The final version of this preprint appeared in the proceedings of
the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)

Code Tagging as a Social Game

Benjamin Biegel*, Fabian Beck†, Benedikt Lesch* and Stephan Diehl*
*University of Trier, Germany
Email: {biegel, diehl}@uni-trier.de
†VISUS, University of Stuttgart, Germany
Email: fabian.beck@visus.uni-stuttgart.de

***Abstract*—Keywords or tags summarize documents on an abstract level and can also be used for describing code fragments. They might be leveraged for retrieving features of a software system, understanding program functionality, or providing additional context. While automatic approaches at best are only able to retrieve information that is already contained in the source code, manual tagging could add valuable extra information from qualified expertise of the developers. However, tagging code is tedious. To make code tagging more fun, we introduce a social gamification approach: developers independently tag code fragments and are rewarded if their solutions conform to the solution of other developers. We implemented the game as a Facebook plug-in. A pilot user study suggests that the game mechanics are motivating and promote the proposition of reasonable tags.**

I. Introduction

Code tags, for instance, meaningful keywords assigned to each method, are valuable: tags can give a quick overview of the code, tags might abstract and relate code to real-world concepts, tags could help to come up with better results for code search and feature location, tags may help reveal hidden semantic relations of the code, or tags (if given by others) could identify potential problems with respect to program comprehension. There exist automatic solutions for code tagging [1], [2], [3], [4]: the vocabulary used in the code for identifiers and within comments is analyzed to generate a list of keywords. However, these solutions can only detect what is already in the code and cannot abstract and evaluate the current implementation. For manual code tagging, several tools already provide some support [5], [6], but do not particularly motivate the developers in tagging the code or do not address crowdsourcing. For other applications, such as labeling images, games that allow collaborative tagging were already successfully applied [7], [8].

In this paper, we transfer the idea of tagging games to source code and suggest a social gamification approach to crowdsource code tagging: Players are shown small code fragments of software projects—here, methods of Java systems—and are asked to enter descriptive tags for each fragment. Based on previous answers of other players, they are rewarded if they guessed the most popular previous tags. Through high-score lists, in particular one showing the scores of their friends, players are motivated to tag more code fragments and use meaningful tags. We implemented a prototype *Facebook* plug-in and tested it in a small user study. The results of this initial evaluation are already partly reflected in the presented game design.

II. Related Work

Manually tagging code fragments can simply be done by using comments in the source code. A more convenient way, however, might be provided through tools that support and standardize the tagging process: Storey et al. [5], [6] present a tool called *TagSEA* that uses code tagging for collaboratively defining waypoints and improving code navigation. When tags for code fragments have been derived, they could also help in many other applications, as De Lucia et al. [4] discuss: They can be used to analyze change impact, to detect code clones, to locate software features, to find semantic couplings of artifacts, to measure software quality, or to retrieve traceability links. Further, code tags can be used for building taxonomies of software terms [9]. Not just code fragments, but also other software artifacts might profit from being tagged by the developers, for instance, work items [10], [11].

There exist automatic code summarization techniques that retrieve a list of keywords for a code fragment from its source code based on term frequency metrics [1], Latent Semantic Indexing (LSI) [2], or Latent Dirichlet Allocation (LDA) [3] De Lucia et al. [4] compared different automatic methods to manual code tagging and found that simply deriving keywords from class names, method signatures, and attribute names comes closer to manual tagging than more complex methods such as LSI or LDA. Xia et al. [12] suggest a technique for recommending tags for items on Q&A pages like *stackoverflow.com*. Moreover, there are methods that create natural language text summaries of source code [13], [14], but that goes beyond the scope of this paper because the task of writing descriptions would require too much effort for the users in a gamification approach like ours.

Deterding et al. [15] define *gamification* as *"the use of game design elements in non-game contexts"*. Hamari et al. [16] give a general overview of gamification and summarize empirical evidence on its impact: most of the early studies conducted so far show positive effects of gamification. Gamification has already been used in some areas of software engineering, for instance, by introducing individual and team achievements into the software development process to reward task completion [17], by counting and sharing numbers of commits to encourage frequent, cohesive commits [18], or by analyzing work patterns and rewarding desirable behavior [19]. Education scenarios have attracted particular attention where gamification is used to foster learning and the application of certain best practices [20], [21]. Coding environments for children and beginners can be even designed as *serious games* [22].

Besides these gamified software engineering scenarios, we are, however, not aware of any gamification approach that specifically targets code tagging. But beyond applications in software engineering, tagging games have been explored for entities such as images [7], [8], places [23], or movie soundtracks [24]. Ahn and Dabbish [7] introduce a game design where pairs of players are rewarded if they suggest the same tags for an image; a similar design is chosen by Goh et al. [8]. In our approach, we also reward guessing the same tags but do not assume that pairs of users play at the same time. Instead, we use asynchronous interaction and particularly focus on the friends of a player.

III. Code Tagging Game

The key idea behind our approach is to apply social game mechanics to code tagging. Keywords or catchphrases respectively are used to characterize a particular source code fragment (e.g., a method). In order to encourage developers to produce qualitative code tags, we make use of two core strategies: First, in the form of points, developers get rewarded for contributing code tags. Second, by connecting the game to a social network, tagging becomes a collaborative game. Since the points of all players are permanently visible on a high-score list, developers could be motivated to compete with each other and thus might be drawn to playing the game more regularly. Most frequently posted common tags for a specific code fragment yield the highest scores. Since tags are added only by the players and consent is rewarded, this approach ensures certain tag quality.

This section describes the game concept that we developed based on a prototype implementation. This implementation was realized as a *Facebook* plug-in; the *Facebook API* offered all the functionality we required for the social networking aspects of the game. Please note that—to reflect the insights gained from an early user evaluation reported in Section IV—the concept discussed in the following goes beyond our prototype implementation. Although the illustrating figures shown are close to the evaluated user interface, they already reflect the later extensions. Also, the visual appearance of the user interface has been re-designed and optimized, but has not yet been actually implemented in the plug-in. In our examples and implementation, we focused on tagging methods of Java systems, but the concept can be easily transfered to other programming languages and pieces of code. A limiting factor, however, would be that the amount of source code presented is graspable in a short period of time. To express the generality of the approach, we use the term *code fragment* in the following to identify those pieces of source code.

A. Basic Game Mechanics

The basic game mechanics are inspired by the well-known game show format *Family Feud*, in which points are rewarded for naming the most popular responses given by 100 people asked before the show. Similar to this concept, for each code fragment, the code tagging game holds a top ten list of previously most posted tags. A player gets points for posting a tag that is already in this list. The higher the tags in the top ten list, the more points are rewarded to the player. Hence, tags are not auto-generated but added by players who tagged a specific code fragment before.

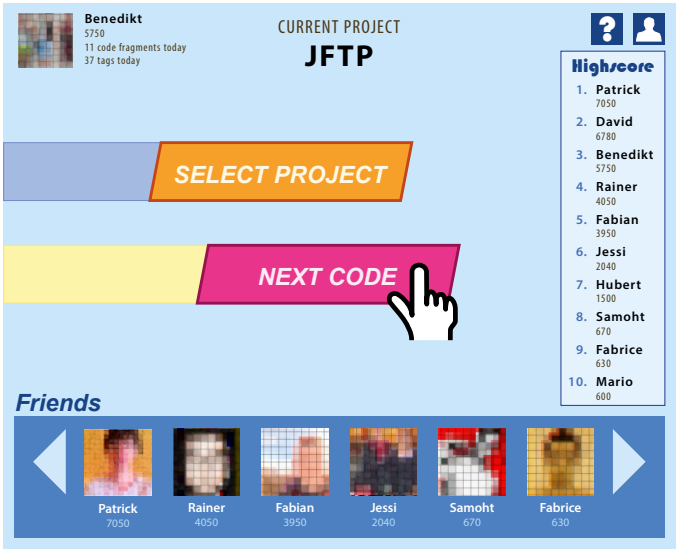

Fig. 1. Project selection screen displaying a project-wide high-score list (right), a friends high-score list (bottom), and player information (top left).

As this concept relies on tags given by others, we have to handle code fragments that have not yet received a considerable number of answers differently: Players get a small amount of points immediately for each new tag and, when enough players have tagged the code fragment, in addition, they receive the usual points. Thus, the high-score list is updated in retrospective.

All in all, for the players the goal of the game is guessing the most popular tags of given code fragments and eventually beating colleagues by getting on top of the high-score list. As an intended side-effect, on the other hand, the game yields tags for real code fragments that are contributed manually by human experts. At startup, players select a software project they are interested in to play against people that have selected the same project before.

B. Social Game Aspects

Competing with other players motivates to play the game regularly. As can be seen in Figure 1 (right), a high-score list shows the scores of all players for a selected software project. This is a popular and effective strategy to encourage players to compete. As in other social games, this effect is amplified by having friends as opponents: an additional friends high-score list (Figure 1, bottom) is provided. This list contains all friends that have already tagged any code fragment in the current project ordered by their scores. Information about the players themselves is displayed in the upper left corner. This enables the players to compare their scores directly with those of their friends and other players. Furthermore, friends can be invited by clicking on a button in the top right part of the screen, which opens a dialog with a list of friends that are selectable for invitation.

C. Tagging

After selecting a specific software project that should be subject of tagging, the actual game takes place in the code tagging screen. As shown in Figure 2, a code fragment drawn

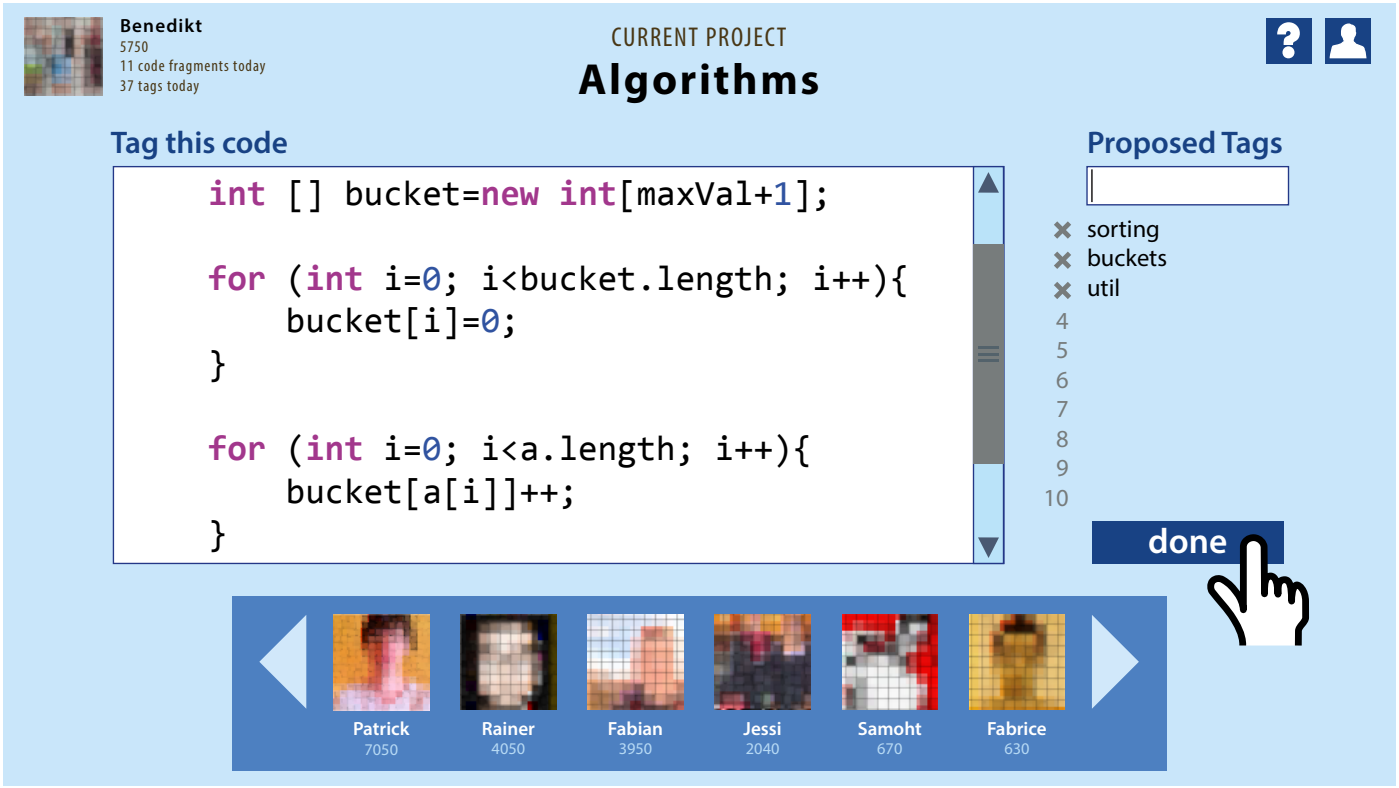

Fig. 2. Code tagging screen displaying a source code fragment (left), the player's tag list (right), and the friends high-score list (bottom).

from the project is displayed (left) and the player is asked to propose code tags (right). In order to keep the player from adding too many tags, the overall amount of tags is limited to ten. Thus, the players are forced only proposing tags of which they are convinced that they are also proposed by other players. A minimum number, however, is not required, thus the players are allowed to finish the current code fragment at any time. Already added tags can be removed again. Only tags are finally considered as the answer that are in the list after clicking *done*.

D. Selecting Code Fragments

Our social tagging game is intended to fulfill both being fun for players and producing high-quality tags for many code fragments. An initial idea was simply selecting the next candidate out of all code fragments randomly. But as already discussed above, before a top ten list for a specific code fragment can be generated, the fragment has to be tagged by some of the players; only then, the players can be rewarded with points. Since points are an important motivating factor, the game should reward players as soon as possible. Hence, code fragments that already received some tags should be preferred over yet untagged fragments in the selection process.

In our tagging game, a code fragment can have four different states:

1) *untagged* fragments have not yet been tagged by any player,
2) *started* fragments were seen by at least one player but still not enough to generate a top ten list ($< x$ players),
3) *incomplete* fragments have already a top ten list because at least x players has tagged it, and
4) *complete* fragments are already sufficiently tagged by y players ($x < y$).

Based on those states, we suggest the following rules for selecting the next code fragment:

(i) players can only tag fragments they have never played before,
(ii) every n^{th} selection is a *started* fragment or, if none is available, an *untagged* fragment,
(iii) every other selection is an *incomplete* fragment,
(iv) every selection is made randomly among all valid candidates,
(v) a selection is skipped if no valid fragment is available and the game ends for a player after n subsequently skipped fragments.

As a consequence, *complete* fragments will never be selected again. When a player has tagged all fragments of a project, the game naturally ends; at latest, the game is over for all players with respect to a certain project when all fragments are *complete*. The selection approach is a balanced trade-off to enable both competing with other players and receiving a sufficient amount of tags for as many code fragments as possible. Parameters x, y and n are customizable; a good default setting needs to be identified in user studies or simulation runs of the game.

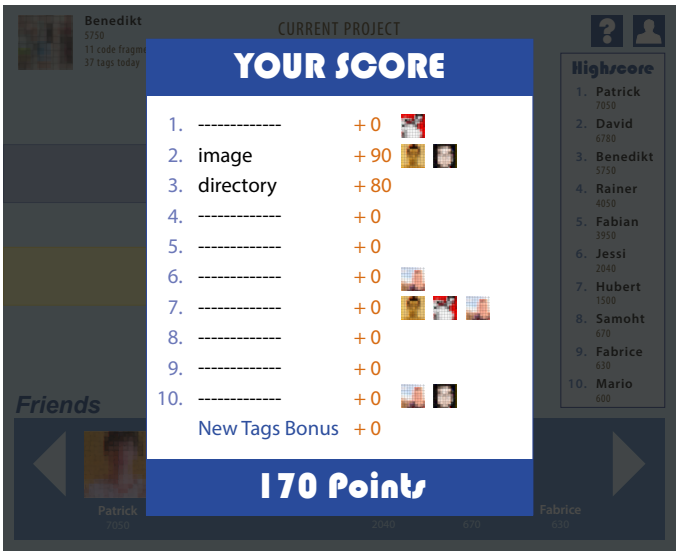

Fig. 3. Score board dialog similar to the score board in Family Feud with profile pictures of friends who also named tags in the top ten list.

E. Scoring

After proposing tags to a specific code fragment, a dialog with the score board is displayed (Figure 3). Similar to the score board in *Family Feud*, only tags are revealed that are proposed by the player. Furthermore, profile pictures of friends are displayed besides the tags they have named. According to the scoring scheme we are suggesting, a player will be rewarded with 100 points if the most popular tag on top of the list was named, and 10 points less for each subsequent tag. If a code fragment has no top ten list yet (i.e., not enough players have tagged the fragment), a player earns 10 bonus points for each contributed tag. After creating the top ten list for a code fragment for the first time, all players who tagged the fragment before will be rewarded retrospectively as following the regular scheme described above.

IV. Evaluation

After implementing a first prototype of the game, we performed a user study testing the game mechanics. The insights gained from the study are already reflected in the game design described above. Hence, the user study can be considered a *formative* evaluation being looped back into the development process—a *summative* evaluation producing concluding evidence has not yet been conducted. For the study, we asked 12 students in computer science (2 Bachelor, 8 Master, and 2 PhD students; 2 of them being authors of this paper) to play the social tagging game within a given time frame of a week. In contrast to the concept described above, we still used a simpler code fragment selection strategy: all players tagged the code fragments, regardless of their state, in the same order. The code fragments were randomly selected methods from the Java FTP client *JFtp* (version 1.0).

In total, 838 tags were used to describe 72 code fragments. Per fragment, each participant usually submitted between two and five tags, whereas at least one tag was also named by another player. We found that the code fragments were tagged on different levels of detail. On one hand, words of the code fragment itself, like Java keywords as well as identifier names, were used directly as tags. On the other hand, the current vocabulary was extended by using completely new words that reflect implemented concepts on a more abstract level (e.g., *getter/setter, method,* and *loop*). Most participants, however, reported that they felt uncomfortable in just copying words from the code fragment. Moreover, they had problems in finding reasonable tags that offer additional information to the given code fragments. At this point, perhaps it would be easier for the participants to see an application scenario first, where the proposed tags might be used (e.g., code search or documentation). More context might need to be provided, for instance, by making the surrounding code or additional documentation available.

We also found similar (and synonymous) tags that were treated as individual tags. Thus, it would be beneficial to merge similar tags, e.g., by using stemming, string metrics or a thesaurus. Another open issue, that we experienced, is to find suitable code fragments for the game at all. Not every fragment is interesting enough to be included in the tagging game and others are too complicated and cannot be sufficiently understood without providing a wider context. Nevertheless, an automatic approach for selecting and extracting code fragments is not required. If the social tagging game will be used, e.g., within a software project, it could be more beneficial to manually select code fragments (e.g., by developers) that have to be tagged next and also merge similar tags together by semi-automatic approaches. Moreover, the simpler code fragment selection strategy showed some weaknesses, in particular, for the players who tagged the most fragments so far: when they proceeded with the game, they only could tag yet *untagged* fragments and had to wait for others to receive their final points for these. The suggested, more sophisticated selection strategy discussed above circumvents the problem by distributing the introduction of new fragments to the game better among all players.

In the end, the study shows that the competitive environment worked well for motivating the participants, and especially that game elements also seem to be applicable for tagging tasks. About half of the participants played the game until they have reached the top of the high-score list at least once. In the coffee breaks, some participants proudly reported their current score and continued to challenge each other. Within one week, each participant contributed about 70 tags on average and read at least 36 code fragments of an external software project. Everything was done on a voluntary basis without stipulating the amount of working hours participants should invest in the study.

V. Conclusion

We presented a gamification approach for tagging source code fragments—here, methods of Java systems. Being understood as a social game, the approach rewards users if they suggest the same tags that others have already suggested. Thereby, the game mechanics foster conforming tags and quickly lead to reasonable descriptions of the code fragments. We implemented the approach as a *Facebook* plug-in and performed an initial evaluation.

The user study showed that the approach works in general and motivates participants in tagging code. While the tested

prototype implementation had certain limitations, the improved game design presented in this paper already partly addresses them (e.g., a better code fragment selection strategy); other issues stay open research questions, such as how to provide more context to allow players to come up with better code tags. The conducted study can only be considered an early, lightweight form of evaluation. Further evaluation will need to follow to find whether the suggested gamification approach outperforms previous manual and automatic solutions. Moreover, there are different variables and components of the approach that could be altered and tested: in particular, other scoring schemes or selection rules might improve the game; also, other gaming modes—for example, playing against a single other person or different levels of difficulty—could be explored.

We expect that the approach would be beneficial for open source projects where code tagging can be outsourced to non-core developers, for companies where the game can be used as part of the onboarding process of new developers, or for people teaching software development who can motivate students to read and understand real code. Furthermore, it would be interesting to extend the approach towards quality management: an atypical mismatch of tags could point to issues in program readability or players can be encouraged to use specific tags to indicate refactoring opportunities.

REFERENCES

[1] M. Ohba and K. Gondow, "Toward mining concept keywords from identifiers in large software projects," in *Proceedings of the 2005 International Workshop on Mining Software Repositories*, ser. MSR. ACM, 2005, pp. 1–5.

[2] S. Haiduc, J. Aponte, and A. Marcus, "Supporting program comprehension with source code summarization," in *Proceedings of the 32nd International Conference on Software Engineering*, ser. ICSE, vol. 2. IEEE, 2010, pp. 223–226.

[3] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, "A theory of aspects as latent topics," in *Proceedings of the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*, ser. OOPSLA. ACM, 2008, pp. 543–562.

[4] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using IR methods for labeling source code artifacts: Is it worthwhile?" in *Proceedings of the 20th International Conference on Program Comprehension*, ser. ICPC. IEEE, 2012, pp. 193–202.

[5] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby, "Shared waypoints and social tagging to support collaboration in software development," in *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, ser. CSCW. ACM, 2006, pp. 195–198.

[6] M.-A. Storey, L.-T. Cheng, J. Singer, M. Muller, D. Myers, and J. Ryall, "How programmers can turn comments into waypoints for code navigation," in *Proceedings of the IEEE International Conference on Software Maintenance*, ser. ICSM. IEEE, 2007, pp. 265–274.

[7] L. Von Ahn and L. Dabbish, "Labeling images with a computer game," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI. ACM, 2004, pp. 319–326.

[8] D. H. Goh and C. S. Lee, "Perceptions, quality and motivational needs in image tagging human computation games," *Journal of Information Science*, vol. 37, no. 5, pp. 515–531, 2011.

[9] S. Wang, D. Lo, and L. Jiang, "Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging," in *Proceedings of the 28th IEEE International Conference on Software Maintenance*, ser. ICSM. IEEE, 2012, pp. 604–607.

[10] C. Treude and M. Storey, "How tagging helps bridge the gap between social and technical aspects in software development," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE. IEEE, 2009, pp. 12–22.

[11] ——, "Work item tagging: Communicating concerns in collaborative software development," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 19–34, 2012.

[12] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR. IEEE, 2013, pp. 287–296.

[13] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC. ACM, 2014, pp. 279–290.

[14] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for Java classes," in *Proceedings of the IEEE 21st International Conference on Program Comprehension*, ser. ICPC. IEEE, 2013, pp. 23–32.

[15] 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*, ser. MindTrek. ACM, 2011, pp. 9–15.

[16] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work?–a literature review of empirical studies on gamification," in *Proceedings of the 47th Hawaii International Conference on System Sciences*, ser. HICSS. IEEE, 2014, pp. 3025–3034.

[17] E. B. Passos, D. B. Medeiros, P. A. Neto, and E. W. G. Clua, "Turning real-world software development into a game," in *Proceedings of the Brazilian Symposium on Games and Digital Entertainment*, ser. SBGAMES. IEEE, 2011, pp. 260–269.

[18] L. Singer and K. Schneider, "It was a bit of a race: Gamification of version control," in *Proceedings of the 2nd International Workshop on Games and Software Engineering*, ser. GAS. IEEE, 2012, pp. 5–8.

[19] W. Snipes, V. Augustine, A. R. Nair, and E. Murphy-Hill, "Towards recognizing and rewarding efficient developer work patterns," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE. IEEE, 2013, pp. 1277–1280.

[20] D. J. Dubois and G. Tamburrelli, "Understanding gamification mechanisms for software development," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. FSE. ACM, 2013, pp. 659–662.

[21] S. Sheth, J. Bell, and G. Kaiser, "A competitive-collaborative approach for introducing software engineering in a CS2 class," in *Proceedings of the 26th Conference on Software Engineering Education and Training*, ser. CSEE&T. IEEE, 2013, pp. 41–50.

[22] N. Tillmann, J. De Halleux, T. Xie, S. Gulwani, and J. Bishop, "Teaching and learning programming and software engineering via interactive gaming," in *Proceedings of the 35th International Conference on Software Engineering*, ser. ICSE. IEEE, 2013, pp. 1117–1126.

[23] J. Goncalves, S. Hosio, D. Ferreira, and V. Kostakos, "Game of words: Tagging places through crowdsourcing on public displays," in *Proceedings of the ACM Conference on Designing Interactive Systems*, ser. DIS. to appear, 2014.

[24] J. Gomes, T. Chambel, and T. Langlois, "SoundsLike: movies soundtrack browsing and labeling based on relevance feedback and gamification," in *Proceedings of the 11th European Conference on Interactive TV and Video*, ser. EuroITV. ACM, 2013, pp. 59–62.