Application Monitor: Putting games into crowdsourced testing

Vivek Venkatachalam (vivekven@microsoft.com)
Marcelo Nery dos Santos (marsant@microsoft.com)
Harry Emil (harryem@microsoft.com)

Abstract

Software test teams around the world are grappling with the problem of testing increasingly complex software with smaller budgets and tighter deadlines. In this tough environment, crowdsourced testing can (and does) play a critical role in the overall test mission of delivering a quality product to the customer.

At Microsoft, we firmly believe in using internal crowd sourced[1] testing to help us reach high levels of test and scenario coverage. To achieve this goal, we use the dogfood program where employees volunteer to use pre-release versions of our products and give us their feedback on a regular basis. However, for a volunteer based crowdsourced testing effort to be really effective, one needs the ability to direct the crowd to exercise certain scenarios more than others and the ability to adjust this mix on demand. What if one could devise a mechanism that provides the right incentives for the crowd to adopt the desired behaviors ime?

This paper describes how we conceived of, designed and implemented Application Monitor, a tool that runs on a user's machine and allows us to detect usage patterns of Microsoft Lync[2] (the software product that this team of authors worked on) in near real-time. The paper then describes a simple game we incorporated into the tool with the goal of making it fun for the crowd. The game also provided us with the ability to direct their efforts to test high-risk features, by appropriately changing incentives.

One learning point was that even crowd behaviors that attempted to game the system served the ultimate purpose, which was to increase testing of specific scenarios. The paper will discuss this and other takeaways as well as point out key issues that other teams wishing to start similar efforts should consider.

Biography

Vivek Venkatachalam is a software test lead on the Microsoft Lync team. He joined Microsoft in 2003 and worked on the Messenger Server test team before moving to the Lync client team in 2007. He is passionate about working on innovative techniques to tackle software testing problems

Marcelo Nery dos Santos is a software engineer on the Microsoft Lync test team. As a remote employee for 15 months, he was an active dogfood user of the product himself. He has also worked on tools to help test performance for the Lync product. His main interests are working on innovative tools and approaches for enabling more efficient testing.

Harry Emil has worked at Microsoft on the Windows and Office testing teams since 1989. His research focuses on the wisdom of crowds, workplace productivity, and real-time multilingual communications.

[1] Crowd sourcing is the process of harnessing the wisdom of crowds to achieve tasks.
[2] Microsoft Lync is an enterprise solution for communications, delivering on Instant Messaging, Voice, Conferencing and more. For more information, please visit: http://lync.microsoft.com

1. Introduction

Wikipedia defines crowdsourcing as “the act of outsourcing tasks, traditionally performed by an employee or contractor, to an undefined, large group of people or community (a "crowd"), through an open call.” (1) It allows organizations (both for-profit and non-profit) to tap into the collective intelligence and labor of a set of people external to the organizations, to accomplish tasks much more efficiently than if they had to be performed in house. As a matter of fact, Wikipedia itself is a shining example of the power of crowdsourcing. Crowd sourced testing, which is simply the application of the crowd sourcing concept to the domain of software testing, is a useful technique to consider to solve a common problem that all software test teams face; i.e. the problem of verifying the quality of their software under a multitude of different environments. Crowd sourced testing implies that at least some portion of the testing is performed via a loosely co-ordinated approach across a crowd of diverse testers that are not officially part of the test team. This allows the test team to focus their limited resources on verifying software quality under a very well defined and constrained set of conditions (that reflect the common case) while still ensuring test coverage across the broader set of hardware and software configurations that exists out in the real world relatively cheaply and efficiently by leveraging the power of the crowd.

Productivity games are another phenomenon becoming increasingly popular in the business world. The central idea is to introduce creative and fun game-like elements into an employee’s regular workflow, to make work seem more like playing a game than just completing a set of routine tasks. The aim of this approach is to increase employee motivation and morale and thereby indirectly increasing their productivity. A great example of a productivity game is the Language Quality Game. (2)

In this paper, we describe our effort at combining these two key ideas in a series of experiments to increase the test coverage of Microsoft Lync during the last few months before release-to-manufacturing. Microsoft Lync is an enterprise communication and collaboration system that offers users a rich set of features (instant messaging, user presence, voice/video calls and conferencing, content sharing and many others). It comprises of a set of server components (collectively known as the Lync server topology) and a set of clients that run on multiple platforms (Windows, Mac, Web client amongst others) and connect to and communicate with the server components to provide this rich functionality. The authors were on the team responsible for verifying the performance aspects of the flagship Lync client, the client running on the Windows platform ; all future references to Lync in this paper imply this specific Lync client.

Since the underlying technologies were based on work we had done for automating our performance test system, we start the paper by briefly explaining the performance test system, to help the reader get a basic understanding of these technologies. In the subsequent section, we describe how this system we built to collect performance data in a lab environment was modified and extended to allow us to also automatically collect performance data from our dogfood users, i.e. our crowd of testers. Next, we describe how we added games into the mix and expanded the scope of the project from a mechanism that collected performance data passively from users to a mechanism that actively tracked scenarios that users were running and provided them with incentives to run a desired set of scenarios. We then conclude the paper by presenting some key takeaways from this project and ideas for improvements.

2. Background

Two of the authors (Vivek and Marcelo) had been primarily involved in designing and implementing a performance test system for the Lync client. We will now give a brief overview of the process to set the stage and provide readers with the necessary background to follow the paper more easily:

a) First, performance scenarios were identified along with defining elapsed time goals for each of these scenarios. Examples of scenarios identified are SignIn, SignOut, and MakeAudioCall etc.

b) Next, for each scenario, for example, SignIn, instrumentation was added to Lync source code by adding a SigninStart event in the piece of code which handled the click of the SignIn button and by adding a SigninStop event in the piece of code that executed when the user had been successfully signed in. The instrumentation was done by using APIs provided by the Event Tracing for Windows (ETW) toolkit[3].

c) The SignIn scenario was then automated using UI automation. To accomplish this, we wrote some test code that, when run, would simulate a real user signing in by entering her credentials and then clicking the SignIn button.

d) The Windows ETW system would be used to capture the SigninStart and SigninStop events in a binary EventTraceLog (ETL)[4] file. This file contains a log of events (with detailed description) along with timestamps that indicated when the events occurred.

e) An analysis process would use the Windows ETW API to parse the log, read individual events in the log, match start and stop events for a given scenario appropriately and compute the elapsed time for the scenario by computing the difference in the time stamps between these events. The average of computed results from a number of the reasons was used as the estimate for what response time a user might reasonably perceive when exercising that scenario.

3. Introducing crowdsourced testing to the process

Almost all products that are built at Microsoft go through an internal crowd sourced testing process called dogfooding (2), which is basically the idea that employees eat their own dogfood before it is released to the external world. This has long been a key part of the test strategy at Microsoft since it allows product teams to get broad test coverage for real user scenarios against a multitude of hardware and software configurations. Participants in the dogfood program for a product are typically comprised of volunteers from across the company that are interested in trying out the cutting-edge version of the product and want to help the product team improve the quality of that product. The Microsoft Lync team too had its own set of dogfood users that would use the product in their regular work and report issues back to the product team for further investigation. While this worked well for issues related to the functionality of the product, we really did not have a good way to get objective performance data back from these users.

As described in the previous section, we already had a working automated performance test system to track the performance for the Lync client in a controlled lab setting. While we had the ability to collect performance data in the lab, we didn't really know how well the performance measured in the lab would translate to the actual performance of the product observed by our internal crowd of testers i.e. the community of Lync dog food users. After putting some thought into ways to collect performance data from dogfood users, we realized that we could modify and extend key pieces in the performance test system and use these to build a tool that could get real-time elapsed data from a dogfood user's machine. The two pieces different from a lab setting are indicated below:

a) Instead of UI automation driving a scenario, it would be a real dogfood user exercising the Lync UI.

[3] A system that provides application programmers the ability to start and stop event tracing sessions, instrument an application to provide trace events, and consume trace events. http://msdn.microsoft.com/en-us/library/bb968803(v=VS.85).aspx
[4] The binary log file generated by an ETW tracing session.

b) Instead of the resultant ETW events being written to a file to be processed after the fact, these events would be processed in real time by another executable program that had registered with the ETW system to in order to receive real time ETW events from the Lync process.

To achieve this, we built a tool that could run alongside Lync, consume ETW events from Lync in real time, detect when a pair of matching start and stop events had been received and output the computed elapsed time in its UI. In addition, the tool reported the computed time back to a central service that we maintained. Since the tool was designed to be easily modified to consume and process events from any Windows application, not just Lync, the tool was given a generic name: Application Monitor. The tool was designed to run unobtrusively in the Windows Notification Area[5] and no additional intervention was required of the user once they had installed and running. It also had an auto-update feature built in so we could push out new features and bug fixes without requiring any additional work from the user other than the initial step of installing it.

Application Monitor was made as a separate tool because we wanted it to be completely isolated from the actual product code. That way, the product code would have instrumentation added only for the performance markers itself, but the consumption and processing of the fired events would be made by a separate process. By using the Windows ETW infrastructure, this task would be achieved by placing minimal impact on the product actual performance and we would also leverage the fact that when no listeners are subscribed to the events, the ETW infrastructure will actually ignore those events with no performance impact.

Once Application Monitor was running, it automatically collected performance data for the scenarios that users were executing assuming the Lync source code for those scenarios had been instrumented. Coupled with a mechanism to send this data back to a central location, we now had an effective way to track elapsed times for scenarios not just in a controlled lab environment but also on real user's machines. The only thing we needed was to convince dogfood users to install Application Monitor separately from the Lync client and leave it running. We found that this simple requirement to install a separate application was an obstacle to get widespread deployment across the dogfood user population, a topic we address in some more detail in the concluding section.

4. Adding games to the mix

The fact that we had this ability to track elapsed times for scenarios meant we also now had even more basic and potentially more valuable ability: the ability to track Lync scenarios being executed on dogfood user's machines. In effect we now had a lightweight, near-real-time telemetry system that let us know which scenarios were being executed the most, which were being executed the least and which had never been executed (based on the sample of users that chose to install Application Monitor). During our internal evangelization effort (to get people running Application Monitor and provide us with real performance data), it caught the attention of a set of people on our team (our co-author Harry was among this set) that had spent considerable time in designing, developing and deploying productivity games internally at Microsoft. After some discussions, all of us realized that the telemetry functionality in Application Monitor could be used to power games related to dogfooding Lync. This was when we decided to add game elements into Application Monitor to a) incentivize users to participate more extensively in the dogfood program and b) add a bit of fun to the dogfood process.

Besides greater dogfood adoption, the game elements also provided incentives for users to install Application Monitor on their machines since the game required Application Monitor to be running, thereby increasing the quantity of performance data that we got from user's machine. Based on this set of discussions and some creative design and coding work, we implemented and deployed a set of

[5] This is more popularly known as the System Tray or SysTray, the lower right section of the Windows task bar.

experimental games based on Application Monitor functionality. A brief summary of these games are presented in the subsequent subsections.

4.1. Easter Eggs

Easter egg games were engaging images that popped up on the dogfood user's screen when they completed a certain set of scenarios in Lync. The scenarios had to be exercised in a specific order and would indicate that the dogfood user had unlocked an achievement. These encouraged ad-hoc exploration of Lync features (and hence more test coverage for the product) as players tried to uncover these Easter eggs and unlock achievements. To make things even more interesting, creative folks from the team came up with what were affectionately known as "HaiClues", clues to discover the steps needed to unlock achievements in the form of Haiku[6].

Some examples of HaiClues can be seen on Table 1 – HaiClues:

Explicit steps needed to unlock an achievement	**HaiClue version**
1. Switch from sharing a PowerPoint deck to sharing a whiteboard 2. Switch from sharing a whiteboard to sharing a PowerPoint deck	We share, we succeed Slides, whiteboard then double back Accomplishment; joy
1. Search for a colleague 2. Add them as a contact in an existing contact group. 3. Expand that contact group	Find a Coworker Groups hold Contacts for Later Growing groups expand

Table 1 – HaiClues

Figure 1 shows an example of an easter egg that is simply a static html page with some text and images.

[6] Haiku is a short form of Japanese poetry consisting of 3 lines of 5, 7 and 5 syllables.

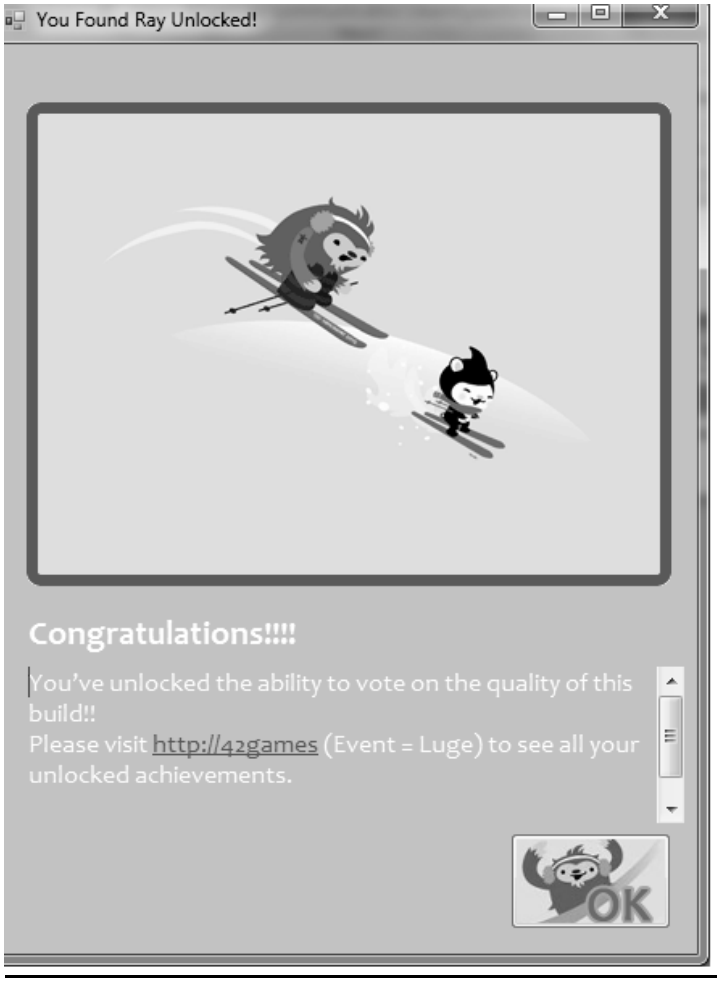

Figure 1 - Example of a static Easter egg

Figure 2 shows an example of an interactive Easter egg in the form of a contact card built on Lync Silverlight controls. In this example, the Easter egg brings up the contact card of a researcher at Microsoft with whom the user can directly communicate using different modalities (click the envelope to send mail, click the text bubble to send IM, click the phone to make a call etc).

Figure 2- Example of a Silverlight based interactive Easter egg

4.2. Olympics themed game

The next step was to make the game more realistic by launching an Olympics themed dogfood game to make it fun for dogfood users to participate as well as use leaderboards as incentive to drive user behavior. Participants self-selected into teams of their choosing and participated in various “events” with the aim being to collect as many achievements as possible in each event, for themselves as well as for their teams. Figure 3 is a screenshot that describes the various ”events” on the Olympic Games, the first two of which used Application Monitor to detect that users had unlocked achievements by completing specific scenarios.

Figure 3- Overview of the games

As an example, the aim of the "Curling" event was simply to try various instrumented scenarios in Lync with Application Monitor running in the background. For each scenario that a user completed and was recognized by Application Monitor, they would get one step closer towards unlocking the corresponding achievement. The number of times the scenario needed to be run to unlock these achievements was set differently for each scenario. ("LaunchCommunicator"[7] was worth 120 points while "ResumeFromHibernate" was worth 5400 points) to motivate participants to run more of the scenarios that mattered to the test team. Since these weights were set on the back-end, we could easily change the weights as needed to get optimal level of scenario coverage by the dogfood users. For example, if we had recently made a big code check-in in the SignIn feature, we could get the crowd of testers to try out this feature more heavily by simply increasing the number of points gained by unlocking this achievement. Some screenshots from various tabs of the leaderboard for the "Curling" event are shown below as an illustrative example (names have been removed to protect privacy of game participants).

[7] Microsoft Office Communicator was the former name of Microsoft Lync

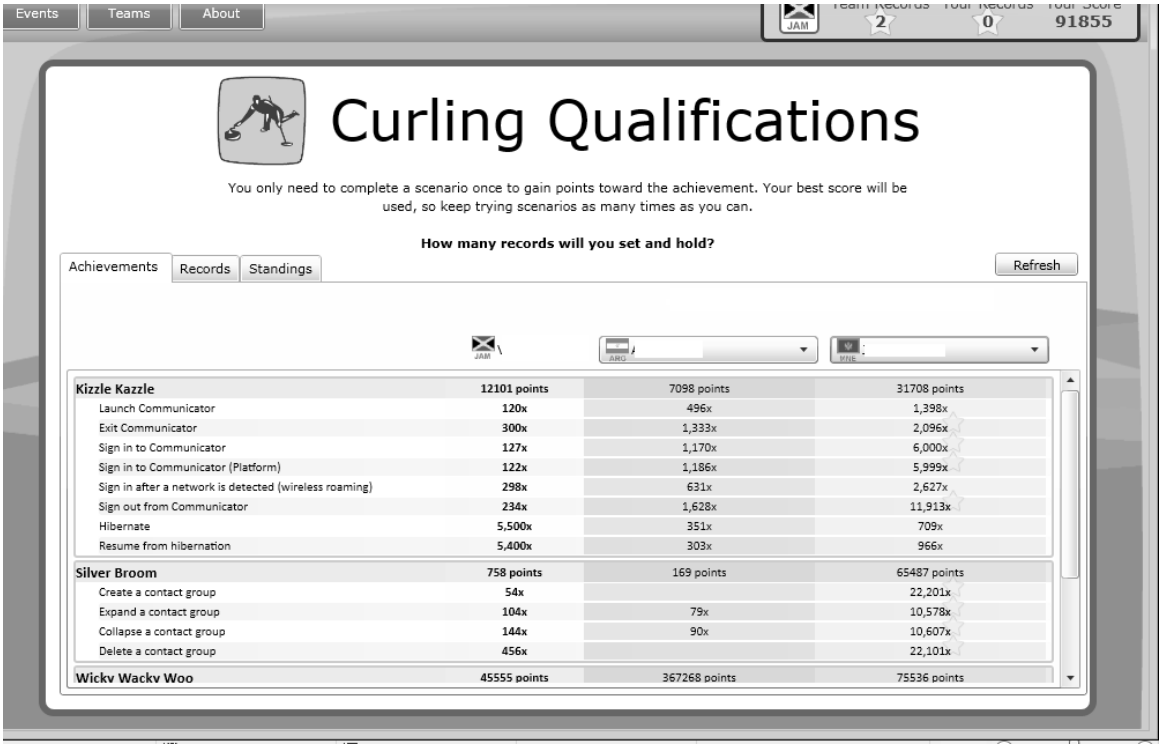

Figure 4- Screenshot of the achievements tab in the leaderboard

We also made it possible for a user to see the leaderboard for a specific scenario within the “Curling Qualifications” event. Figure 5 shows the leaderboard for the “Launch Communicator” scenario. This allowed participants to easily see what the highest score was and how they were doing on each scenario compared to their peers.

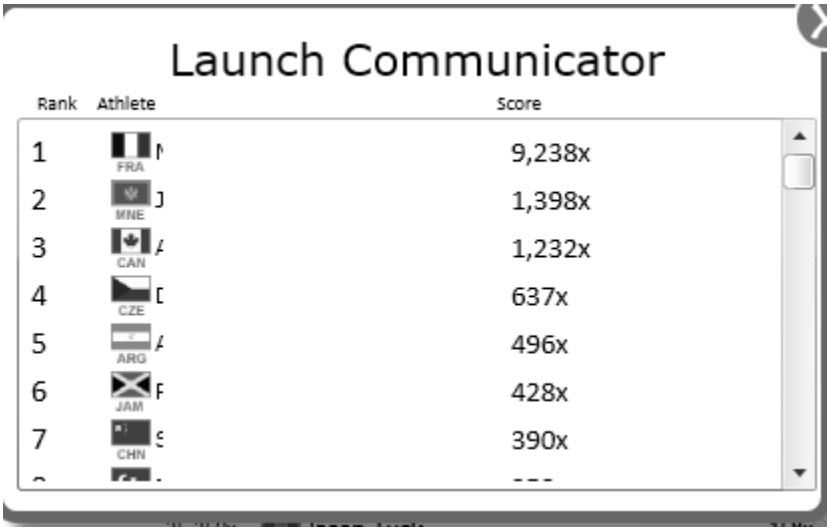

Figure 5- Standings for the "Launch Communicator" scenario

4.3. The "Spell Communicator" game

The "Spell Communicator Game" was the next phase of our experiment. It was advertised to dogfood users with instructions stating: "*Each of the letters in 'Communicator' is associated with a key scenario for Office Communicator and will light up when you have successfully completed it. After spelling Communicator, completing a bonus scenario will cause the Communicator "fishhook' logo to light up, and fame and fortune will result. OK, so maybe not fortune, but certainly fame will be yours."*

Once we had all the games in place and Application Monitor was known on the broader team, we encouraged developers and testers to add instrumentation for their features, so they could feature as letters in the Spell Communicator Game. This was a great way to improve the quality and quantity of instrumentation in Lync source code.

Figure 6 shows a couple of key UI elements of the game. Letters in blue (the letter 'o' in the figure) indicate scenarios that have been completed while hovering the mouse on a light gray letter causes it to darken (the letter 'n' in the figure) and display a hint on how that achievement can be unlocked. In a sense, this UI allowed a user to easily see his "score" and figure out how what other steps were needed to complete the game.

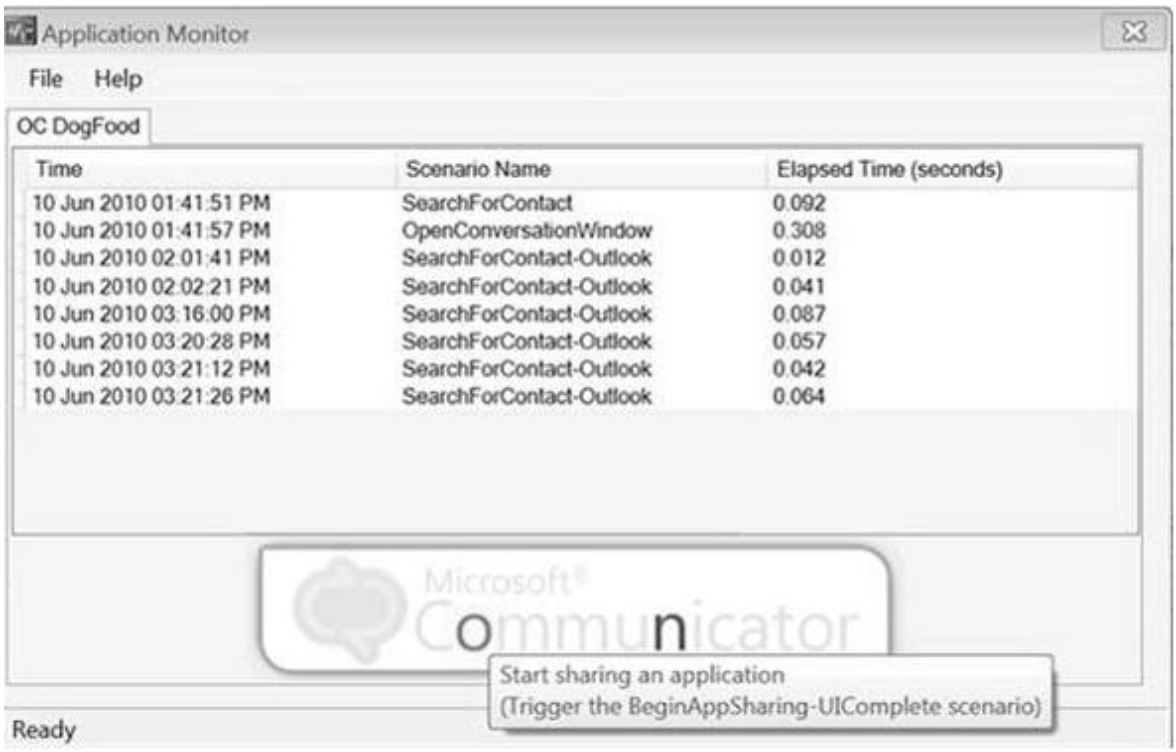

Figure 6- Screenshot of Application Monitor with the Spell Communicator game

The way Application Monitor was designed, it was fairly simple to have the scenario completion information sent to a WCF service which stored and tracked scenario completion information for each participating user in a database. The rules on whether or not an achievement should be unlocked were stored on the server and the server's response based on these rules would indicate whether or not to light up an achievement. The achievement definitions and images were defined at the server as well and Application Monitor simply updated the appropriate image on the user's machine. In a sense, this feature introduced a player vs.self game element into Application Monitor. Since these achievements were only unlocked per build, the user would have to start over again every time they upgraded their build. This was

a useful feature because it helped us get this basic level of coverage on each new build that users installed, essentially helping us smoke-test each build at low cost to the test team. In essence, we were getting the users to run a very specific set of tests for each build we released while providing them with some interesting and fun experience in return.

Making the achievements view more dynamic would only involve changes on that service to reply a different set of achievements data for different people, and making this configurable for each team/person would be implemented by creating a front end to change the DB data. Application Monitor code did not have to be modified to support a new set of achievements images or data.

5. Conclusion

Based on the amount of data we collected it was clear that we were successful in a) providing strong motivation for a certain section of dogfood users to increase their participation levels and b) in attracting a new set of dogfood users to try out Microsoft Lync. At its peak, we had about 200 users actively participating in the games and we got coverage on approximately 80 instrumented scenarios. We got a lot of useful information about which scenarios that were being executed most frequently and which were being executed the least frequently. The great thing about publishing this data was that testers and developers of Lync features that previously had no instrumentation for their scenarios were motivated to add instrumentation for their features as well. This helped increase coverage in terms of number of scenarios that could be tracked. In some cases, we also saw that the really competitive players even attempted to game the system by automating game scenarios so they could quickly get to the top of the leaderboard. While this behavior was not really conducive to the spirit of the game, it was still a fantastic outcome for us because this ensured very good coverage for those scenarios for almost zero additional cost to the test team.

There are also some issues we found along the way. The biggest issue by far was that the original instrumentation in Lync was added to help identify scenarios that were run in a deterministic fashion and in a controlled set of circumstances in the performance test lab. However, real word usage by dogfood users is very different and this exposed some holes in our product instrumentation for scenarios that did not terminate normally. For example, if a dogfood user cancelled the SignIn process or SignIn failed for the dogfood user, this was not detected correctly by Application Monitor and sometimes resulted in false positives i.e. Application Monitor would report completion of a scenario that the user really had not completed. Needless to say, this is a big problem as it strikes at the very heart of the system. The key takeaway from this is that the instrumentation needs to be added very carefully so as to capture all entry/exit paths for a given scenario. In addition, the logic in Application Monitor needs to be more robust to unexpected event ordering. The ideal implementation would use a carefully designed finite state machine to drive this logic, so as to always make the correct inference about which scenario was executed.

As mentioned earlier, another issue was that the need for users to perform a separate install posed a biggest hurdle to getting a more widespread adoption of the tool. If we were to do this again from the very beginning, we would definitely explore options to increase the incentives for dogfood users to install this tool by providing other useful functionality (for example, allow an easy way to report their feedback on Lync, allow an easy way for users to find out about known issues on the build of Lync they are currently running etc.) in the tool making it more attractive for users to consider installing and running it on their machine.

Acknowledgements

This project would not have been possible without key contributions and support from the following people:

- Mike Jackson – Software development engineer on the Lync team instrumental in the overall initial design of Application Monitor.
- Ross Smith – Director on the Lync client test and key supporter and evangelizer of productivity games. On his blog devoted to productivity games, he defines them as “a sub-category of serious games designed to improve the productivity and morale of people doing "regular work". (2)
- Joshua Williams – Senior tester on the Lync team and game master[8] for the various games documented in this paper.

References

1. Crowdsourcing. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Crowdsourcing.

2. *SCORE ONE FOR QUALITY! USING GAMES TO IMPROVE PRODUCT QUALITY.* **Williams, Joshua, et al., et al.** Portland : Pacific Northwest Software Quality Conference, 2009.

3. Eating your own dogfood. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Dogfooding.

4. **Smith, Ross.** *Productivity Games - Ross Smith .* [Online] http://productivitygames.blogspot.com/.

[8] The person who acts as the referee/organizer for conducting a game is known as a game master.