Testing in Production: 
It’s not just for services anymore!

Adil A. Gheewala – adilg@microsoft.com
Sam Bedekar – sbedekar@microsoft.com
Weifeng Yao – weiyao@microsoft.com

Abstract

The test team in an organization often has a large set of new and regression tests to execute to ensure the quality of the features. The ever growing test matrix is another strain on the test organization as it needs to support multiple legacy clients, release after release.

Testing in production where real users test real scenarios is a great way to harness the power of the crowd, get a more end user centric test coverage, and reduce some of the tests the actual tester needs to perform. End users would install the application and use it in their normal mode. The TIP system collects the relevant user interactions and allows the owner to make decisions based on the data – feature test coverage and/or feedback on the feature.

The Lync Telemetry v-team has developed a tool called Scenario Compass which listens for events from Lync and any other Office 2013 applications to distinctively identify the set of actions/features a user exercises while using them. This allows the feature area owner to have data behind his crowd sourced tests. Scenario Compass also collects machine and application specific data which can be used to check off configurations in the test matrix. This allows the test team to focus their efforts on configurations with low coverages. Scenario Compass also flags failing cases. If a certain anticipated event does not occur in a given amount of time, that scenario is flagged as a failure. Using the failure analysis component of Scenario Compass, one could also do performance tests in the real world.

Scenario Compass has been used in the Lync client team to optimize the test passes by identifying areas covered by internal Microsoft users and reducing effort in them and identifying holes in coverage/configurations and focusing more in them. We also have built games into the tool to increase adoption and more importantly influence users to execute a certain set of features.

This paper, describes some of the challenges we faced in our testing, how we overcame them, the solution we built and how others can leverage those in their environment.

Biography

Adil Gheewala is a Senior Test Lead at Microsoft Corporation, who joined Microsoft in 2002 and worked in Windows for 4 years before joining the Lync client team. Adil is passionate about optimizing test methodologies & processes, and driving complex features to high quality.

Sam Bedekar is a Senior Test Manager at Microsoft Corporation in the Lync team. He has worked in UC space for ten years. Sam has a passion for test and the impact it can have on product quality and the industry.

Weifeng Yao is a Software Development Engineer in Test at Microsoft Corporation. He joined Microsoft in 2008, and worked in Office Communications Group China before moving to Lync client team in Redmond, WA in 2009. He is passionate about exploring different test methodologies and push high quality product, and solving hard technical problems.
1. Introduction

Coined by Jeff Howe and Mark Robinson in the June 2006 issue of *Wired* magazine (Howe, 2006f), the term *crowdsourcing* describes a new web-based business model that harnesses the creative solutions of a distributed network of individuals through what amounts to an open call for proposals. Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call.

Testing of a product usually involves a large set of configurations on which each feature needs to be exercised. With every release of the product, legacy clients and features need support while new features get added. This further increases the cost an organization needs to pay for testing and stabilizing the quality of the product.

In an organization employees often use the product in development themselves (a practice called “dogfood”) for a good amount of time trying to iron out bugs that need fixing. The channel to collect the list of issues is usually an email distribution group where the internal Microsoft users post their questions on the failing scenario. They rarely send out a periodic list of scenarios that succeeded for them.

In this paper, we describe how the organization can use the Scenario Compass tool to deploy, gather and analyze usage data, both success and failure scenarios, from the crowd without needing them to change any of their execution, how the test organization can focus their efforts based on execution of scenarios by the crowd. The premise is that if a large set of users are hitting a scenario, at least a handful of them would report any visual glitches which tools like Scenario Compass cannot identify. We also describe how the data collected by Scenario Compass is not limited only to the test team, and can be used by the User Experience team to identify the utility of features; the performance team in identifying real world performance numbers, the developers in predicting the amount of unit test needed on a feature they plan to change based on coverage it has from the crowd.

We begin our paper by explaining the problem space and the challenges faced due to a large test matrix. Next, we cover details of Scenario Compass, followed by the architectural design and results generated from the data collected from it. We then conclude the paper by presenting some key takeaways from this project and efforts that we are making for the next version of the tool.
2. What Is the Problem Space

The variations of tests to execute are huge and continue to grow from version to version. Each new release adds more complexity to the matrix. Just some of the variables in the current matrix are:

- The different OSs and architecture of those OSs
- The different architectures of the client product
- The install method of the product
- The SKU of the product
- The current version of the client to interop with, the previous version of the client, and the version before that.
- The current version of the client to be able to work with same and previous version of the server
- Interop with other legacy clients
- Different modes in which the client can be run.
- For the Lync client it could be desktop mode, remote desktop with Virtual Desktop Infrastructure mode on/off, Pairing with deskphone on/off
- Different versions of supported browsers
- DPI and screen resolutions.
- Touch & non-touch
- Topologies (OnPremise, Online)

Given the extensive test matrix, it's not a good ROI for the test team to execute all their tests on various combinations of the configurations. In such situations the test team executes the tests on some of the important configurations and can then rely on the masses to execute scenarios on several different configuration variations.
3. **Why Create Scenario Compass**

In a services environment, engineers can build logic into the product to direct and measure usage for the purposes of Testing in Production as well as Active and Passive monitoring. Telemetry and A/B testing is a key component in large scale production deployments. The architecture allows the administrators and engineering teams to measure, change, and react in real time to problems or respond to patterns observed in user choice.

A traditional desktop product often lacks the telemetry or mechanism to store, analyze, and react to real-time telemetry from the user. The desktop application may not have a server backend or the communications protocol may only contain business logic specific to the transaction at hand. Metadata may not be available or may not be of the fidelity desired for TiP (Testing in Production). In the examples above, install methods, architectures, browsers, DPI, etc. are client specific configurations which may not be in the protocol between the client & server.

To fill this need, we built the Scenario Compass. Scenario Compass will "observe" the target product through a combination of Out of Process events, ETW markers, or proprietary APIs/performance markers. Actions can be monitored, collected, and analyzed for usage. Scenario Compass will gather the configuration data from the user's machine. Scenario Compass will then upload the data to a central store where it can be aggregated & processed. The scenarios are associated with the configuration data for a complete picture of "what" happened and "how" it happened. Scenario Compass also provides a mechanism to send a response back to the client application to directly engage the user if desired.
4. What is Scenario Compass

Scenario Compass is a tool created by the Lync Client team and can be visualized as two separate applications:

4.1 Scenario Creator:

- The test owner needs to define which scenario is of interest by defining a set of events that get fired when that scenario is executed by someone.
- Using the Scenario Creator tool, the feature area owner defines the ordered sequence of events that make up a scenario.
- Once the scenario is defined, it is pushed up to the database and down to the Scenario Compass application.
4.2 Scenario Compass Application:

- Scenario Compass is a lightweight application that listens to product instrumentation.
- Scenario Compass application is installed on the internal Microsoft users’ machine.
- It attaches out of process to Lync and Office applications and listens for events fired by them.
- Based on the scenarios defined in the database, Scenario Compass application maintains an internal state machine to verify if any of those scenarios are hit and uploads the occurrence to the database.
- Scenario Compass also uploads various machine configuration information like OS, architecture, install type, etc.

Scenario Compass Application snapshot:

![Scenario Compass Application snapshot](image)
5. Design

![Diagram of Scenario Compass application architecture]

Architecture Overview

Scenario Compass application is a Client-Server system. It has 3 core pieces including Scenario Compass client, the web service and storing Database.

As a high level overview, Scenario Compass client listens to the markers (data points in source code) from different marker sources, recognizes the markers as scenarios using definitions retrieved from the web server, and submits the hit data to the web service. The web service will process the data and store it in a database. Besides scenario hit data, Scenario Compass client also recognizes scenario failures and target application resource usages; for example, CPU and memory usage when target application exceeds a threshold.

Scenario Compass client is designed to be extensible to handle markers from different sources. We use the term Marker Provider for this role which is responsible for collecting raw marker data from marker sources and feed the scenario/failure recognition component.
Currently we have 2 types of marker providers - Office marker provider and Lync event provider. Office marker is a data structure instrumented in the office applications mainly for performance measurement purposes. When office applications perform different actions, corresponding marker data will be written to a shared memory space so Scenario Compass can read it. Lync event marker doesn’t require additional instrumentation in the product. Instead, it utilizes the rich platform features and testability designs built in to the Lync application. More specifically, Lync client will start a COM interop layer when it starts. The interop layer allows out of process consumer applications to access (a.k.a interop) the properties and subscribe to the events. Scenario Compass is one of such out of process consumers. When Lync event marker provider sees Lync client starts, it will subscribe to the events and receive notifications from Lync client through this interop layer. When an event is received by the provider, it will be converted into marker that recognition component can understand, and recognize them as scenario hits or failures, which is the same process as Office markers.

The Scenario and failure recognition component is a finite state machine implementation. During Scenario Compass client startup, the state machines will be initialized from the scenario definitions created by Scenario Creator tool. At runtime, each marker will be tested by the state machines to determine if a marker will push the state machine to the next state, or a failure state. When a state machine reaches its end state, a scenario hit is generated and submitted to the web service.

For Scenario Compass client to work effectively, scenarios should be created prior to the data collection. This is done by Scenario Creator. It is a data centric UI application that allows feature owners to enter scenario definitions he or she want Scenario Compass client to recognize. A scenario can be considered as sequence of expected actions. In Scenario Compass, a scenario consists of a sequence of tasks, and a task consists of a list of markers. There are 2 ways to create a scenario, build bottom up or record and play. Using the first method, feature owners will search and create markers, then create tasks using the marks, finally create scenarios consist of the created tasks. Using the second method, record and play, feature owners will run Scenario Compass in a calibration mode to run the scenarios and collect marker sequence for that scenario using Scenario Creator, after the markers are collected, feature owners can select the markers they’re interested in and save as tasks and scenarios. The second way makes the scenario creation much simpler.
6. Case Study – Lync Deployment:

A key part of our test portfolio is known as our “Dogfood program.” The term comes from dog food manufacturers having to taste their own dog food to deem whether it was desirable for consumption. In the context of Lync, we find that a large scale internal deployment of the preproduction application helps find issues that may not be found in local testing.

The dogfood program relies on a key crowdsource tenet – that at scale, users will complain & report if they encounter issues. The top issues will then reveal themselves and can be corrected before shipping the product. Conversely, one of the challenges of a Dogfood program is in detecting “what went right.” It may be hard to distinguish what was working properly versus a feature that may not be used heavily in the deployment. Similarly, even if a feature is used frequently, it is hard to determine the spectrum of configurations covered in the usage of that feature.

For the Lync Dogfood deployment, we deployed Scenario Compass alongside Lync to fill in this information. This required each test owner to define the scenarios in terms of events that the tester would want to measure.

An area owner can decide to view results based on how many dogfooders hit his/her feature scenarios on a particular build or on a particular topology, for example how many users used the product in high DPI, if users used the feature on an x86, x64 architecture, and so on. Scenario Compass can provide all the data needed to quantitatively answer these questions.

The feature area owner can use our website to pivot the data off several variables. This chart shows the scenario coverage of Application Sharing feature are for the last 3 days on the last 3 Beta2 builds of Lync on specific topologies.

Scenario coverage by build.

![Recording for Beta 2 builds (last 2 weeks of Beta2)](image-url)
Scenario failure information:
An area owner can decide that his scenario should complete in a given amount of time when defining the scenario in Scenario Creator. The time taken for a particular scenario can be calculated from the data uploaded by Scenario Compass and based on that a scenario can be marked as pass or fail. This timing information can also be used by the performance team to identify the how long their scenarios take to execute on the user’s machine and compare the data (if any) with that taken in the lab.

Using target percentages per configuration, one can create a heat map as such of the test matrix (variations of tests), so the team knows where to focus efforts on. The heat map below has been created using data collected for Beta2 and percentage of overall usage set for each cell. E.g. this heat map of OS usage, conveys that Office application usage on Win7 has high but usage on other OSs like Windows Server 2008 R2 and Windows 8 was low.
<table>
<thead>
<tr>
<th>Scenario Type</th>
<th>OS</th>
<th>Install Type</th>
<th>OS architecture</th>
<th>Lync Architecture</th>
<th>Network connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV Conferencing</td>
<td>Windows 7</td>
<td>MSI</td>
<td>x86</td>
<td>x86</td>
<td>Wired</td>
</tr>
<tr>
<td>PPC</td>
<td>Windows Server</td>
<td>C2R</td>
<td>x64</td>
<td>x64</td>
<td>Wireless</td>
</tr>
<tr>
<td>PPC</td>
<td>2008 R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPC</td>
<td>Windows 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPC</td>
<td>Consumer Preview</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio</td>
<td>Windows Embedded</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio</td>
<td>Standard, Service Pack</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mso</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataCollab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Based on data we can build performance measurement charts as below.

From this chart we can identify that there are several users that are way over the original performance target planned by the team. This helps in readjusting the expectation and picking a more realistic target and to investigate the users that are over the mark.
7. Interaction:

7.1 Feedback:

We built a channel in Scenario Compass to be able to reach the dogfooder. This is currently used to notify them that they have just completed a special scenario and we’d like their feedback on their experience of the scenario.

a. End of call rating and tying QoE data
We can pick any scenario defined in Scenario Creator and tag it as a ‘game’ scenario. We can also associate an action Scenario Compass would take when that game scenario is hit.
We have tagged end of a call as a game scenario. When the user hits this scenario, Scenario Compass, pops up a toast near the systray like this

![Toast Example]

When the user clicks on the toast we launch a survey that the user can take to describe his experience.
Once the survey is completed the response is recorded. Scenario Compass uses a context tag to correlate the survey response and the scenario hit information that it uploads. The QoE data that Lync uploads automatically can also be correlated to the scenario hit information, tying all the configuration information, the QoE data and the survey response together.

b. Recording playback by dogfooders.
Recording is a feature in Lync. The true E2E test for this feature is to record a call, conference and then play it back to ensure the recording completed successfully.
We are in the works to have another game in Scenario Compass where we pop up a toast to users that complete a recording session and request them to play back the recording and give feedback on its quality.
7.2 Incentivizing Users Through Games

- With Scenario Compass running on several user’s machines we want to encourage people to execute certain scenarios or use certain features. We can show the scenario of the day on Scenario Compass’s main UI and give an indicator of it being completed by the user or not.

- Easter Eggs
  Our Test Director has hidden an Easter egg Lync scenario in Scenario Compass which when hit by a user would pop up a toast which would lead to an email as below. He also provided several clues to help folks focus in certain areas to hit this Easter egg. We found that these incentives helped increase Lync feature usage of sparsely used features.
8. **Key Takeaways and Next Steps**

This project has been enlightening on several levels. Architecting and building the tool for scale presented the majority of the technical challenges. Running the dogfood program with sufficient participation to collect the data was a separate set of challenges.

We found three main tenets that are key to success:

1. **Good scenario definitions is the key**
   Identifying which scenarios will provide the best ROI (in terms of reducing manual tests) is very important. The right scenarios help confirm that a particular configuration is covered and hence a subset of features can be skipped. These valuable results that help the feature team can only be achieved if the quality of the scenarios themselves is high. To make Scenario creation easier, we created "Scenario Spy" to help capture the correct events from Lync.

2. **Large set of users is important**
   Having a large set of dogfooders is important to get more use of your scenarios. You can get automatic triage of sub features based on its usage from dogfooders. For example, towards the end of a milestone if a bug fix needs to be made in an area that has very shallow coverage from a large set of users, then the feature team has the usage data of the feature which could help in determining whether the fix should be taken or not. Results from the usage can also provide a direction on which scenarios are more frequently used, so more feature weight can be added there. Knowing which scenarios are failing help the owners to get a better sense of product quality and where more testing is required. Having a large scale deployment is needed for the data to be statistically significant to make these decisions.

3. **Built for extensibility**
   Historically in every release we add additional backwards compatibility, additional support in the product, and more integration points. The test matrix continues to grow whether it’s configurations, combinations, etc.. With the appropriate investments in Scenario Compass scenarios and config capturing logic, the feature team (with roughly the same number of resources) can handle the additional test burden. In addition to measurements, the tool adds a new channel for owners to engage with the users to help cover the increasing number of scenarios. Using ‘Games’, a feature area owner now has a medium to get that feature more visibility. Performance data for features can now be collected from the users which are more indicative of what the customers will experience rather than lab run results. Easter egg games are a big incentive for people to try out different scenarios.

There are also some more subtle optimizations we learned over time in terms of the deployment.

a. **Less distraction for users**
   Users are very sensitive to their attention in the world of ever-increasing distraction. To minimize the “distraction footprint,” Scenario Compass runs in an extremely quiet mode. We scaled the UI down as the deployment scaled up. When the user base of Scenario Compass was small we were able to have a higher touch engagement. If that meant prompting the user to click the ‘ok’ button to repair then we would pop up a UI for that. With the ramp up of the number of users we had to shift the attention to not being in their face at all. Once we had statistically significant deployment numbers, we opted for loss of data as opposed to bothering the users to repair for getting higher rates of data.

b. **Having a bank of machines**
   Having a large set of dogfooders means a significant increase in traffic to the web service and database. We had to move to a model where the web service, database and the update/install location for the tool were all on different physical machines. A ‘Fail Safe’ shutdown switch was
added – this is a mechanism to automatically shutdown Scenario Compass in case we identified an issue after deployment. We implemented not one but two Fail Safe switches for Scenario Compass. When either of them are enabled Scenario Compass shuts down. One of the switches is a file on a file share and the other is an http link in case the file share is not accessible. We have used the Fail Safe switch a couple of times to make some fixes that came in as design change requests from management. This helped us achieve our “less distraction” goal for the end users.

c. Resilient start of Scenario Compass: We want Scenario Compass to be running any time the user is logged on to the machine. For this we created scheduled tasks to launch Scenario Compass on log on and at midnight each night. The reason for midnight SC was to start SC in case the user did not log off and back in and Scenario Compass was not already running on the machine (say Fail Safe Switch was now disabled).

d. Batching and throttling of upload information:
With increased users, and even after having separate machines for the web service and database, the traffic to these components becomes unmanageable due to limitation of network, number of requests the web service can handle, and/or the CPU of the machine. We had to update our design to include batching of the scenario hit information to the web service in intervals of every 5 minutes or when 15 scenarios are hit, whichever is earlier. We also had to include throttling of the access to the WS or update location. Earlier, each Scenario Compass would query the WS for a large amount of metadata. For example whether a new Scenario Compass build was available for auto-update, if the Fail Safe switch on, are new scenarios available, etc. This caused a flood of requests to the WS which could not handle the load. We accomplished throttling by randomizing the start time of each Scenario Compass when the user logged in or after midnight, upto a max of 5 minutes delay. This randomized the polling time for each Scenario Compass since they all were relative to Scenario Compass’s start time.

Some of the next steps, to increase the versatility of Scenario Compass that we are investigating are:
- Automatic uploading of logs when a failure scenario is hit. This will help in debugging the issue in addition to knowing the frequency of failure.
- With increased number of users, Scenario Compass itself should be A/B tested as changes to Scenario Compass, the games, or the scenarios are rolled out.
- Having the ability to configure which games a particular user could participate in. We can decide that we don’t want to distract senior executives with all the games the rest of the users are participating in until they are ready.
- Feature of the day. We are putting the final touches on being able to influence the users to try out certain scenarios.
9. Conclusion

We found that having a client side Telemetry application has been a significant investment that is paying dividends. Scenario Compass has provided us a large amount of data that has guided us to make business decisions around where we spend our test resources. It’s filled a gap that’s existed in desktop applications for some time.

Like all solutions, Scenario Compass has its pros & cons. Being aware of these pros/cons, mitigating or addressing the cons, and making the investments is a business decision that everyone has to make.

- Some of the Pros:
  - Detection of configurations
  - Detection of scenarios encountered cross-referencing the configurations in which they were measured
  - A/B testing is now possible for desktop applications
  - An avenue for direct user engagement allowed us to incentivize certain behaviors through games and rewards

- Some of the Cons:
  - Scenarios have to be explicitly defined in order to be measured. Quality of the Scenario definitions played a key role in usefulness of the data.
  - Usefulness of the Scenario Compass data is proportional to the scale of the Dogfood deployment – it requires a good dogfood program deployment.
  - Need to pay the costs associated with maintaining Scenario Compass & related infrastructure

For the Lync team, we moved ahead with the investment and it has paid off in helping us ship our next Wave of products.

Acknowledgements

Thanks to those who worked on this project. Mahesh Ambravaneswaran, Bruno Torres, John Gilbert, Purvi Vaidya, Vijay Jayaraman, Shankar Arigela, Veronika Ruzin, Joshua Williams, Robert Musson, Harry Emil, Andy Lim, Ross Smith.