey on advantages of continuous code reviews was performed.	Performed textual analysis on code review data.

Platform as a Service (PaaS) platform like Progress Rollbase that didn't require the students to write a lot of code. The 33 projects considered in our study consist of 20 Python based projects and 13 Java based projects. The course had 10 graduate as well as senior undergraduate students act as Teaching assistants.

The course timeline is for 16 weeks out of which projects are for 12 weeks. This timeline remains common across all the teams and projects. Overview of the timeline is shown in Table 3.1. Detailed timeline is explained in the paper [88].

3.2 Results

This study focuses on finding the impact of code review activity on code comprehension, most common bugs and code smells within the code written by sophomore students and results showing sentiment

with in the code review comments while repeating these tasks multiple times. Code smells can be classified as *within classes* and *between classes* [41]. The term code smells "Between Classes" relates to smells that exist due to some relationship between these classes. The term "Within Classes" refers to smells within a class.

3.2.1 Static analysis and Conformance to standards

Pylint is a static analysis tool for python programming language. Pylint primarily looks for programming errors, helps enforce a coding standard and sniffs for particular code smells. The The most common errors identified by Pylint during the evaluation of the Pacman game assignment were: (1) Too few public methods, (2) Too many branches, (3) Method could be a function, and (4) Too many local variables. Static code analysis tools were also run on chosen projects after the final phase of our study. The ratings for python before and after each iteration's code review and refactoring were captured. The variation in the pylint rating is shown via a boxplot diagram in Figure 3.1 and Table 3.2. As seen in the table, the pylint ratings do not show a significant variation (mean changes from -2.529 to -2.47).

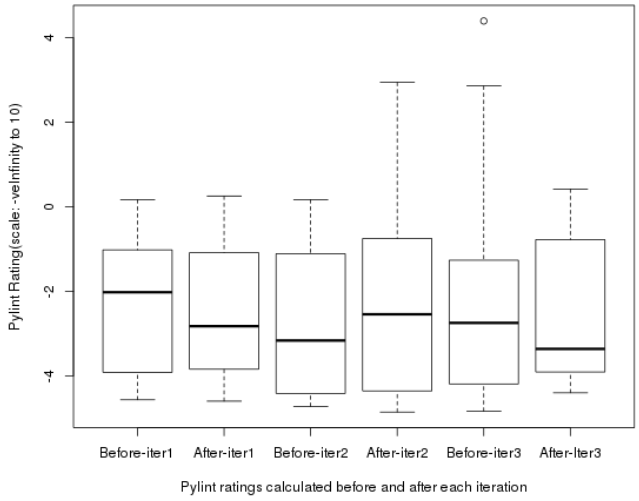

Figure 3.1 Variation in Pylint ratings calculated on python projects before and after code reviews of each iteration.

A survey is conducted to collect the qualitative details of students opinion on code review activity. Comparing results from survey after the first iteration of project and Pylint reports, it is observed that actual answers given by students and the results from static code analysis tools vary. Results shown in Table 3.2 reveal that after each iteration Pylint rating changed by small number. The median value varied from -2.82 to -3.35 given the increase in lines of code from initial iteration to the final iteration. This change is primarily attributed to the increase in code base while the project evolves. However, this

Table 3.2 Mean and Median values of Pylint ratngs. Ratings on a scale of -ve infinity to 10.

Iteration	Mean	Median
Before Iter1	-2.31	-2.07
After Iter1	-2.529	-2.82
Before Iter2	-2.67	-3.16
After Iter2	-2.49	-3.15
Before Iter3	-2.74	-3.359
After Iter3	-2.47	-3.35

negative impact is not intended and there should be a way thought of to improve the students motivation to learn and perform the activity better.

3.2.2 Code smells, Bugs and software code quality

The impact of code reviews to identify code smells was significant. The impact of continuous code reviews is clearly seen by the end of second iteration itself. Students were refactoring and fixing the code review comments given during earlier iterations without any explicit instructions from the mentors or the TAs. Additionally, they ended up writing small classes to avoid the *"long classes"* and *"long methods"*. Interestingly, it was noticed that some of the code smells like "Lazy class" and "alternate classes with different interfaces" began to rise [88].

As part of the survey taken at end of Iteration 2, one of the questions that was posed to the students was: *Did your understanding of the project improve after continuous code review and code refactoring process? Please measure yourself over the below scale:* The answers to the survey question were:

- 74 students said code reviews has significant impact in understanding the project.
- 9 students said there is no impact of code reviews.
- 23 students remained neutral.

It can clearly been seen that 74 of the 106 Students who responded to this question were positive, saying that continuous code review process had significantly improved their comprehension skills. However 30% of the students either stood neutral or felt the need for a better system to do code reviews by avoiding the boredom involved in repeating the task over a period of time.

3.2.3 Textual analysis of code review comments

In addition to the survey, to qualitatively find the impact of continuous code reviews, peer code review comments were analyzed. Code review comments from BitBucket's code review platform while students performed peer code reviews are analysed to find out the sentiment among the textual comments. Sentiment analysis of the peer code review comments with respect to sentiment and subjectivity was performed.

Comments are extracted using BitBucket REST API and analysed the comments for the above metrics using sentiment analysis and textual analysis tools. Source code to calculate sentiment is hosted on github and is available in [2]. *Pattern* from Clips is used to calculate the subjectivity and sentiment among the code review comments.

From the textual analysis of code review comments it is found that most of the comments were small and straightforward sentences with mean length of around 65.02 characters and only 18.15% of the comments contain over 100 characters. 55% of the comments had zero subjectivity score and 80% had subjectivity less than 0.5 as evident from Figure 3.3. This was similar to sentiment scores, where a peak was formed around 0 as shown in Figure3.2. 13.56% of the comments were negative, and 21.03% were positive, though only 2.3% had score over 0.5 suggesting high positive polarity. Most of the comments were instructive or informative, either providing instructions for further work or confirmation of the recent updates.

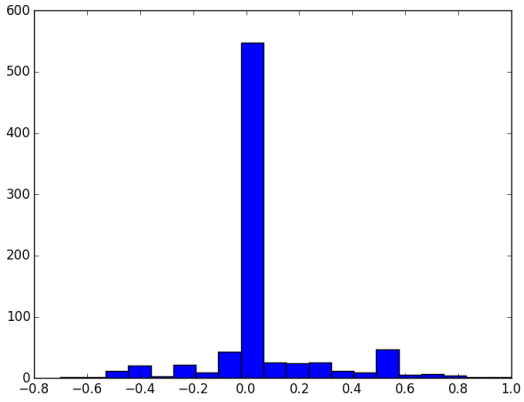

Figure 3.2 Sentiment analysis of code review data

Sentiment analysis on the code review data as shown in Figure 3.2 suggests that, even though most of the code review comments are useful and carry neutral sentiment as shown earlier, it did not necessarily imply lack of seriousness in the review activity. Most of the teams ended up with nearly 120 review comments per team over the three iterations. As opposed to the qualitative survey, sentiment analysis has shown that students neither are highly positive nor deeply negative through out the activity.

[2] www.github.com/anhsirksai/sentiment-SECorpus

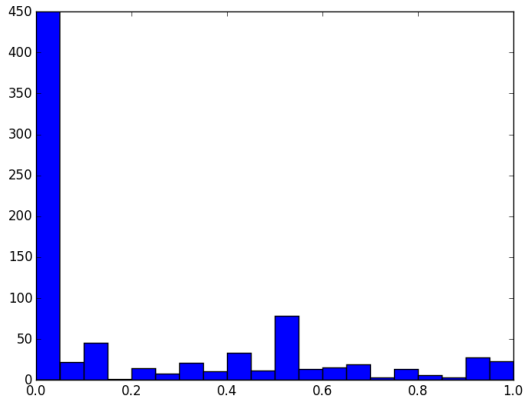

Figure 3.3 Subjectivity distribution analysis of code review comments.

3.3 Outcomes

These activities are performed following the iterative development model by allocating fixed time for code review and bug fixing during the course timeline. Initially students took time to get accustomed to this process oriented development but by the end of third project iteration, students gained better understanding of the project. However, the feedback from students and mentors of the projects, it is learnt that, the process of code reviews is time consuming and can get redundant if it's done frequently. Hence, there should be a way to reduce the boredom involved in performing such tasks and literature has shown us gamification of tasks such as code review process by introducing various game design elements like badges, levels, points, etc will help.

Chapter 4

Game Design Elements for Software Engineering domain.

Its not always about competing to win, but sharing how you won.

Daniel Debow

Scope, commanality and variability approach and its application to software engineering and gamification domains to identify the game design elements suitable for gamyfying SE activities are discussed in this chapter.

4.1 Scope, commanality & variability analysis

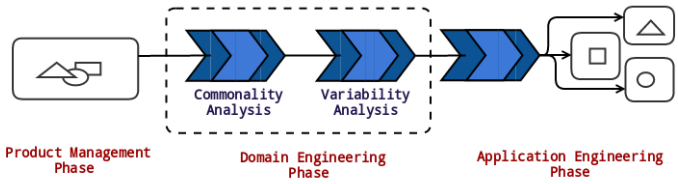

Figure 4.1 Product line approach

A major objective of this work is to facilitate service orientation to maximize the re-use of components while implementing the generic architectural framework. This approach is taken from Product line engineering. Using a product line approach [99], [73] helps building re-usable components while building a series of applications. The approach consists of three phases: (1) Product Management Phase (2) Domain Engineering Phase and (3) Application Engineering phase as shown in Figure 4.1.

4.1.0.1 Product Management

"scope" is to define a product family of gamified applications for software engineering in this case. The limits of our architectural framework are defined to include support of game design elements that

Table 4.1 Classification of Applications available and Domains

Gamified application	Domain
CodeBrag, collabreview, GitPoints, Getbadges	Software Engineering, collaborative learning
Treehouse, code hunt, code combat	Goal Tracking and Proof of Achievement
Mint.com [67]	Fostering Financial Independence and Goal Tracking
Recyclebank [78]	Customer Loyalty and Rewards
Step2 [92]	Customer Loyalty
Keas [57]	Employee Wellness, Cost Reduction
The World Bank [96] Evoke	Solving World Problems
DevHub	Project Completion
4food [1]	Social Loyalty, Brand Awareness, Engagement, and Social Missions
Proof [66]	Motivation, Goal Tracking
Engine Yard [40]	Customer Engagement
Beat the GMAT [12]	User Engagement, Goal Tracking and Competition
Bluewolf [20]	Employee Engagement and Motivation
ChoreWars [24]	Healthy Competition, Employee Motivation
SCVNGR [84]	Building Brand Awareness, Team Building
The Email Game [39]	Brand Awareness, Team Building, Productivity

are suitable for but not restricted to software engineering domain. The framework can later be extended to support game design elements required for other domains. However, it needs to be noted that if the domain is large, establishing across different applications within a domain becomes difficult.

4.1.0.2 Domain Engineering phase

James Coplien et al. [29] proposed an approach called Scope, Commonality and Variability (SCV) analysis to identify the common and variable aspects in a given scope, while working with a product line of systems. A list of gamified applications in software engineering domain as well as other domains (shown in tables previously) are analyzed as part of this work. Pedreira et al. [74] lists down the systematic mapping between software engineering and gamified applications. Table 4.2 lists down the game design elements identified among the existing applications and prior literature in the field of gamification.

SCV analysis is used to identify the re-usable components that are common across all realizations (Gamified applications) developed using the architectural framework. After listing down the common

and variable game design elements across domains, Game design elements are fine tuned to those that are specifically applicable to software engineering activities represented in the feature Matrix (Table 4.3). However, it needs to be noted that additional game design elements may be added as and when needed. These game design elements are further classified into static and dynamic categories based on the implementation of these as services. The services are served by the *provider* depending on the user input given by *consumer*. Static game design elements when called will return the same value irrespective of the state of the machine. Dynamic game design elements are the ones whose output will depend on the earlier execution state of the same or different function within the application/*consumer*. Static game design elements such as a unique UI common to the product family are implemented. A leader board and activity stream along with user profile are common game design elements and are implemented as part of generic architecture to improve the re-use of code base.

Table 4.2 Mapping of Game Design Elements and gamified application across various Domains

Domain	Game Design Elements	Applications
Branding and customer engagement	Avatars, Points, Leaderboards, Achievements, Badges, Levels, Missions	Samsung Nation, YoungOffice, Insider
Customer Loyalty, Education and Marketing	Social Network, Rewards, Avatars, Points, Leaderboards, Achievements, Badges, Levels	FourSquare, Teachcode
Employee wellness, Causes, Environment Recycling	Real prizes, Virtual Currency, Facebook credits, Virtual goods, Gifting, Sharing, countdown, Teams	Hallmark, Megagame
Others	Contests, Tactics, Recommendations, Activity Stream, Notifications, Followers, Play privately, Bonus, Medals, Game Analytics, Milestones, Accomplishments	Others

Both static and dynamic game design elements can be broadly classified into three categories namely: Behavioural, Feedback and Progression. *Behavioural* game design elements are usually focused on human behaviour and interaction with system. Depending on the player action and simulation *Feedback* design elements will be awarded to the user to improve the motivation of the user. *Progression* design elements define the state and structure of the gamified application.

4.1.0.3 Application Engineering

A realization of the architecture may or may not require all the game design elements. In other words, specific game design elements like points, badges, play-privately may be needed for realizing software engineering activities (for example, code review and bug reporting) while some other game design elements can be considered for future realizations. It is noticed that there is no single set of mandatory game design elements that are suitable for all the software engineering activities.

A generic set of game design elements applicable over various domains are collated in Table 4.2. By performing SC&V analysis on the available frameworks, game design elements and gamified applications, a list of game design elements suitable for software engineering domain are identified and are represented in Table 4.3. A comprehensive list of domains, gamified applications, frameworks supporting gamification and game design elements are presented in tables 4.1, 4.4 and 4.2. With in the scope of this thesis, Gamification of any software engineering activity will be a composition of a subset of services listed in table 4.3.

Table 4.3 Classification of Game design elements.

Classification			
	Behavioural	Feedback	Progression
Static	Achievements, Rewards	Checklists, Milestones	Points, Badges, Medals, Levels
Dynamic	Leaderboard, Teams	Activity stream	Missions, Countdown, Recommendations, Progress bar

Another specific game design element that can be designed as a service is ”play-privately”. To implement this as a service, three different views of users should be implemented namely, *Admin*, *private user* and *public user.* Hence the service is not atomic to itself and can be implemented as a composite of ”user profile”, ”rewards”, ”badges”, ”leaderboard” and other dependent services. A user is given a choice to make his profile public or private. In case of private, the user shall still be able to see the public profiles and compare his ranking with the public profiles. The ”public profile” user will not be able to see the details of a private user. However ”admin view” will have details of all the users and the data will be used while the global leader board display is implemented.

The other important design element (although not a game design element per-se) , ’FAQ’(Frequently asked questions) section should be included in any game. This should also include the explanation about the logic behind awarding game design elements to the user (player). This helps user understand how to progress through levels during the game and allows user to sail in the intended direction of gamifying an application.

Table 4.4 Classification of Frameworks available and game design elements

Available Frameworks	Game design elements supported.
Badgeville [11]	Competition, Feedback, Levels & Missions, Profiles, Ranks & Badges, Activity Feed, Ratings & Reviews, Social Login
Bigdoor [17]	Economy/Marketplace, Feedback, Profiles, Progress Bar, Ranks & Badges, Activity Feed, Community Forums, Likes & Comments, Ratings & Reviews, Share Plugin
Bunchball [21]	Competition, Economy/Marketplace, Feedback, Levels & Missions, Profiles, Progress Bar, Ranks & Badges, Teams, Time Pressure, Activity Feed, Community Forums, Share Plugin, Social Login
beintoo [13]	Competition, Economy/Marketplace, Activity Feed, Community Forums, Likes & Comments, Ratings & Reviews, Share Plugin, Social Login
UserInfuser [25]	Leaderboard, Widgets, Activity stream, Badges, Points, User profile
IActionable [55]	Competition, Progress Bar, Ranks & Badges, Time Pressure, Ratings & Reviews
CrowdTwist [32]	social loyalty, rewards
Manumatix [62]	Competition, Activity Feed, Social Login
Gigya [48]	Economy/Marketplace, Levels & Missions, Profiles, Ranks & Badges, Activity Feed,Likes & Comments, Live Chat, Ratings & Reviews, Share Plugin, Social Login
Punchtab [76]	Economy/Marketplace, Feedback, Levels & Missions, Profiles, Teams, Activity Feed, Community Forums, Share Plugin, Social Login
Keas [57]	Teams, Likes & Comments, Ratings & Reviews, Social Login
RepTivity [80]	Competition, Feedback, Levels & Missions, Activity Feed, Community Forums
RoarEngine [82]	Competition, Economy/Marketplace, Feedback, Levels & Missions, Profiles, Progress Bar, ctivity Feed, Community Forums,Ratings & Reviews, Social Login
Gaminside [43]	Feedback, Levels & Missions, Progress Bar, Ranks & Badges
Kudos Badges [59]	Competition, Economy/Marketplace, Ranks & Badges, Teams, Likes & Comments, Ratings & Reviews, Social Login
Mplifyr [70]	Competition, Feedback, Levels & Missions, Ranks & Badges, Teams, Activity Feed, Likes & Comments, Ratings & Reviews, Social Login

Chapter 5

Architectural Framework

Time itself is a solution to all problems.

P. V. Narashimha Rao

This chapter presents the proposed architectural framework design details in section 5.1, detail on how to build a new service in section 5.2, and a prototype implementation with the code changes required to gamify an application in section 5.2.3.

5.1 Design Detail

The architectural framework facilitates separation of static and interactive (dynamic) game design elements by implementing them as RESTful web services. The service based architecture shown in figure 5.1 consists of two layers. First layer consists of **Base Framework** along with **database utilities** and common services that are required to generate a gamified application which does not perform any SE activity. In order to generate a custom gamified application suitable for gamifying SE activity, the functionality provided by **services layer** should be incorporated. This will lead to realization of desired gamified application. The **Services** layer provides all the game design elements which are required to gamify software engineering activities, but all of these services may not be required for any single instance of gamification.

Static services are mostly common across realizations and hence they can be inherited from the framework. Unlike static services, dynamic game design elements are implemented as services which facilitate the option to tweak and extend the implementation by developer of gamified application.

Figure 5.1 shows our extensible architecture for gamifying software engineering activities. The extensibility is made possible by highly decoupled modules facilitating incorporation of other available modules as and when needed. The key parts of the architecture are : Services, Modules, Service Broker, Module Broker, Hook Executor, Rule Engine, Feedback Engine, Databases and Web Server.

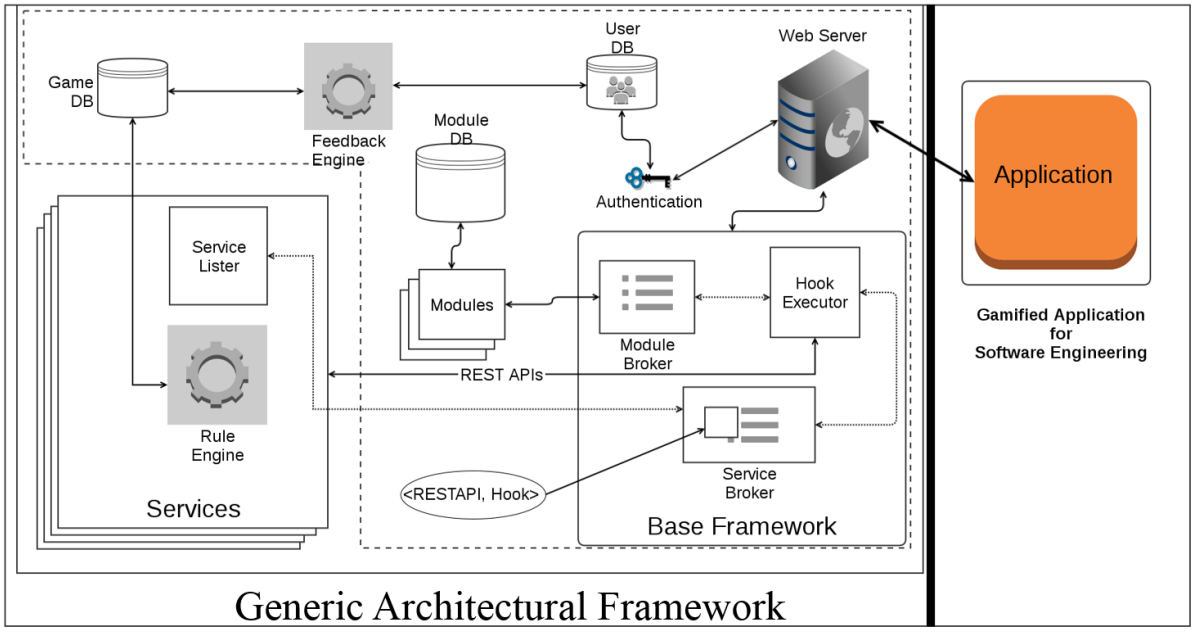

Figure 5.1 Architecture Diagram.

Authentication module: The User sending request to application should be authenticated before performing any interaction with application. Support for authentication module is provided using openid [1] or Mozilla's persona [2].

Base framework: It consists of all the common game design elements identified during the domain engineering phase 4.1.0.2 by performing commonality analysis over various domains shown in table 4.1. Along with the common game design elements, the architecture provides support for user interface elements such as charts, tables, forms using ember.js [3] a javascript based mvc framework. These elements can be dynamically consumed by modules and services, thus saving developer's effort.

Modules: Various utilities required to perform software engineering activities such as code review, documentation, bug tracking, etc exist as separate modules in our architecture. Modules can be disabled and enabled as and when required. Whenever a new module is introduced, the module will register its supported URLs, hooks and functionality with "Module Broker".

Module Broker: Module broker is an API for loading and interacting with modules.

Services : The game design elements are modeled as services. Services are responsible for providing the game design elements as REST APIs. Sample REST APIs provided by services are shown in Table 5.1. This table includes API for "badges" and "leader board" game design elements.

[1] http://www.openid.net/

[2] https://www.mozilla.org/en-US/persona/

[3] http:///www.emberjs.com

Table 5.1 Game design element (Service) end-points

HTTP Method	URI	Action
GET	http://[hostname]/api/badges/badge/[badge_id]	Retrieve image of a badge
GET	http://[hostname]/api/badges/user/[user_id]	Retrieve list of badges award to a user
POST	http://[hostname]/api/badges/event	Share event with badges service
GET	http://[hostname]/api/leaderboard/get	Retrieve stats
POST	http://[hostname]/api/leaderboard/event	Share event with leader board service

Service Broker: Service broker is responsible to communicate with services on behalf of base application. The Service Broker also contains $< RESTAPI, Hook >$ map from service listers, which is updated automatically as and when a new service is added to the generic framework. Except this, the base framework provides application developer the freedom to update $< RESTAPI, Hook >$ mapping according to the need of the application. Module broker is the point of contact from the application side for modules, while service broker from the services side.

Hook Executor: Each of the user actions is broadcast as an event. Hook executor holds the responsibility in our architecture to successfully execute all assigned tasks on each event. Along with this, it calls all services listening to that particular event by invoking lookup on $< RESTAPI, Hook >$ map. Asynchronous requests made to services are handled by Hook Executor.

Rule Engine: A rule engine is required to reward the user according to the rules and regulations defined for a game play. Depending on the user activity or event the rule engine alters the game design elements associated with a user's profile. The rules for each gamified application varies and are different among each realization of the architecture, as defined by the application developer. Within the scope of this thesis, rule engine is developed as a prototype and some code should still be written by the developer while generating a new gamified instance. While extending this work, creating a complete rule engine to handle configuring of rules from with-in a config file is being taken care.

Feedback Engine: Any gamified application, must consist of feedback mechanism. A suitable way of providing a Feedback Engine is through an event-driven approach. In our architecture, feedback engine will be a "server" that runs continuously in the background, collecting data from the rule engine and send information that will persuade users either via communication channels like mails or using *Activity stream* game design elements.

Databases: We configure three separate database schema to support gamified data, module data and user data. The data will be visible to user/admin depending on the user preference. Also, to implement

”play-privately” as a service, the game database scheme needs a minor modification, i.e. a new column ”public”, with value either 0 or 1. thus, creating a logical layer above normal Database schema.

A composition of services (game design elements and game qualities) when applied on applications performing software engineering activities can turn them into gamified activities. As an example instance, we have instantiated the generic architecture for gamifying the tools that support code review software engineering activity.

Hooks are special kind of functions, which when called, will broadcast events. As described above, all modules during initialization register hooks implemented in them with module broker, by passing a $< FunctionName, HookName >$ map. A collection of these maps is used to decide which functions are listening to a particular broadcasted event, and are executed accordingly. The idea of implementing hooks minimizes developer's time by providing relevant end-points quickly (in the form of events) to execute newly written function. This process can be best described as event driven Service-oriented architecture. [102] [47]

5.2 Implementation Detail

5.2.1 Game design element as a service

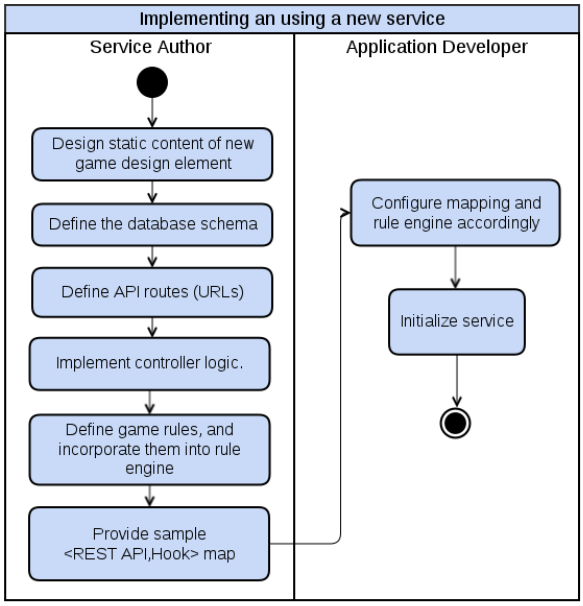

Figure 5.2 Activity Diagram - Implementing a new service.

The game design elements are designed and served as RESTful web-services. This in addition to the usage of MVC based framework enables extensibility in the architecture. All the services have URL namespace assigned to them. A URL Namespace is used for allocating unique URL prefix to every new

service, saving any possible routing conflict. This framework currently supports the most commonly used game design elements namely, *Badges*, *Leaderboard*, *User Profile*, *Activity Stream* and *Timeline* as services. New game design elements can be designed and implemented as shown in Figure 5.2. The implementation process can be repeated for any new game design element. For example, in case of Badges, the image of specific badge can be designed using a good image editing tool. Once the static content is made available, host the content on server from where nginx[4] can serve this content. For this service to be consumed by a gamified application and to assign the value to user profile, the metadata of the game design element should be maintained in a database table. This data will be copied to user table every time the game design element is awarded to a user profile. This requires design of database schema for the game design elements and storage of the metadata in game database. To serve the game design element as a RESTAPI, the route should be uniquely defined. The Controller Logic used to invoke the game design element shall be implemented within the service itself. Sample ¡REST API, Hook¿ mappings are provided to application developers. The developer should define constraints to call the Hooks, by tweaking these sample mappings. After these steps, the architecture would successfully support the newly implemented game design element.

The extensible architectural framework has been implemented by the authors. The framework is being extended as and when a new game design element is required for realizing a new gamified instance of an SE activity. Such an extensibility facilitates crowd sourcing the development of a new service by anyone apart from the authors of the framework. This also helps us to iteratively develop a new service (game design element) as and when required.

5.2.2 Implementation

The base framework is written using Flask (backend) and Ember.js (frontend). The realization of game design elements is straight forward, by creating (and using existing) APIs on Flask, and using visual modules available by architecture. The following explains a sample module implemented in base architecture:

1. Generalized comment module
   This module can be initiated on any webpage, on which Ember.js is enabled. It initializes automatically according to the default parameters which can be overwritten depending on page. All these parameters can be controlled based on unique-id of each intializations from seperate administrator panel also.

```
1  ...
2  previewAllowed:false,
3  expended:false,
```

[4] http://nginx.org/en/

```
markdown:true,
replyPlaceholder: '',
replyDisabled: false,
commentThreadLevel: 1,
hostname:'localhost',
fetchURL:'/comments',
postURL:'/comments',
publishURL: function(cid) {
  return '/comment/'+ id +'/publish';
}
unpublishURL: function(cid) {
  return '/comment/'+ id +'/unpublish';
}
archiveURL: function(cid) {
  return '/comment/'+ id +'/archive';
}
id: window.location.href,
cache_in_localstorage: 0,
...
```

Listing 5.1 Snippet displaying configurable parameters of comments

This module can be initiated on each issue description page, and place for discussion will automatically be created using it.

2. Generalized datatable
   This module can also be added on any page with Ember.js pre-enabled. Open-sourced pre built Ember.js component[5] are modified to suit the needs of this framework. This module can automatically adjust to any number of columns and rows, and type of content in these rows, automatically. It checks weather the content is plain text, or html, and accordingly gives filter and sort feature automatically.

```
...
isAjaxifiedSearch:false,
hostname:'localhost',
ajaxUrl:'/leaderboard/1',
headerAlias:null,
sortable:true,
searchable:true,
```

[5]https://github.com/abhijeetpandey/embercomponents

```
filterable:true,
sortProperties:['id'],
sortAscending:true,
pagination:true,
hidden:[],
search:'',
appliedFilters:[],
...
```

Listing 5.2 Snippet displaying configurable parameters of datatable

This module can be directly consumed to display leaderboard, available badges, issues listing, etc.

3. Generalized form builder The generalized form builder module can be given set number of fields and their types, and initiated on any namespace, for example '/issues'. It automatically created form for those fields at '/issues', and all the form entries can be seen on URLs like '/issues/n', where $n = \#issue$.

```
...
// Note: The page we are initializing this module, and localhost are not
    same hostnames.
namespace:'/issues',
APIhostname:'localhost',
ajaxUrl:'/issue/1',
header:'',
showRecent:0,
formFields:{
  'id', 'int'
  'title', 'text',
  'description', 'text',
  'screenshot', 'fileupload.img',
  'version', '1.0.1',
  ...
},
validationRegrex:{
  'title', '[^a-zA-Z0-9]',
  'description', '[^a-zA-Z0-9]'
  ...
},
```

```
21  hiddenFields: ['id', ...]
22  uniqueField:'id', // based on which url namespace would be used
23  ...
```

Listing 5.3 Snippet displaying configurable parameters of form builder

5.2.3 Prototype

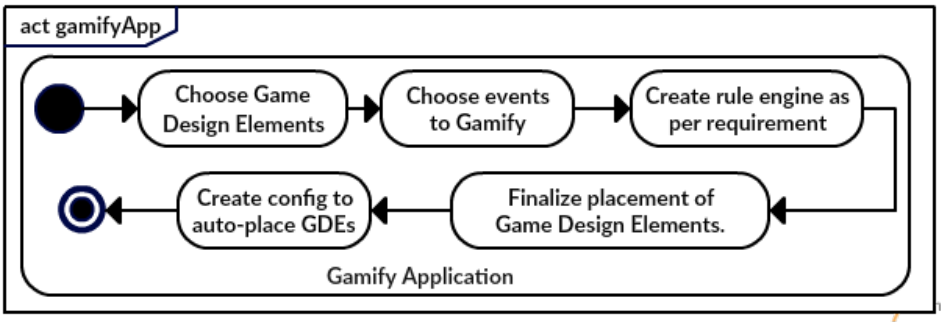

Figure 5.3 Implement a new gamified application

The prototype's code has been developed keeping in mind the extensibility, and ease-of-change for a new developer. Figure 5.3 Handlerbars[6] is being used by Ember.js for defining views. Here is a example which describes non-gamified implementation of *issue tracker*:

```
1  ...
2  Ember.TEMPLATES['issuePage'] = Ember.Handlebars.compile('' +
3  '{{issue #id}}' +
4  '{{#each comments}}' +
5  '{{comment this}}' +
6  '{{/each}}';
7
8  Ember.TEMPLATES['issue'] = Ember.Handlebars.compile('' +
9  '<div class= issue >' +
10 '  <div class= left-col >' +
11 '    <div class= title >{{title}}</div>' +
12 '    <div class= desc >{{description}}</div>' +
13 '  </div>' +
14 '  <div class= right-col >' +
15 '    {{user-card}}' +
```

[6] http://handlebarsjs.com/

```
16  '  </div>' +
17  '</div>';
18
19  Ember.TEMPLATES['comment'] = Ember.Handlebars.compile('' +
20  '<div class= comment >' +
21  '  <div class= left-col >' +
22  ' {{user-card-small}}' +
23  '  </div>' +
24  '  <div class= right-col >' +
25  '    <div class= content >{{content}}</div>' +
26  '  </div>' +
27  '</div>';
28
29  Ember.TEMPLATES['user-card'] = Ember.Handlebars.compile('' +
30  '<div class= user-card >' +
31  '  <div class= profile-pic >{{gravatar}}</div>' +
32  '  {{name}}' +
33  '</div>';
34  ...
```

Listing 5.4 Sample code snippet for illustrating prototype implementation

As we can see from the example all initially, template of 'issuePage' is defined, following with 'comment' and 'issue'. These templates can be hierarchically further divided into more granuar levels according to the need. Now to implement gamification over this issue tracker, following changes marked in red color are required to be made.

```
1   ...
2   Ember.TEMPLATES['issuePage'] = Ember.Handlebars.compile('' +
3   '{{issue #id}}' +
4   '{{#each comments}}' +
5   '{{comment this}}' +
6   '{{/each}}';
7
8   Ember.TEMPLATES['issue'] = Ember.Handlebars.compile('' +
9   '<div class= issue >' +
10  '  <div class= left-col >' +
11  '    <div class= title >{{title}}</div>' +
12  '    <div class= desc >{{description}}</div>' +
13  '  </div>' +
```

```
14 '  <div class= right-col >' +
15 '    {{user-card}}' +
16 '  </div>' +
17 '  {{vote-style-1 id="issue#@key'}}" +
18 '</div>';
19
20 Ember.TEMPLATES['comment'] = Ember.Handlebars.compile('' +
21 '<div class= comment >' +
22 '  <div class= left-col >' +
23 ' {{user-card-small}}' +
24 '  </div>' +
25 '  <div class= right-col >' +
26 '    <div class= content >{{content}}</div>' +
27 '  </div>' +
28 '</div>';
29
30 Ember.TEMPLATES['user-card'] = Ember.Handlebars.compile('' +
31 '<div class= user-card >' +
32 '  <div class= profile-pic >{{gravatar}}</div>' +
33 '  {{name}}' +
34 '  <div class= level-xp >{{level-style-1 @key}}</div>' +
35 '  <div class= badges >{{bages-large @key}}</div>' +
36 '</div>';
37 ...
```

Listing 5.5 Sample code snippet for illustrating prototype implementation

A comparison between gamified and non-gamified code, clearly indicates that only line number 17, 34 and 35 need to be added. This shows that, adding any new game design element in future would be easier to a new developer.

Chapter 6

Evaluation of Architectural Framework

Experts often possess more data than judgement.

Colin Powell

6.1 Evaluation

The architectural framework is evaluated from the perspective of the authors, where a gamified application is realised using the proposed architecture. From the perspective of developer, where a game design element implementation time and efforts required come down to be able to justify the notion of extensibility. From the perspective of end-user, where the gamified application is successful in retaining the players interest and persuading him / her to come back to perform the same task again and again.

6.1.1 Gamification of code review process - Author's perspective

Based on the architecture proposed, the modules required for a gamified code review application are instantiated. Game design elements like *Ranking system*, *Levels* and *Missions*, *Progress bar*, Counting the number of *likes* for activity and *favourite comments*, *Bonus*, *Badges*, *points* and *rewards* can all be used as incentive constructs.

Implementation of the architecture started with building a base framework consisting of authentication module, Module broker, Hook executor and Service Broker. Flask - a python based micro framework, enables implementation of restful web services. As there are no rigid scaffolding rules and requirements, Flask provides more flexibility. For the same reason Ember.js, a javascript mvc framework over the popular anjular.js is chosen to implement front-end web components. Nginx is chosen over apache as web server since, as Nginx works as a better asynchronous server, is event-driven and handles requests in a single (or possibly very few) threads. Although *Apache HTTP Server* is considered to be more suitable for dynamic websites, our requirement of handling multiple requests to serve various game design elements in parallel pushed us towards Nginx. The UI elements were developed

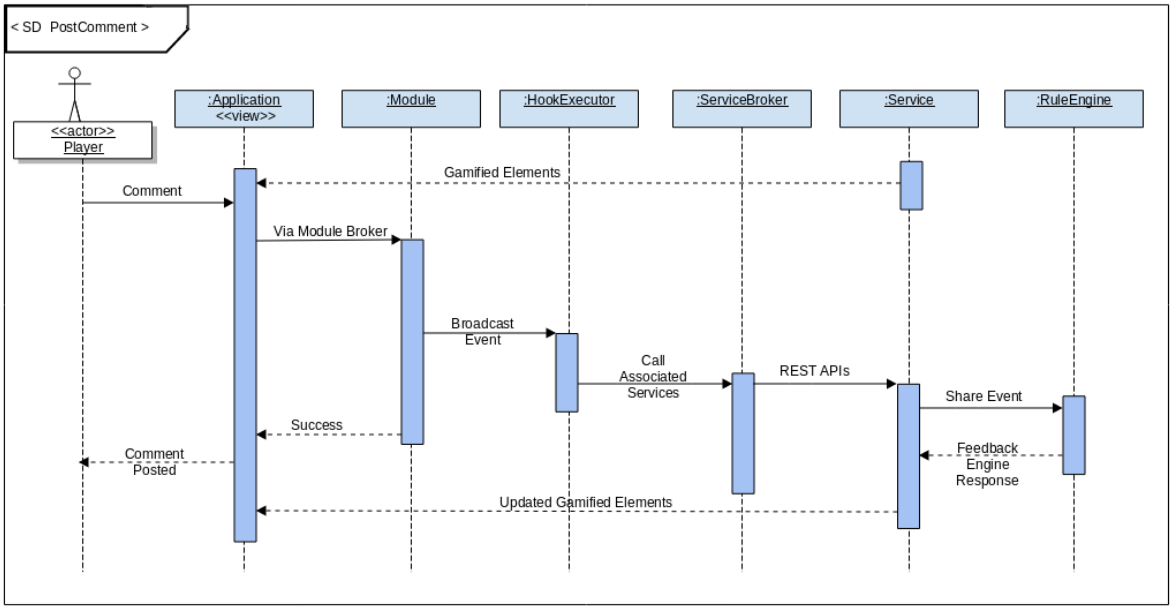

Figure 6.1 Sequence diagram showing the control flow to award a game design element to user profile.

as Ember components [1]. The base framework is extended by adding *module database*. *"diff utility"* module (extracted from open sourced diffcommenter[36]) and code review module were implemented. The code review module provides hooks namely `cr-comment-posted`, `cr-comment-edited`, `cr-comment-deleted`, etc. that can be used for gamification purposes.

For the code review instance, a set of static and dynamic game design elements are identified and are provided as Restful web services. The list of services: 1) in the list of static category - *points, checklists, rewards, badges, medals, levels, milestones and Up-votes*, 2) in the list of dynamic category *Leaderboard, Progress bar, Activity stream, Recommendations and Real-time Feedback*. The initial scope included only badges and leader board. However, while realizing badges and leader board as services, large chunks of duplicate code is identified within the implementation of these two services. Hence, "Factory Design Pattern" [42] is incorporated to reduce the duplication effort involved in implementing the back-end for game design elements by improving code re-usability.

Even though an earlier study [38] suggested to avoid factory design pattern the usage of factory pattern, this pattern benefited us by reducing the development efforts during early stages of implementation. It also provided benefits of implementing additional game design elements like *avatar* game design elements along with initial scope of 5 game design elements. The services interact with the service broker over REST APIs for registering $< RESTAPI, Hook >$ map when the service is enabled, and then Hook executor calls the APIs as and when required. The working of Hook executor, can be

[1] http://emberjs.com/api/classes/Ember.Component.html

seen in Figure 6.1 for one such activity user performs i.e. post new comment on the gamified software engineering tool.

The instantiation and implementation of such an architecture has its own set of challenges. In general, the challenges identified while building a gamification engine are divided into: Integration problems (such as: Front end integration, Rules and business logic Integration) and Feedback process [5]. In our architecture, a separate rule Engine and feedback Engine that addresses the challenges mentioned are designed. To validate the extensibility of our proposed architecture, multiple services and a run-time instance for code review process that made use of some of these services are implemented.

As described earlier in the Figure 5.2, implementation of a new service does not require implementing from scratch, instead existing code base can be reused following "factory pattern". The extensibility of this architecture leads to support of utilities (like "difftool") required for SE activities. While gamifying a different SE process, if the underlying utility support is missing, the new module 5.1 can be added within the generic architecture, the architectural framework is hosted on linux server. This approach of implementing game design elements gives the flexibility to use the services as plug-and-play modules while implementing a new gamified software engineering application.

A code review application is gamified using the generic architectural framework. The gamified application currently supports review of code, creation and management of user profiles, activity stream of all the registered users is displayed. Apart from these, a "leader board" is initialized from the generic framework and "badges" as a service is being utilised.

The realized architecture for code reviews consisted of the following components:

- Web server : Nginx
- Database : Mysql
- Frameworks : Django, Ember.js, Flask
- Modules : diff utility
- UI requirements : Tables
- Services : Badges, Leader Board, User Profile, Timeline, Activity stream.

Screen shots for the gamified instance that is developed can be found in the below Figures 6.2, 6.3, 6.4, 6.5 and 6.6

Evaluating extensibility of the framework, in terms of developing gamified versions of other SE activities using the framework is done with the help of developers other than authors of the framework. The results are shown in the next section.

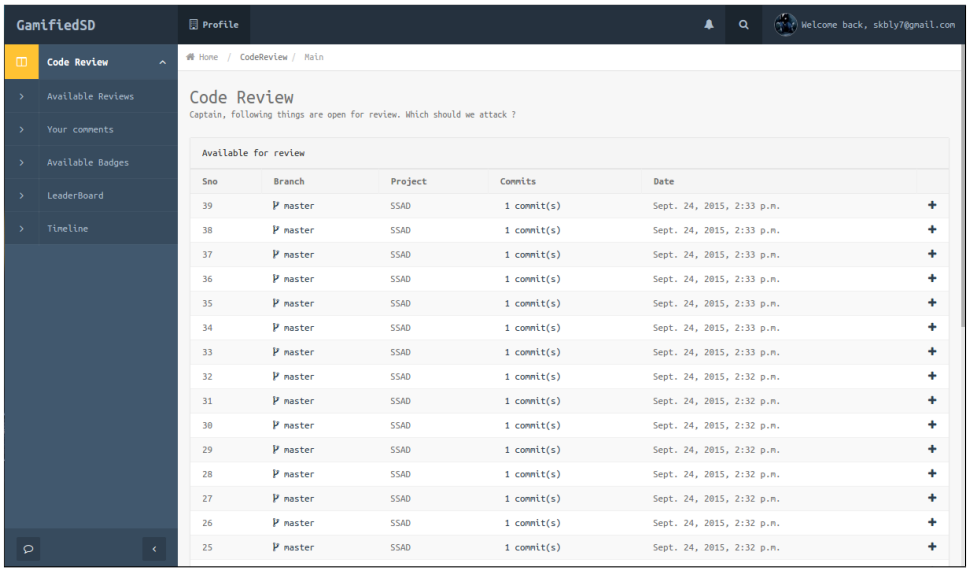

Figure 6.2 screenshot of code review tool showing commits.

Figure 6.3 screenshot of Badges that can be awarded.

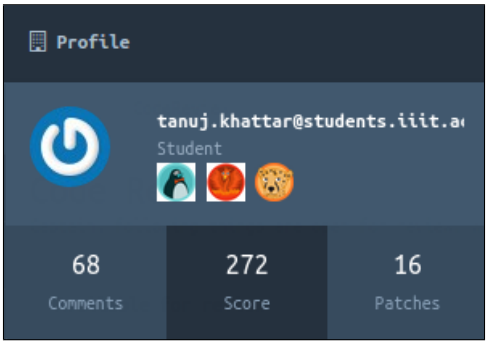

Figure 6.4 screenshot of Avatar.

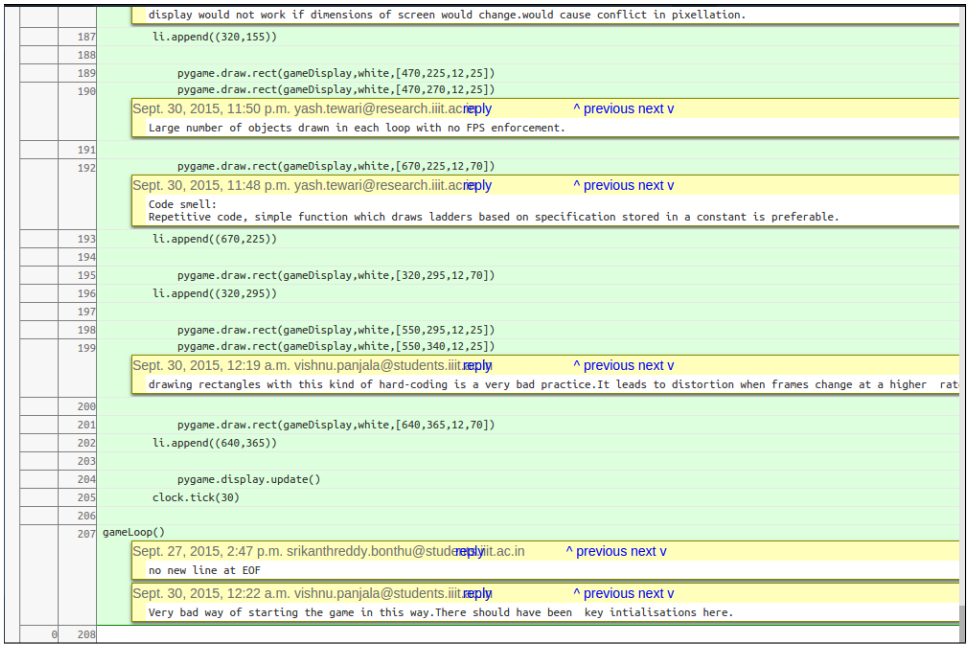

Figure 6.5 screenshot of diff tool with in the code review application.

Figure 6.6 screenshot of Leaderboard.

6.2 Extensibility experiment and results - Developer's perspective

A controlled experiment is conducted involving 10 developers out of which 2 are from industry and the rest, students of IIIT-Hyderabad. The experiment is designed such that each developer does not need to spend more than a couple of hours to either build a new service or to use the architectural framework to build a new gamefied instance.

Below Table 6.1 represents the biographical analysis performed in the developers of this survey.

developerID	age	experience in programming	experience in building micro services	experience with REST API	experience with ember	student / professional
1	24	good with python	has used flask	yes	no	Student
2	29	good with C# and java	yes	yes	no	professional
3	22	good with python	yes	yes	no	student
4	24	good with python	yes	yes	yes	professional
5	34	good with python	no	no	yes	professional
6	28	good with python	no	yes	yes	professional
7	22	good with python	no	no	no	professional
8	21	good with python	yes	yes	no	student
9	20	good with python	yes	yes	no	student
10	22	good with python	no	yes	no	student

Table 6.1 Biographic study of subjects involved in evaluation.

Table 6.2 shows the time taken by each developer to perform activities required to implement a new game design element. We have two modes of participation for people involved in evaluating our work i.e people directly sit with us and implement the service or people who are remotely available to evaluate. The repository [2] has instructions for developers while developing a new game design element. The repository on github [3] has separate branches created for game design elements developed by developers. Some developers preferred to create separate repositories on their own github handle by cloning the official repository and it is intimated during the experiment where, authenticity of the development activity is verified.

6.3 Usefulness of code review comments Activity - End user's perspective.

A study involving students was conducted where the gamified instance of code review tool developed is used along with other code review platforms. This study also provides a thorough validation on the adoption of the tool generated against existing tools. This study is performed to identify whether the

[2] https://github.com/IIITSERC/SE$_2$016$_G DE/blob/master/install.md$

[3] https://github.com/IIITSERC/SE_2016_GDE/

Task(implement GDE)	develop ID	time taken to install dependencies	time taken to explain framework by author	time taken to understand framework by developer	time taken to implement	successfully implemented (Y?N)	number of bugs found
implement a GDE	1	5	10	25	50	partial	2
implement a GDE	2	5	10	15	40	Yes	0
implement a GDE	3	5	20	20	75	partial	0
implement a GDE	4	10	5	8	15	Yes	0
implement a GDE	5	5	15	20	40	partial	0
implement a GDE	6	1	10	20	60	Yes	1
implement a GDE	7	5	25	30	60	partial	0
implement a GDE	8	1	10	10	45	Yes	0
implement a GDE	9	1	10	15	50	Yes	2
implement a GDE	10	1	10	10	50	Yes	0

Table 6.2 Part 1 - Results of the evaluation of framework by developers. (Time in min)

Task(instantiate a game)	develop ID	register (time)	initiate GDEs (time)	add gde in codebase (time)	add rules (time)	event addition in codebase (time)	satisfaction (1-5)
instantiate a game	4	1	1	10	2	5	5
instantiate a game	5	1	5	25	5	15	3
instantiate a game	6	1	4	20	3	10	4
instantiate a game	7	1	5	30	5	20	4
instantiate a game	8	1	2	15	3	10	5
instantiate a game	9	1	3	25	4	20	2
instantiate a game	10	1	2	20	5	15	3

Table 6.3 Part 2 - Results of the evaluation of framework by developers. (Time in min)

gamified application is successful in retaining the players interest and persuade him to perform the repetitive task. This activity is given as a mandatory assignment where in the students are divided into 5 groups each consisting of approximately 35 students and been asked to use one of the five code review tools: *Bitbucket* code review platform (referred to as Bitbucket from now on), *Phabricator*, *Github comment counter* (referred to as Github from now on), *CodeBrag* and *GamifiedSD*. The details of the game design elements available with the tools are shown in Table 6.4.

This quantitative study was performed involving 180+ undergraduate students enrolled in software engineering course at IIIT - Hyderabad. Students were taught about peer code review process, software quality by introducing the concepts of code smells, anti-patterns, refactorings, linting tools and code refactoring tools during the first three weeks of the course timeline. This activity helps in comparing the adaptability to the new gamified tool developed as well as in identifying the quantitative impact of using the gamified code review instance. Quantitative impact analysis is performed by calculating the usefulness of code review comments given by the students using these tools while performing peer code review activity, comparing the total number of comments given by students using different tools and average time spent on code reviews.

Table 6.4 Comparison of chosen code review tools

Tool	Gamified (Y/N)	Likes	Activity Stream	Points	Leader board	Badges	User Profile	Notifications
Bitbucket	No		Yes				Yes	Yes
Phabricator	No		Yes				Yes	Yes
Github	Yes		Yes				Yes	Yes
Codebrag	Yes	Yes						Yes
GamifiedSD	Yes	Yes	Yes	Yes	Yes	Yes	Yes	

6.3.1 Usefulness of comments

The usefulness of code review comments in gamified and non-gamified environments has been calculated and shown in Table 6.7. The total number of comments extracted from each tool, number of useful comments and not useful comments are shown. The percentage of useful comments with in each tool is shown in column 4 of the Table 6.7. The comments are classified as useful and not useful after pre-processing the extracted comments as discussed in Section 2.2.

The details of the code review comments' data sets used in the study are as follows:

1. **Code Review platform by Stack Exchange**: The data from platform is publicly available[4] at Stack Exchange. From the available data, we have selected answers and comments from questions having $python$, $algorithms$, $datastructures$, $beginner$, oop, $strings$ or $parsing$ as tags.

2. **Comments from all the deployed tools**: This data set consists of comments made by students during the assignment. Basic statistics is shown in Table 6.5.

3. **Review comments from Android project**: Publicly available code review comments' data set consisting of comments made by professionals on Android project [71].

Review comments from deployed tools and android project (200 comments each) are extracted and manually categorized them as useful or not-useful depending on the specific piece of code and context. Manual classification of the complete data set provided by stack exchange is difficult due to it's size (45,000+ comments). To facilitate classification, a feature vector of attributes having different weights and values scaled between 0 - 1 using MinMaxScaler[5] (to remove dominance of any particular attribute) are defined. The attributes selection and weights allotment was based on the insights from existing literature [4][30] and support provided by stack exchange in terms of classifying answers into "hot", "featured" and "active".The attributes with their weights (in brackets) are: 1. Number of upvotes for the answer (10). 2. Is the answer accepted (12) 3. Does question has any accepted answer (-3). 4. Number of upvotes to the question (8). 5. Reviewer reputation (1). 6. Average reputation of people involved in

[4]https://archive.org/download/stackexchange

[5]http://goo.gl/jn7sA3

Table 6.5 Count of code review comments

Tool	# Students involved in code review	Number of comments	Average
BitBucket	36	2935	81
Phabricator	34	1979	58
Github	37	1510	41
CodeBrag	37	2778	75
GamifiedSD	39	2950	75

question (6). 7. Average reputation of people involved in answer (5). 8. Total number of people involved in question (4). 9. Total number of people involved in answer (3).

The weightage of each of these attributes were given from 10 to 3 initially, but after tuning the weights to extract the most useful comments, the adjusted weights were from 12 to -3 although not all the 12 values are assigned. Adjustment of weight for the reviewer reputation needs a special mention among the changes done. Initially it is assumed that reviewer reputation played a vital role in selecting the best answer. This lead to some false positives in useful comments. The primary reason being that comments given by the same reviewer in his early learning stage are less useful.

Assigning useful and not useful label: Using the above list of attributes and weights, score for each comment is calculated and assigned a specific rank (score $\propto$ rank). Considering the list of attribute weights as x, where x_i means i^{th} attribute weight, and y as list of values for these attributes (for one particular answer) obtained as a result of MinMaxScaling to 0 - 1 scale. The formulae for calculating score is

$$\sum x_i \times y_i$$

Top 1000 answers are considered as useful and assigned a label "1" and the lowest 500 answers are considered as not useful and assigned a label "0".

Test Data: The test data for our experiments is the code review data collected from 180+ students who performed code review using the five tools. The average number of comments given while performing code review ranged from 41 to 81 as shown in Table 6.5.

Predictive Model Generation: Predictive modeling is a process to create a statistical model of future behavior. Predictive analytics in the area of Gamification is concerned with forecasting probabilities and trends on the Impact of gamification over the domain in which it is employed.The predictive model was based on three feature vector creation methods: $tf-idf$, $word2vec$, and $characternt-grams$ and four

classification methods: $LinearSVM$, $RandomForest$, $DecisionTree$, and $GaussianNaiveBayes$. All 12 possible combinations are considered while classifying the training data. During the experiment, training data consisted of 1500 comments from stack exchange and 100 comments from tools which were manually classified by us. All the manually labelled comments from tools (approximately 200+ comments) are tested using those 12 models. The precision, recall and support results of the possible combinations is shown in Table 6.6. The main criteria for choosing a predictive model was **Better overall accuracy** and **Better recall value**. As shown in the Table 6.6, Linear SVM along with word2vec approach gave the best results and are finally chosen while building the predictive model. In this table, a 0 in the first column indicates "not-useful" comment and a 1 indicates "useful" comment.

Table 6.6 Predictive model generation. '0' indicates not useful and '1' indicates useful comments.

	tf-idf				word2vec				character n-grams			
	precision	recall	f1-score	support	precision	recall	f1-score	support	precision	recall	f1-score	support
GausianNB												
0	0.54	0.93	0.69	73	0.54	0.93	0.69	73	-	-	-	73
1	0.93	0.55	0.69	127	0.93	0.55	0.69	127	-	-	-	127
total	0.79	0.69	0.69	200	0.79	0.69	0.69	200	-	-	-	200
DecisionTree												
0	0.7	0.85	0.77	73	0.74	0.88	0.8	73	0.68	0.82	0.75	73
1	0.9	0.8	0.85	127	0.92	0.82	0.87	127	0.88	0.78	0.83	127
total	0.83	0.81	0.82	200	0.85	0.84	0.84	200	0.81	0.8	0.8	200
RandomForest												
0	0.7	0.9	0.79	73	0.72	0.9	0.8	73	0.63	0.93	0.75	73
1	0.93	0.78	0.85	127	0.94	0.8	0.86	127	0.95	0.69	0.79	127
total	0.85	0.82	0.83	200	0.86	0.83	0.84	200	0.83	0.78	0.78	200
LinearSVM												
0	0.73	0.88	0.8	73	0.71	0.93	0.8	73	0.38	0.99	0.55	73
1	0.92	0.81	0.86	127	0.95	0.78	0.86	127	0.89	0.06	0.12	127
total	0.85	0.83	0.84	200	0.86	0.83	0.84	200	0.7	0.4	0.27	200

The usefulness of comments given by students seem to be low i.e less than or equal to 40% in column 5 of Table 6.7. Although instructions were given to the students about writing semantically relevant comments, it was noted that students reported style related comments too i.e code compliance with respect to PEP-8 standards. These were initially not considered to train the model. The number of useful style related comments are shown separately in column 6. Useful style related comments have a direct correlation with readability of code and hence contribute to ease of understanding.Therefore, the model is re-trained to classify PEP-8 related comments to be useful. The total number of style related comments given by students is shown in column 6 of Table 6.7. The new scores see an average of 10% - 15% increase in the number of useful comments given by students irrespective of gamified or non-gamified environment. An interesting inference can be made : if the same study was done on developers who are already well versed in code conformance, the results would have given a more accurate measure of the

impact of gamification.

Table 6.7 Classification Results.

		Style comments considered as "not useful"			Style comments considered as "useful"	
Tool	Total comments	# Useful	# Not Useful	% of Useful comments	Style comments count	% of Useful comments
GamifiedSD	2947	994	1953	33.72%	520	51.37%
BitBucket	3435	1179	2256	34.32%	534	49.86%
Phabricator	1976	743	1233	37.60%	339	54.75%
Code Brag	2782	849	1933	30.51%	513	48.95%
GitHub	1510	606	904	40.13%	283	58.87%

Survey Results

All students who used gamified environment/tools were asked to fill in a survey questionnaire after the code review assignment was completed. The survey included the following questions: (1) Which game design elements for Gamification did you like the most?, (2) Did gamification help in peer code review activity ?

In spite of the code review comments, it was surprising to note that nearly 50% of the students were in favour of gamification in code review process and only 19% students disliked it and felt it is not required. The results also provided us a prioritized list of game design elements like leaderboard, self score, likes, challengers, badges, rewards, etc. needed for gamification of code review process.

Analysis of this activity

A comparison on number of comments given by students using the gamified instance developed (GamifiedSD) versus existing tools show that the gamified instance is successful in achieving the adaptability. 6.5. The Quantitative impact of the comments obtained using various tools is shown in Table 6.7. Both quantitative analysis along with the survey results support the idea of gamification and the adaptability of the tool developed. This activity also shows that the gamified tool (GamifiedSD) developed is successful in retaining the developer's interest in performing code review activity.

6.4 Summary

In this chapter, gamified instance for code review activity is realised. The results on how the subjects involved either instantiated gamified applications or implemented a new service to extend the framework are presented. This chapter also presented a comparison of adoption of the tool developed along with

existing code review tools. Next chapter concludes this thesis by highlighting the contributions of this thesis and talk about the future research directions.

Chapter 7

Conclusion and Future work

I am still learning.

Michelangelo

at age 87

In this chapter, thesis summary followed by the scope of extending this work in future is discussed.

The focus of this thesis is to identify a process (Gamification) / tool (GamifiedSD) / framework (Architectural framework) to improve the user motivation in performing repetitive SE activities. Various approaches from the literature like serious games, simulation games are studied after which, gamification is identified as an ideal solution which requires minimum efforts from a developer while building a new gamified application. Further SC&V analysis is performed in identifying the game design elements suitable for SE domain which are essential components of any gamified application. Extensibility issues are observed within the existing frameworks apart from being domain specific. Using python's flask and Ember.js is a deliberate choice while building a framework instead of building a new domain specific language which forces the developer to learn a new language.

7.1 Contributions

1. A study on how software engineering activities are taught in CS curricula is performed and found the repetitive nature of SE tasks involved. Gamification is seen as a way to improve motivation in performing a task without getting bored.
2. A set of game design elements suitable to SE domain are identified to build a product line of gamified applications for SE activities.
3. An extensible architectural framework is proposed, designed and implemented. Game design elements suitable to SE domain are identified and are implemented as independent services.
4. Quantitative and qualitative analysis on the impact of gamifying SE activities is performed and results are presented.

7.2 Future work

1. This design supports extending of framework by building new services, but there is still scope to include automatic code generation i.e, when ever a new service needs to be instantiated, it can be done by taking a set of configurable parameters and a new service could be instantiated accordingly. However this is still a part of future work and can be done as an enhancement feature to our framework once the framework is rich enough with most of the game design elements supported by the framework it self.

2. There is scope to extend the framework to make it as generic as possible for it to be applicable to any domain. Perform SC&V analysis across multiple domains to implement a generalized list of game design elements. Hence, work can be done by simply identifying game design elements for other domains and extending the framework from there on.

3. Efforts to develop game design elements which will consider cognitive aspects also while designing a new game design element. Instead of using the most common game design elements, innovative approaches cab be adopted while rewarding a user.

4. A Single Javascript based complete gamification solution can be designed and developed instead of using Flask for building micro-services. A solution like google Analytics. The benefit of doing so - developer does not need to know multiple technologies while working on a single project. On the fflip side such a framework requires proficiency of the developer in Javascript.

Related Publications

1. **Full Papers**

   (a) Sai Krishna Sripada, Y. Raghu Reddy & Shivam Khandelwal (2016, Feb). **Architecting an extensible framework for Gamifying Software Engineering concepts.** In Proceedings of the *9th India Software Engineering Conference (ISEC '16).* ACM, New York, NY, USA, 119-130. DOI=http://dx.doi.org/10.1145/2856636.2856649

   (b) Sai Krishna Sripada & Y. Raghu Reddy (2015). **Code Comprehension Activities in Undergraduate Software Engineering Course - A Case Study.** In Proceedings of the *Software Engineering Conference (ASWEC), 2015 24th Australasian* Adelaide, SA, 2015, pp. 68-77. IEEE.

2. **Short Papers**

   (a) Sai Krishna Sripada, Y. Raghu Reddy & Ashish Sureka (2015, May). **In Support of Peer Code Review and Inspection in an Undergraduate Software Engineering Course.** In Proceedings of the *28th Conference on Software Engineering Education and Training* Florence, 2015, pp. 3-6. IEEE

Bibliography

[1] 4food. http://4food.com/. [Online; accessed 21-June-2016].

[2] E. Adams. *Fundamentals of Game Design*. New Riders Publishing, Thousand Oaks, CA, USA, 2nd edition, 2009.

[3] AICTE. All india council for technical education. http://www.aicte-india.org/downloads/mugcomputersc.pdf. [Online; accessed 21-June-2016].

[4] M. Allamanis and C. Sutton. Why, when, and what: Analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 53–56, 2013.

[5] M. Ameling, P. Herzig, and A. Schill. A generic platform for enterprise gamification. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 219–223, Aug 2012.

[6] K. Anewalt. Using peer review as a vehicle for communication skill development and active learning. *J. Comput. Sci. Coll.*, 21(2):148–155, Dec. 2005.

[7] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 712–721, 2013.

[8] badgecraft. https://www.badgecraft.eu/. [Online; accessed 21-June-2016].

[9] badgelist. http://www.badgelist.com/. [Online; accessed 21-June-2016].

[10] badgeos. https://www.badgeos.eu/. [Online; accessed 21-June-2016].

[11] badgeville. http://badgeville.com/. [Online; accessed 21-June-2016].

[12] beatthegmat. http://www.beatthegmat.com/. [Online; accessed 21-June-2016].

[13] beintoo. http://www.beintoo.com/. [Online; accessed 21-June-2016].

[14] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens. Modern code reviews in open-source projects: Which problems do they fix? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 202–211, 2014.

[15] M. Bernhart and T. Grechenig. On the understanding of programs with continuous code reviews. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 192–198, May 2013.

[16] B. Biegel, F. Beck, B. Lesch, and S. Diehl. Code tagging as a social game. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 411–415, Sept 2014.

[17] bigdoor. http://bigdoor.com. [Online; accessed 21-June-2016].

[18] M. Billinghurst. Augmented reality in education. *New Horizons for Learning*, 12, 2002.

[19] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: Examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 4–14, 2011.

[20] bluewolf. http://www.bluewolf.com/. [Online; accessed 21-June-2016].

[21] Bunchball. http://bunchball.com/. [Online; accessed 21-June-2016].

[22] B. Burke. *Gamify: How gamification motivates people to do extraordinary things*. Bibliomotion, Inc., 2014.

[23] H. business reveiw. https://goo.gl/vUgVMS, 2013. [Online; accessed 21-June-2016].

[24] chorewars. http://www.chorewars.com/. [Online; accessed 21-June-2016].

[25] cloudcaptive. http://www.cloudcaptive.com/. [Online; accessed 21-June-2016].

[26] codecombat. https://codecombat.com/. [Online; accessed 21-June-2016].

[27] codehunt. https://www.codehunt.com/. [Online; accessed 21-June-2016].

[28] cognizant 20-20 insights. https://goo.gl/562SQY, 2014. [Online; accessed 21-June-2016].

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

[30] D. Correa and A. Sureka. Fit or unfit: Analysis and prediction of 'closed questions' on stack overflow. In *Proceedings of the First ACM Conference on Online Social Networks*, COSN '13, pages 201–212, 2013.

[31] credly. http://www.credly.com. [Online; accessed 21-June-2016].

[32] crowdtwist. http://www.crowdtwist.com/. [Online; accessed 21-June-2016].

[33] S. Deterding, D. Dixon, R. Khaled, and L. Nacke. From game design elements to gamefulness: Defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, pages 9–15, New York, NY, USA, 2011. ACM.

[34] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon. Gamification. using game-design elements in non-gaming contexts. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 2425–2428, New York, NY, USA, 2011. ACM.

[35] devhub. http://www.devhub.com. [Online; accessed 21-June-2016].

[36] Diffutility. https://github.com/Babazka/diffcommenter. [Online; accessed 21-June-2016].

[37] D. S. Dorn. Simulation games: One more tool on the pedagogical shelf. *Teaching Sociology*, pages 1–18, 1989.

[38] B. Ellis, J. Stylos, and B. Myers. The factory pattern in api design: A usability evaluation. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 302–312, May 2007.

[39] emailga. http://emailga.me/. [Online; accessed 21-June-2016].

[40] engineyard. https://www.engineyard.com/. [Online; accessed 21-June-2016].

[41] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[42] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[43] gaminside. `http://www.gaminside.com/`. [Online; accessed 21-June-2016].

[44] Gartner. Gartners 2013 hype cycle for emerging technologies maps out evolving relationship between humans and machines. `http://www.gartner.com/newsroom/id/2575515`, 2013. [Online; accessed 21-June-2016].

[45] Gartner. Gartner says by 2014, 80 percent of current gamified applications will fail to meet business objectives primarily due to poor design. `http://www.gartner.com/newsroom/id/2251015(accessedApril2016)`, 2014. [Online; accessed 21-June-2016].

[46] getbadges. `https://getbadges.io/`. [Online; accessed 21-June-2016].

[47] S. Y. Ghalsasi. Critical success factors for event driven service oriented architecture. In *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, pages 1441–1446, New York, NY, USA, 2009. ACM.

[48] gigya. `http://www.gigya.com/`. [Online; accessed 21-June-2016].

[49] github. `http://www.github.com`. [Online; accessed 21-June-2016].

[50] gitpoints. `http://www.gitpoints.com`. [Online; accessed 21-June-2016].

[51] E. Guzman, D. Azócar, and Y. Li. Sentiment analysis of commit comments in github: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 352–355, New York, NY, USA, 2014. ACM.

[52] P. Herzig, M. Ameling, and A. Schill. *A Generic Platform for Enterprise Gamification*. WICSA-ECSA '12. IEEE Computer Society, 2012.

[53] C. Hundhausen, A. Agrawal, D. Fairbrother, and M. Trevisan. Integrating pedagogical code reviews into a cs 1 course: An empirical study. *SIGCSE Bull.*, 41(1):291–295, Mar. 2009.

[54] C. D. Hundhausen, A. Agrawal, and P. Agarwal. Talking about code: Integrating pedagogical code reviews into early computing courses. *Trans. Comput. Educ.*, 13(3):14:1–14:28, Aug. 2013.

[55] iactionable. `http://iactionable.com/`. [Online; accessed 21-June-2016].

[56] IEE. Computer science curricula 2013:, Dec. 2013.

[57] keas. `http://www.keas.com/`. [Online; accessed 21-June-2016].

[58] S. Krusche and L. Alperowitz. Introduction of continuous delivery in multi-customer project courses. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 335–343, May 2014.

[59] kudosbadges. `http://www.kudosbadges.com/`. [Online; accessed 21-June-2016].

[60] Z. Laliwala and S. Chaudhary. Event-driven service-oriented architecture. In *Service Systems and Service Management, 2008 International Conference on*, pages 1–6, June 2008.

[61] X. Li. Using peer review to assess coding standards - a case study. *Frontiers in Education Conference, 36th Annual*, pages 9–14, 2006.

[62] manumatix. `http://manumatix.com/`. [Online; accessed 21-June-2016].

[63] T. J. McCabe. A complexity measure. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[64] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 192–201, 2014.

[65] mediawiki. `http://www.mediawiki.com`. [Online; accessed 21-June-2016].

[66] mindbloom. `http://www.mindbloom.com/proof`. [Online; accessed 21-June-2016].

[67] mint. `https://www.mint.com/`. [Online; accessed 21-June-2016].

[68] A. Mora, D. Riera, C. Gonzalez, and J. Arnedo-Moreno. A literature review of gamification design frameworks. In *Games and Virtual Worlds for Serious Applications (VS-Games), 2015 7th International Conference on*, pages 1–8, Sept 2015.

[69] Mozilla. `http://openbadges.org/`. [Online; accessed 21-June-2016].

[70] Mplifyr. `http://www.mplifyr.com/`. [Online; accessed 21-June-2016].

[71] M. Mukadam, C. Bird, and P. C. Rigby. Gerrit software code review data from android. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 45–48, 2013.

[72] A. Murgia, P. Tourani, B. Adams, and M. Ortu. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 262–271, 2014.

[73] L. Northrop. Introduction to software product lines. In J. Bosch and J. Lee, editors, *Software Product Lines: Going Beyond*, volume 6287 of *Lecture Notes in Computer Science*, pages 521–522. Springer Berlin Heidelberg, 2010.

[74] O. Pedreira, F. Garcia, N. Brisaboa, and M. Piattini. Gamification in software engineering, a systematic mapping. *Information and Software Technology*, 57:157–168, January 2015.

[75] C. R. Prause and M. Jarke. Gamification for enforcing coding conventions. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 649–660, New York, NY, USA, 2015. ACM.

[76] punchtab. `http://www.punchtab.com/`. [Online; accessed 21-June-2016].

[77] S. Purao and V. Vaishnavi. Product metrics for object-oriented systems. *ACM Comput. Surv.*, 35(2):191–221, June 2003.

[78] recyclebank. `https://www.recyclebank.com/`. [Online; accessed 21-June-2016].

[79] Y. R. Reddy and K. V. Nori. Teaching software product engineering in undergraduate computing curriculum. In *2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE T)*, pages 175–178, April 2014.

[80] reptivity. http://www.reptivity.com/. [Online; accessed 21-June-2016].

[81] P. C. Rigby and C. Bird. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 202–212, 2013.

[82] roarengine. http://roarengine.com/. [Online; accessed 21-June-2016].

[83] D. Rojas, B. Kapralos, and A. Dubrowski. Gamification for internet based learning in health professions education. In *2014 IEEE 14th International Conference on Advanced Learning Technologies*, pages 281–282, July 2014.

[84] scvngr. http://www.scvngr.com/. [Online; accessed 21-June-2016].

[85] L. Singer and K. Schneider. Influencing the adoption of software engineering methods using social software. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1325–1328, June 2012.

[86] L. Singer and K. Schneider. It was a bit of a race: Gamification of version control. In *Games and Software Engineering (GAS), 2012 2nd International Workshop on*, pages 5–8, June 2012.

[87] W. Snipes, A. R. Nair, and E. Murphy-Hill. Experiences gamifying developer adoption of practices and tools. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 105–114, 2014.

[88] S. Sripada and Y. Reddy. Code comprehension activities in undergraduate software engineering course - a case study. In *24rd Australian Software Engineering Conference (ASWEC), 2015*, May 2015.

[89] S. Sripada, Y. Reddy, and A. Sureka. In support of peer code review and inspection in an undergraduate software engineering course. In *Software Engineering Education and Training (CSEET), 2015 IEEE 28th Conference on*, pages 3–6, May 2015.

[90] S. K. Sripada, Y. R. Reddy, and S. Khandelwal. Architecting an extensible framework for gamifying software engineering concepts. In *Proceedings of the 9th India Software Engineering Conference*, ISEC '16, pages 119–130, 2016.

[91] stackoverflow. http://www.stackoverflow.com. [Online; accessed 21-June-2016].

[92] step2. http://www.step2.com/loyalty/. [Online; accessed 21-June-2016].

[93] T. Susi, M. Johannesson, and P. Backlund. Serious games: An overview. *Technical Report HS-IKI-TR-07-001*, 2007.

[94] teamtreehouse. https://teamtreehouse.com/. [Online; accessed 21-June-2016].

[95] N. Tillmann, J. De Halleux, T. Xie, S. Gulwani, and J. Bishop. Teaching and learning programming and software engineering via interactive gaming. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 1117–1126, May 2013.

[96] urgentevoke. http://www.urgentevoke.com/. [Online; accessed 21-June-2016].

[97] A. Uskov and B. Sekar. Serious games, gamification and game engines to support framework activities in engineering: Case studies, analysis, classifications and outcomes. In *Electro/Information Technology (EIT), 2014 IEEE International Conference on*, pages 618–623, June 2014.

[98] B. Vasilescu, A. Capiluppi, and A. Serebrenik. Gender, representation and online participation: A quantitative study. *Interacting with Computers*, 26(5):488–511, 2014.

[99] D. M. Weiss and C. T. R. Lai. *Software Product-line Engineering: A Family-based Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[100] M. W. Whalen, A. Rajan, M. P. Heimdahl, and S. P. Miller. Coverage metrics for requirements-based testing. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, ISSTA '06, pages 25–36, New York, NY, USA, 2006. ACM.

[101] I. Whitepaper. Serious solutions with serious games straining with complex systems. `https://www-935.ibm.com/services/multimedia/serious-games-overview.pdf`, 2011. [Online; accessed 21-June-2016].

[102] Q. Zagarese, G. Canfora, E. Zimeo, I. Alshabani, L. Pellegrino, and F. Baude. Efficient data-intensive event-driven interaction in soa. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1907–1912, New York, NY, USA, 2013. ACM.

[103] zurmo. `http://www.zurmo.com`. [Online; accessed 21-June-2016].