Experiences Gamifying Code Navigation

Are development practices a habit?

Will Snipes
ABB Corporate Research
Industrial Software Systems
Raleigh, NC USA
will.snipes@us.abb.com

Anil R. Nair
ABB Corporate Research
Industrial Software Systems
Bangalore, India
anil.nair@in.abb.com

Emerson Murphy-Hill
North Carolina State University
Department of Computer Science
Raleigh, N.C. USA
emerson@csc.ncsu.edu

ABSTRACT

As software development practices evolve, we face the continuous challenge of communicating new practices and tools to the community. As developers, we both like to try new things and like to stick with the familiar. In addition to training, discussing, and presenting software engineering practices and tools, we propose a method to both motivate and monitor their adoption through a tool embedded in the Integrated Development Environment (IDE).

Blaze monitors developer practice and tool use over the long term and reports the lasting change a communication effort makes in everyday habits. Embeding Blaze in the developer's work environment as a Visual Studio extension provides the capture of developer work patterns while giving feedback to users on their practices. Blaze encourages everyone to use the new practices by assigning points to specific commands and tools. With a rich data source, we evaluate developer practices, define metrics that give individual feedback, and create an environment for developers to see how they are doing compared with their teammates.

Data collected from the pilot group show a dichotomy of navigation and code understanding practices used by developers even the same developer in different situations. We observed that the duration of an edit session is more correlated with categories of activity that benefit from improved practices and tools. The Blaze tool was well received by most participants, however, the intended improvements in navigation practices promoted by the pilot did not occur. We conclude that more proactive interventions are necessary to move developers to change their code navigation habits.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

International Conference on Software Engineering 2014 Hyderbad, India

General Terms

gamification

Keywords

empirical, training, learning, usability

1. INTRODUCTION

Developers with years of experience develop habits and ways of doing things that are difficult to change even when they might recognize better methods. For instance, the commands they use and ways they use to navigate the code base in the development environment make them less efficient than using newer practices. Robillard et. al. show that developers who think of code as a graph and use commands that allow them to follow code structure are more effective and efficient than developers who browse through code. [15]

Perhaps the persistence of their habit is why developers are hard to turn from their old practices. Developers may need some coaching to continue a new practice following an introduction. To continuously coach the developer, we conceived of a tool in our prior work to provide constant feedback and motivate them to adopt a new practice. [20] This tool creates a competitive game that gives continous feedback to the developer continuously on adopting the improved practice. The more they change their habits, the higher they score in the game.

Taking a step back, gamification is a popular movement that people apply to customer oriented web sites and social games. Why would software developers writing serious software be interested in this? Our first step was to challenge this question through a survey that queried both whether developers would be interested in gamifying their workspace and what aspects of that space would be most interesting to them.

With these questions answered, we identified a set of practices and defined a game meeting expectations from the developers feedback. The game, implemented in a Visual Studio add-in called Blaze, integrates into the Visual Studio workspace and establishes a light anonymous competition between developers to improve their navigation practices.

During the pilot of Blaze we observed the variation in developer navigation practices and identified some potential reasons for this variation. We evaluated developer's acceptance of game elements in the tool and their desires for addi-

tional functionality. While further data is required to assess the impact of the tool, the initial developer acceptance encourages an expanded pilot.

The rest of the paper is organized as follows: Section 2 reviews the preliminary survey-based assement, Section 3 Discusses the design considerations for developing the Blaze tool/game, Section 4 Discusses the design of the study and participant selection, Section 5 is Related work, Section 6 results from the pilot, Section 7 Threats to Validity, and Section 8 has our Conclusions.

2. ASSESS THE VIABILITY OF GAMIFICATION FOR SOFTWARE ENGINEERING

2.1 Assessment Background

When discussing gamification, folks raise a concern about replacing an activity that is intrinsically motivated with extrinsic motivation provided by points and achievements. To discover what is behind this concern, several studies are useful to understand what should be done.

Early studies attempt to evaluate achievement motivation in students and relate that to their performance in school. The study by Hermans describes results from applying questionnaire to evaluate achievement motivation in students [**?**]. Each question is ranked for its correlation to achievement motivation. Some key questions about the diligence with which students approach their work are correlated with achievement motivation.

Maehr proposes an affirmative theory whereby individuals achieve as a member of a social group choosing the behavior that will meet the expectations and values of the group that is significant to them. [9] The presence of a social group where the person is motivated to belong and excel within is important for creating achievement motivation. Maehr states that since achievement behaviors can be triggered by circumstance, that they are present in most people regardless of background.

Beecham and colleagues provide a thorough analysis of existing literature on studies of motivation in Software Engineering [**?**]. It has some valuable lists of factors that motivate software engineers with the most common being "the work" motivates us. The list of de-motivators is also useful with common job satisfaction items like stress and inequity in recognition, plus poor quality software (low accomplishment) and lack of influence in decision making. The paper also lists characteristics of people in the professions including need for independence (autonomy), desire to learn new skills/tackle challenges.

Understanding the concerns with maintaining intrinsic motivation while adding elements of achievement motivation, we developed an assessment survey questionnaire to determine whether the community would be receptive to a gamification approach to software development. We designed the survey to assess how developers are motivated to use training, practices, and tools and whether they are open to having that information shared with their peers and managers, and being recognized with badges and points. The developer study conducted with over 130 software developers at ABB challenged whether achievement motivation would help software engineers and whether they would accept sharing data on tools and practices they use to help others.

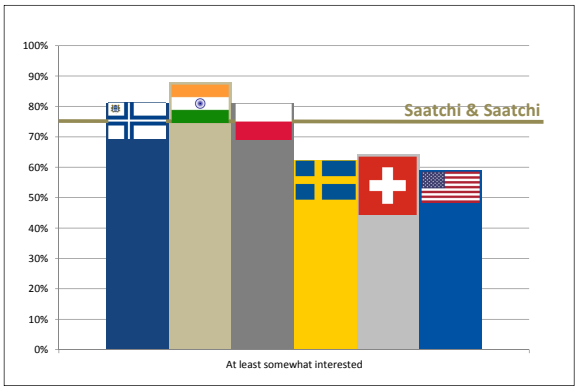

Figure 1: Acceptance of gamification

2.2 Assessment Results

Using a question from the Saachi and Saachi survey [16] as directly as possible, we asked developers at ABB "How interested they would be in working for a company that incorporated some aspects of games into software engineering tools as a way to increase productivity in the workplace"? We segmented results by country in Figure 1 and found that gamification of software engineering is more interesting to developers in India, Poland, and Finland than in other major countries. The responses indicating that developers were at least somewhat interested is 75-88% In India, Poland, and Finland. For the US, Sweden, and Switzerland the responses indicate 60-65% are at least somewhat interested. The overall response from our survey is comparable to the Saachi and Saaschi survey [16] upon which this question was based that found 75% of 18-45 year olds were at least somewhat interested.

We asked developers how likely they would be to try tools and practices recommended to them through an automated usage tracking system. Answers showed that 95% of the developers surveyed are likely to try tools and practices recommended to them. This indicates that a recommendation system would positively impact the deployment of software engineering tools and practices at ABB.

One area we questioned was whether sharing detailed usage information with colleagues and the company as a whole would be a concern for developers. A graduated scale was used for the community that would be shared with and distinction between whether that sharing was anonymous or the information was personally identifiable. Figure 2 shows that over 90% of respondents are comfortable with sharing either anonymous or non-anonymous information with their team members. Their comfort level decreases as the scope of who the information is shared with increases. People were less comfortable sharing with anyone (the category for people outside ABB) particularly if the data were not anonymous. Two conclusions from this feedback are that people are very willing to share information that could help their team. Second, sharing within the company is acceptable if we take care to make the data anonymous.

Developers were asked what would motivate them to use tools and practices and given 4 choices to rank. The top motivator for using tools and practices is coworker recommendation with 75% of respondents in Figure 3 ranking in

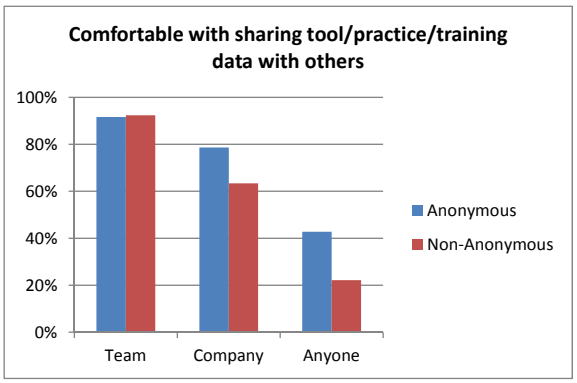

Figure 2: Comfortable with sharing usage information

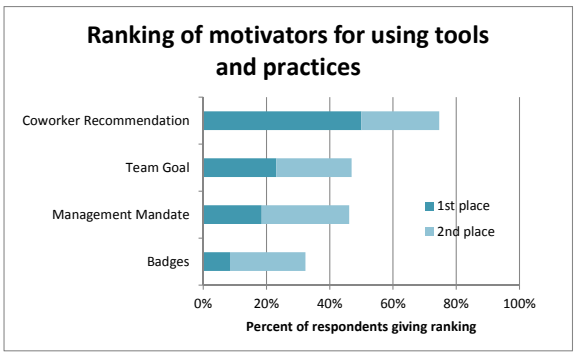

Figure 3: Motivation for using tools and practices

1st or 2nd place. The lowest was having badges posted on their social profile where 33% ranked this in 1st or 2nd place. Badges also ranked low in motivations for taking more software engineering training. We conclude that a facility to share coworker recommendations on tools and practices has the best opportunity to increase their adoption. Since a competition among teams ranks slightly higher than management mandate, we have the choice to apply one or the other or both as motivation for using key practices.

Developers were asked to rank awards for different actions by which they would find the most desirable. Overall 68% of developers ranked receiving an award for consistently performing quality practices as either 1st or 2nd place. This supports a conclusion that focusing on awarding quality practices would motivate developers to do them consistently.

- 95% of developers responding would try tools and practices suggested by an automated recommendation system particularly if they were provided in context with the current workflow pattern the developer follows.

- Responses indicate a preference for team competition over individual awards (badges) and awards for quality practices ranked highest. So, the tool will incorporate team competition ideas and recognize quality practices.

- Over 90% of respondents approved of sharing data with their team members

3. GAME DESIGN

The pilot evaluates the aspects of game implementation as well as the study of the results of their results. Here we discuss how the aspects of selecting a practice suitable for gamification, designing the game, and competitive element considerations.

3.1 Select a Practice

In their paper on game design patterns, Hamari et. al. issue a warning that assigning achievements to required tasks reduces intrinsic motivation because player autonomy is reduced by the achievements [6] . This warning indicates that we should avoid giving points for things that are already an expected deliverable such as finding or fixing bugs, completing development tasks, or meeting required process criteria. Practices to consider are those practices with room to go beyond a checklist or minimum criteria where developers have the opportunity to excel.

Selecting a practice to study involves identifying a practice, defining specific characteristics of the practice to measure, and identifying the actions in Visual Studio that relate to the practice. We currently limit ourselves to practices that exist entirely in the IDE thus can be monitored using Blaze. We identified potential practices of test-driven development, debugging practices, navigation practices, eliminating static analysis bugs, and check-in frequency.

Practices of check-in frequency, static analysis bug elimination are highlighted in this paper's related work section with prior gamification studies in classroom environments. Johnson and Kou achieved automated monitoring of test-driven development practices with Zorro thus may provide a candidate for future gamification application. Debugging practices have developer community opinions on optimal methods, however, detailed debug practices with modern IDEs have not received research community attention, thus are also reserved for future work.

Based on studies of structured navigation by Robillard et. al. that identify structured navigation as a practice leading to effective program maintenance [15], we identified structured navigation as a key practice. Structured navigation fits the requirement of a practice that is contained in the IDE, is something that has proven benefits, and lends itself to measurement and scoring, is not an assigned task, and leaves room for developers to excel.

3.2 Design the Game

To setup the game for a practice, the Blaze tool provides an XML configuration file where the researcher can configure Blaze to categorize and assign points to the commands that are part of a software engineering practice. Blaze allows the researcher to define multiple command category levels. Thus we can categorize commands that are navigation then further classify them into structured and unstructured navigation.

We categorize commands as structured navigation when they allow the developer to locate a code element following the program structure from another element. For example, Navigate To *Ctrl+,*, Go To Definition *F12*, and Find All References *Ctrl+K,R* are commands that follow structure of the code, which we classify as structured navigation.

Commands that we considered as unstructured navigation include scrolling through a file using keys or the scroll-bar, opening files from the Solution Explorer, Find commands (e.g. Find in Files), Go-to line number, and children of these commands (e.g. Find Next).

Gamification practices consider the feedback of achievements and points as critical to a successful outcome. The design of achievement awards should gradually encourage the participant to reach higher and higher levels of success in the activity. Achievements must feel earned to the participant so that internally they recognize the effort to get the award was significant and feel satisfied [6]. Hamari and Eranti describe a general relationship between achievements in games and regular game play In games achievements are typically a parallel scoring system to the main game play. They make the game more engaging providing multiple ways to increase your score and multiple challenges in one interface.

Using these guidelines we assigned points to structured navigation commands which received 1-10 points each while unstructured navigation commands received 0 points. Assigning negative points is discouraged as this may demotivate folks who do not have detailed knowledge of how the points are assigned. [] In creating levels, following the guidance from Hamari [6], we built levels following an exponential curve. Users initially level up after a working day or two extending to achieving the next level after a week or even a month following an exponential curve. The levels make the game motivating for beginners and challenging for more experienced developers. We designed the points and levels such that above average developers can complete all levels during the study period.

3.3 Competitive Elements

By selecting an intact team, we hope to leverage aspects of motivation such as the Leaderboard to spur feelings of competition among teammates. The leaderboard was presented so that developers could see their own score in relation to the top 5 people on the leaderboard. However, the identifier of the other participants was generated as 3 capital letters assigned based on their position on the leaderboard. Therefore the assigned letters could even change when an individual's ranking changed. Not knowing who of your colleagues was ahead of you could be a factor in motivation, but also avoids one potential source of conflict in an intact team.

Competitive dynamics may be more effective when teams from different areas can compete as a team against other teams.. This tried and true method both builds esprit de corps and enhances achievement towards the intended outcome. This study design is potential future work.

4. STUDY DESIGN

Utilizing the unobtrusive continuous monitoring capabilities of Blaze, we designed the pilot to reveal feedback to the developer in 3 stages of 1 week duration each. We employed the three stage model defining the first stage as a baseline capturing the current practices of the pilot group. The second stage employed the intervention that included an informative web site on structured navigation commands in Visual Studio, information about the Sando search tool, and information about how points would be awarded during the pilot. The third stage provided the participants with continuous feedback in the from of points accumulated and an indicator arrow **??** that points up when their navigation ratio is higher than their historical average.

4.1 Intervention Staging

Part of the research objective was to determine whether we can achieve adoption of a practice with alternative interventions that are more deliberate than word of mouth and perhaps more scalable than webinars or presentations. As previously discussed we sought to provide Blaze as a means to influence developers to use structured navigation practices. We constructed the interventions during the pilot to roll out in stages each week that they use the tool. This allowed us to establish a baseline and determine the changes as the pilot continued.

In the initial stage, participants were informed of the purpose of the study what to expect from the Blaze tool. The participants installed the tool in their Visual Studio environment. The tool did not present any window for them to see and simply collected data in the background.

After a week's data collection marking the end of the first stage, Blaze started to automatically pop-up a window containing a button for a web page with information. The page included a link to download another tool for code search called Sando [17] and information about using that tool in a video. The sub-page on navigation contained a page of tutorial on the built-in structured navigation commands in Visual Studio. The About Blaze page contained details of the point scoring system and background on Blaze's purpose. The usefulness of the communication site is rated as part of our study completion survey.

In the third stage developers received instant feedback from Blaze on the points they accumulated during the development session. The up/down arrow provided a simple indicator on whether they were increasing their use of structured navigation or were using it less than their personal historical average. This final configuration continued until the end of the pilot.

4.2 Pilot Participants

As mentioned in fig. 1, India is one country where as per our survey, the acceptance to gamification is the highest. Hence, we conducted this study with an intact team of 6 developers working in an R&D facility in India on a large industrial software system. We approached a team which develops and maintains an industrial software product. Initially, we met the whole team along with the management staff to explain the research project and the Blaze tool we developed. We explained the aim of the study, what data is collected, and how the data will be processed and managed.

It was important to gain the confidence of the participants that their data were kept confidential so their activities would be close as possible to real-world. if the participants felt like they are being tracked and the data is going to use for some study that would identify them to their peers or managers, then they may deviate from the natural development style.

We assured the team about the confidentiality of the data and the management also informed the team that the data collected by the research group will not be used for any other purpose other than the study. Developers were asked to volunteer for the pilot by their management, however, managers did not know who participated in the study. Their name and data were kept confidential to ensure that no one

apart from the research team knows that he/she is participating in the pilot. Mechanisms in our data collection to identify individuals were under the control of each developer, they could chose to share or keep confidential the association of a generated unique id with researchers or each other.

Management wished to have no knowledge of who was participating and took great care to remain detached from the pilot proceedings. Data at this level of detail has not been collected in ABB development organizations except for a few minutes of video study that might be conducted. Researchers were careful to create the system so that developers controlled their identification with the data and fortunately, management bought into the necessary anonymity of the participants.

4.3 Pilot Survey

The goal for the pilot follow-up survey was to collect feedback on the tool and the effect it had on the developers' navigation practices and knowledge. The survey questions were segmented into portions that evaluate prior and post knowledge and practices for navigation, key steps in the pilot itself, the influence of game elements on their navigation practices, typical tasks and demographic information.

The key questions about this pilot were did we have an effect on knowledge of the developers regarding structured navigation commands. We asked them to rate their knowledge of structured navigation commands prior to the pilot on a 1-10 scale to understand their baseline. We asked them to contrast their knowledge after the pilot to assess what they learned from the pilot.

The other key question is did we have an effect on their navigation practices. Here too we asked them to rate how much they used structured navigation prior to the pilot using a 4 choice rating of not at all, some, many times, and as much as possible. Then we asked whether the used structured navigation commands about the same amount, more, or a lot more during the pilot. These two questions assessed their intent in terms of using structured navigation commands and we checked that with the data.

A key set of questions focused on whether using Blaze increased their attention to navigation commands by asking how much specific game elements influenced their practices. The scale started with not at all, a little, some, to a lot.

The other questions included whether they reviewed the training materials on structured navigation commands, installed the Sando code search tool, and what types of development tasks they would consider using structured navigation for in general. Another demographic question is how well they know the code they work in rated from "I work in code that I wrote", "I maintain code that I am familiar with", to "I recently learned the code"

Survey results are discussed in the results section **??**

5. RELATED WORK

5.1 Monitoring Practice Studies

Johnson and Kou defined Zorro [7], a system for detecting whether developers use Test Driven Development techniques based on data from Hackystat. Zorro divides development activities into episodes delimited by events such as configuration management code check-in, start of a unit test run, or start of a build. Using the distinct events that developers follow within these episodes, Zorro determines whether the episode followed Test-Driven Development practices per their classic definition of test first - code - test pass, or a different scenario. In two student-based studies comparing Zorro classifications with a simultaneous observational screen video, Zorro achieved between 70% [8] and 89% [7] accuracy in classifying episodes into their proper TDD scenarios. The studies did not, however, attempt to use or evaluate the influence of instant feedback aimed at encouraging participants to follow the classic definition of Test Driven Development. In our study, we attempt this leap to present data collected directly to developers attempting to influence them to change their practices.

Murphy-Hill, Parnin, and Black [13] use the Mylyn Monitor to explore whether or not developers use the automated refactoring tools present in Eclipse. They look for specific refactoring commands in Eclipse and determine the amount of time developers use tools versus hand refactoring the code. For this study we build a tool that captures the use of all commands within the Visual Studio environment including refactoring, navigation, and edit commands.

Robillard, Coelho and Murphy explore hypotheses around how developers can be more effective at performing a maintenance task [15]. Key conclusions are that developers are more more successful finding and fixing bugs when they create a detailed plan for implementing a change, use structured navigation with keyword search or cross-reference search, and only review methods once during their search. This builds on this work by testing methods for increasing the use of structured navigation in developers' everyday practice.

The Pro Metrics (PROM) tool provides a framework for collecting data for further analysis from tools used by developers. [3] It provides a flexible data model and a plug-in architecture to facilitate collection from different data sources. Studies conducted using prom include a series of studies by it's creators on trends in time spent Pair Programming [2], benefits of refactoring on productivity [10], impact of refactoring on re-usability [11], and prediction of effort [1]. The PROM tool and studies that use it track the time spent editing files and methods from the IDE and code metrics obtained trough source code analysis. Applying these data to specific research questions such as whether refactoring improves productivity [10] shows the utility of combining automated effort data with code metric data. Taking a different focus on techniques, our study used more detailed events captured from the IDE to detect how the user was navigating through the code in addition to knowing how long the editor is open for a particular file.

Murphy-Hill et. al. study a large usage history data set and apply several different algorithms to accurately suggest to users new commands in their environment to use. [12] Algorithms based on command history performed well particularly when synchronized chronologically with the recipient's usage history. Novice users were better served by recommendation algorithms that did not require a long history, where more expert users benefited from more sophisticated algorithms that included more history. The "Most Widely Used" algorithm that recommends commands based on the collective usage profile of the team performed nearly as well as the more sophisticated algorithms. The paper presents the methodology for making recommendations and the evaluation of the accuracy of the recommendations. Our work goes the next step of evaluating whether a specific recom-

mended practice gets adopted by developers when influenced through gamification methods.

5.2 Gamification

Singer and Schneider demonstrate the use of a message board and points for encouraging students to increase their frequency of commits to the source code repository. [19] The communication mechanism enabled students to see each others' progress and resulted in more frequent commits than baseline. Participants valued the communication and collaboration aspect and some valued the competition enough to change their opinion on optimum commit frequency. The subsequent thesis by Singer [18] describes results of an experiment conducted across two iterations of a class where active feedback on commits was deployed to one course and another course served as the control group where commit frequency was simply monitored. Results show an increase in the frequency of commits at a statistically significant level. This study uses fine grained instrumentation to identify practices used between commits such as structured navigation.

Dubois and Tamburrelli discuss theoretical concepts behind applying gamification to software development. [4] Their approach is to consider how gamification could be applied throughout the software development life-cycle to motivate many practices in different phases. They provide guidance on assessing gamification in the software development domain that includes analyzing the actors and game mechanics, defining integration with existing tools and processes, and designing the evaluation of results. Dbois and Tamburrelli demonstrate this guidance using a study of students working on a class project. Defining a control group and subject group of students, both groups receive the same feedback about their code comments, test coverage, code size, and static analysis rule violations. The subject group can see each others' rankings while the control group only sees their personal rankings. Results though inconclusive indicate that competition increases the desired results. In our study we apply both individual feedback and competitive based feedback (anonymized) to participants and seek to observe a change in their behavior after an intervention.

6. RESULTS

6.1 Pilot Data Observations

Blaze logs events from actions the developer takes or actions from Visual Studio itself. The event is basically a GUI event that the program manages with event handlers that are registered with the application to listen for the event. Blaze becomes a global event handler listening for all events by registering for them in Visual Studio.

Blaze records key attributes along with each event in visual studio. The log captures a time-stamp for each event normalized to GMT. The name of the event provides a low-level classification. The event type reflects an internal classification in Blaze. An optional field for Artifact Reference captures data such as the file-name being edited and the currently selected line number. Finally an anonymized unique identifier for the user of blaze allows us to investigate differences between developers.

Analysis of the log extracts several key information bits from this raw source. First we classify the event names at a higher level into categories related activities such as Navigating, Reading, Editing, Building, Debugging, Testing, and using a known Tool.

The second information we extract is aggregating the events along the time-line into sessions. A session is an abstract concept that groups multiple events into a sequence with a beginning and an end. Of the many possible ways to define sessions, we chose to define sequences as all the events that transpire before and during a set of edits to a specific file. When the developer edits a new file, all the events after the last edit to the previous file are part of the new sequence for the new file. Although not at the method level, this is similar to the say Robillard et. al. generated sequence based on located methods for their study. [15]

6.1.1 Observations About Developers' Habits

We are interested in understanding whether developers are consistently using practices that make them inefficient. One way to check that is to determine whether our measurements are associated with specific developers in the study, in other words, can we independently classify the edits performed by each developer. If this is possible, then we can discuss whether developer have consistent ways of navigating and understanding for each edit. Thus the hypothesis is developers have consistent ways of navigating and understanding code while they are making changes.

To test this hypothesis we construct a classification tree using Weka [5] with the J48 classifier. Slicing the data and training the model using 10-fold cross-validation, the model identifies 53% of the 647 edit sessions with the correct developer among 6 participants. The tree uses nearly all available fields including time for all categories and time of day for edits in its attempt to associate sessions with their developer generating a Kappa statistic of 0.403 and square root of the relative squared error at 101%.

While there is some consistency to how developers work, we cannot say that individual developers are always consistent in their work patterns. Thus developers are somewhat flexible in the commands they use when navigating and making changes to code and may be more open to adopting new tools and techniques.

6.1.2 Observations About Drivers of Edit Session Duration

We selected navigation as the subject of the pilot game because it was identified by Robillard et. al [15] as correlated with developers who effectively completed a maintenance task. Are we focusing on the right area of developers' activity in Visual Studio?

In order to test whether navigation is an important area to address, we construct a hypothesis that use of unstructured navigation is an important factor in developer productivity. Edit session duration, we assert by opinion, is a factor related to developer productivity though there are many other factors we could consider. With the data available from Blaze, we can explore the categories of events that correlate with the duration of edit sessions over the period of the pilot.

Using Weka [5], we performed an attribute selection process to identify the most significant attributes related to edit session duration. The attribute search method we chose was Greedy Step-wise search that selects attributes based on their correlation with the target and stops adding attributes when the next attribute decreases the correlation. The process identified the top 5 out of 10 possible categories

	Df	SSQ	MeanSq	Fval.	Pr(>F)
Unstr. Nav.	1	0.039	0.03920	1383	<0.001
Read	1	0.017	0.01741	614	<0.001
Residuals	645	0.018	0.00003		
Total	647	0.074	0.05664		

Table 1: ANOVA for Edit Duration

as follows:

1. Unstructured Navigation
2. Read
3. Build
4. Debug
5. Edit

Using R [14], we constructed a linear regression model for Unstructured Navigation to determine the correlation with edit duration. The linear model found a positive correlation with unstructured navigation with a p-value <0.001 on 646 degrees of freedom. The model with only unstructured navigation had an R-squared of 0.52 showing a good portion of the variation is explained by that category. In order to have a more acceptable explanation of edit duration, we conducted a multiple linear regression model with both Unstructured Navigation and Read categories. The Read category of events captures the activities the developer is doing to look through the code in the editor. They are navigation commands that move incrementally like pressing the up/down arrow keys, or moving the scroll-bar an incremental amount. Combining these two categories results in a linear model with a p-value <0.001 and an R-squared of 0.76, more acceptable for drawing conclusions.

Conducting an ANOVA analysis shown in Table 1 for this regression model, we can explore the contribution of Unstructured Navigation and Reading to the model. The contribution of each variable can be determined by calculating the portion of the total variation that each variable explains. Unstructured Navigation explains 53% of the variation. Out of the remaining variation found by subtracting the amount explained by Unstructured Navigation from the total, Read explains 49%.

After this analysis we cannot disprove that unstructured navigation is correlated with edit duration, thus it is an important factor in developer productivity. Combining unstructured navigation with reading describes a larger portion of the overall edit duration and motivates us towards improvements in both navigation methods and code comprehension.

6.1.3 *Observations About navigation ratio improvement*

To evaluate whether developers improved their practices for structured navigation we established a metric Navigation ratio as the number as the number of structured navigation events in a session or period of time over the number of unstructured navigation events.

Structured navigation events included:

- Navigate To (Ctrl+,) is a fuzzy search interface that locates symbols matching your search terms.

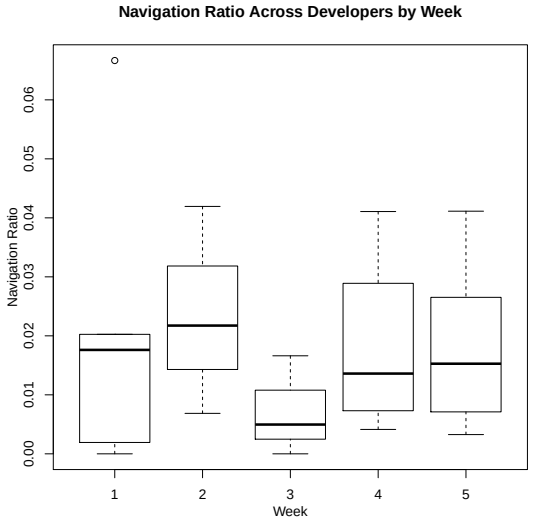

Figure 4: Navigation Ratio by Week

- Go To Definition (F12) brings up the code where the selected symbol is defined.
- View Call Hierarchy (Ctrl+K Ctrl+T) provides a two way analysis of a symbol's dependencies and uses.
- Class View (Ctrl+W, C) priovides a searchable browser of classes and class hierarchy
- Find All References provides a list of refernces for an object
- Navigate to Event Handler brings up the event handler from the XAML editor
- View Class Diagram generates a class diagram for the project
- View Obeject Browser is a search tool for objects in the solution

Unstructured navigation events included selecting a file in an explorer window or selecting the tab for a file, using arrow and page up/down keys to go up/down through a file, scrolling, clicking on a file element, and using any of the built-in find commands such as Find In Files or Quick Find.

During the pilot we staged the deployment of information feedback to developers so they would gradually improve their structured navigation practices. Our hypothesis was that this instant feedback and information would result in an increase in the use of structured navigation commands in Visual Studio.

To evaluate whether developers use more structured navigation when they receive instant feedback, we monitor the change in navigation ratio over the pilot study period. Expecting that as developers learn about navigation commands and tools that count towards the score in the second week they will begin using the practices more. Also expecting that noticing the leaderboard competition will encourage some developers to use the navigation practices more.

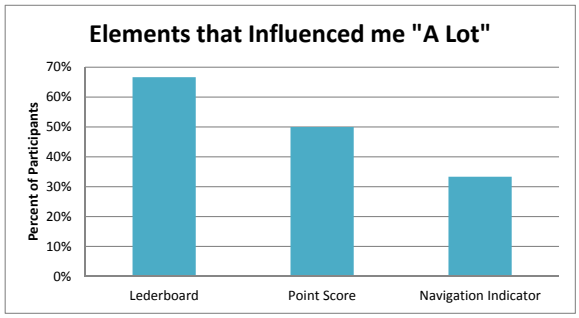

Figure 5: Influence of Feedback Elements

What we observed visually in Figure 4 is that the structured navigation practice did not increase significantly from the beginning to the end of the study. Overall, we did not see a significant effect towards increasing navigation ratio for the interventions in week 2 or week 3. Weeks 4 and 5 that follow the formal period of study show the navigation ratio continues in a similar distribution pattern.

6.2 Pilot Survey Results

Our post pilot survey attempts to shed some light on the reasons behind the consistency in navigation ratio over the pilot.

On the assessment of structured navigation knowledge, half of the participants replied that they had very good prior knowledge about structured navigation and they use structured navigation as much as possible. Other half of the participants have some understanding on structured navigation use it some of the time. Even developers whose navigation ratio was lower than average, used structured navigation occasionally as much as their more frequently using colleagues. While analyzing the points they scored with Blaze tool, we observed that the participants who mentioned that they have good knowledge in structured navigation are on top of the points table. Incidentally, while cross verifying their performance on the coding, they are better productive compared to the participants on the bottom of points table.

The chart in figure 5 shows how the individual feedback features influenced developers to think about using structured navigation. Four out of six developers said the leaderboard influenced them "a lot" and all rated the leaderboard as having some influence. Three out of six said their individual point score influenced them "a lot" and all said the score had some influence. The element with mixed ratings was a graphical indicator on whether they were improving over past use of structured navigation commands. The indicator received the influencing "a lot" rating from one participant and a "not at all" influence rating from one participant with the rest having some influence. The results indicate the more obvious and more clearly comparative information in the leaderboard was the strongest influence for developers.

The participants requested that the system to provide more feedback on tips to help them increase their score. They indicated that more obvious feedback such as hints that pop-up when they launch the tool would help them more than passively provided information. Their desire for personal metrics and historical views of the data encourages us that this detailed level of measurement and analysis is perceived as helpful. Features to meet these requests are part of future work.

There was also feedback that people wanted more information presented to them by the tool on what structured navigation commands to use, so they may not have closed the knowledge gap during the pilot.

7. THREATS TO VALIDITY

There are several threats to validity for this result. By making the purpose of our measurement and contest known although without details we may have triggered the participants to think about navigation commands they use before establishing a baseline. We attempt to control for this by comparing the pilot participants' data to data from Blaze users who are knowledgeable Visual Studio developers but did not receive the interventions. Another threat is the intact team represents people with different backgrounds and experience levels who may have established habits for navigating through source code. We attempt to control for this by asking post survey questions of the participants about their familiarity with the code they work in and their experience levels. The threat of non- generalizable of results remains particularly because the pilot was conducted within an intact team in a single industrial site in India. As the pre-study questionnaire shows, the developer's desirability for a gamification type of intervention vary by country.

8. CONCLUSIONS

Developers accept gamification ideas into their environment even when coupled with detailed development activity monitoring. We identified specific concerns particularly around maintaining anonymity of the developer as their data are shared with other developers. The participants' feedback indicates a desire for more instructional hints on how to succeed in the game similar to how real games provide hints when the user might be stuck. Implementing a more comprehensive set of practices in the game would expand the usefulness of Blaze as a training tool. Other ideas might include making the game more subtle so that it is not as obvious a competition, but builds the elements of self-improvement that move the experience from an extrinsically motivated one to an intrinsically motivated one.

9. ACKNOWLEDGMENTS

The authors wish to thank ABB and the pilot participants for their support of this research.

10. REFERENCES

[1] P. Abrahamsson, R. Moser, W. Pedrycz, A. Sillitti, and G. Succi. Effort Prediction in Iterative Software Development Processes – Incremental Versus Global Prediction Models. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 344–353. IEEE, 2007.

[2] I. Coman, A. Sillitti, and G. Succi. Investigating the Usefulness of Pair-Programming in a Mature Agile Team. In P. Abrahamsson, R. Baskerville, K. Conboy, B. Fitzgerald, L. Morgan, and X. Wang, editors, *Agile Processes in Software Engineering and Extreme*

Programming, volume 9 of *Lecture Notes in Business Information Processing*, chapter 13, pages 127–136. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[3] I. D. Coman, A. Sillitti, and G. Succi. A case-study on using an Automated In-process Software Engineering Measurement and Analysis system in an industrial environment. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 89–99. IEEE, May 2009.

[4] 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*, ESEC/FSE 2013, pages 659–662, New York, NY, USA, 2013. ACM.

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1), 2009.

[6] J. Hamari and V. Eranti. Framework for Designing and Evaluating Game Achievements. In *Digra 2011 Conference*, Sept. 2011.

[7] P. M. Johnson and H. Kou. Automated Recognition of Test-Driven Development with Zorro. In *Agile Conference (AGILE), 2007*, pages 15–25. IEEE, Aug. 2007.

[8] H. Kou, P. Johnson, and H. Erdogmus. Operational definition and automated inference of test-driven development with Zorro. *Automated Software Engineering*, 17(1):57–85, 2010.

[9] M. Maehr. Culture and achievement motivation. *American Psychologist*, 1974(29):887–896, 1974.

[10] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti, and G. Succi. A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team. In B. Meyer, J. Nawrocki, and B. Walter, editors, *Balancing Agility and Formalism in Software Engineering*, volume 5082 of *Lecture Notes in Computer Science*, pages 252–266. Springer Berlin Heidelberg, 2008.

[11] R. Moser, A. Sillitti, P. Abrahamsson, and G. Succi. Does Refactoring Improve Reusability? In M. Morisio, editor, *Reuse of Off-the-Shelf Components*, volume 4039 of *Lecture Notes in Computer Science*, pages 287–297. Springer Berlin Heidelberg, 2006.

[12] E. Murphy-Hill, R. Jiresal, and G. C. Murphy. Improving Software Developers' Fluency by Recommending Development Environment Commands. In *Foundations of Software Engineering*, Nov. 2012.

[13] E. Murphy-Hill, C. Parnin, and A. P. Black. How We Refactor, and How We Know It. *IEEE Transactions on Software Engineering*, 38:5–18, Jan. 2012.

[14] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[15] M. P. Robillard, W. Coelho, and G. C. Murphy. How effective developers investigate source code: an exploratory study. *IEEE Trans. Softw. Eng.*, 30:889–903, Dec. 2004.

[16] Saatchi and Saatchi S. Gameification for business brands and loyalty, June 2011.

[17] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz. Sando: an extensible local code search framework. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, New York, NY, USA, 2012. ACM.

[18] L. Singer. *Improving the Adoption of Software Engineering Practices Through Persuasive Interventions*. PhD thesis, Gottfried Wilhelm Leibniz UniversitÃd't Hannover, 2013.

[19] L. Singer and K. Schneider. It was a bit of a race: Gamification of version control. In *2012 Second International Workshop on Games and Software Engineering: Realizing User Engagement with Game Engineering Techniques (GAS)*, pages 5–8. IEEE, June 2012.

[20] W. Snipes, V. Augustine, A. R. Nair, and E. M. Hill. Towards recognizing and rewarding efficient developer work patterns. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1277–1280, Piscataway, NJ, USA, 2013. IEEE Press.