How Microsoft Builds Software: Redux

A case study of the mobile development experiences of an SDET intern

By Philip Peng

For Jonathan M. Smith

University of Pennsylvania

CIS-099 Independent Study Report

Table of Contents

Introduction

Internship

Over the summer of 2011, I took on an internship at Microsoft as an SDET Intern (Software Development Engineer in Test intern). It was my first real internship with a large software company; I worked as a part-time IT specialist previous summers and during the school year but only for educational institutions. As an undergraduate studying computer engineering at University of Pennsylvania, the internship aligned well with my field of study and gave me the opportunity to experience software development in an industry setting.

Prior to the internship, I took the course CIS 350 “Software Design and Engineering” with Professor Jonathan M. Smith, which focused on Colwell and Brook’s models of software development. In one particular CIS 350 lecture, we discussed the old Microsoft approach at software development, described in Cusumano and Selby's 1997 article, “How Microsoft Builds Software”. The article was published two years after the release of Windows 95 and their flagship product, Microsoft Office 95. Since then, Windows has received numerous revisions (Windows 98, 2000, XP, Vista, 7) and Microsoft has vastly expanded their product line (Bing, Xbox + Kinect, Windows Phone 7, Windows Live services, etc.).

To efficiently accommodate these various different development requirements, the Microsoft software development cycle has changed over the past 15 years. This report is a case study of my summer experiences as an SDET Intern at Microsoft as part of the mobile division working on a new software product. Since each division and product team adapts different processes specific to their needs, this report is in no way representative of all the official procedures or guidelines followed by the company. I have, however, tried to record my summer experiences as accurately as possible to give readers an insight into how Microsoft builds software.

Team and Product

Prior to my internship at Microsoft, my main experiences are in working on open source software projects for embedded devices. While I was a developer for the "iPodLinux" project, the members of our loosely-knit team lived around the globe and mostly contributed independently on various different aspects of the project. "ZeroSlackr" project was a solo effort, with me handling the development, documentation, and release processes. In my game development experiences with writing "Beats, Advanced Rhythm Game" for the Android platform, I also worked alone, filling all three roles of designer, developer and tester. At Microsoft, however, the projects are of a completely different magnitude and requirement. With a user base of millions around the world, Microsoft's software focus was strongly on quality and reliability. To achieve this, closely-communicating teams would need to be formed for each product with responsibilities distributed to specialized positions. For me, this was the first time working in a rigid team structure on a large-scale project.

The team that I was placed in for my summer internship was unofficially known as the "Mobile SkyDrive" team. Officially, we were part of the "Devices & Roaming Experience Team, Windows & Windows Live Division" (WWL-DRX) as the feature team working on SkyDrive integration for mobile devices. The WWL division covers development of the main Windows operating system (Windows 7 and 8) and accompanying Windows Live brand of products and services (Hotmail, Messenger, SkyDrive, etc.). The DRX team covers devices and remote accessibility services for WWL products, mainly SkyDrive.

Effectively, my team was in charge of writing the mobile apps for Windows Live's new SkyDrive cloud storage service. We would be targeting the three major mobile platforms: Windows Phone 7, iOS, and Android. Despite not having used or even heard of SkyDrive prior to this internship, I was most likely placed on the team due to my previous experiences with mobile/embedded devices development. The product itself would also be a "Version 1" product, scheduled for its first release at the end of September (a month from the writing of this report). For me, this was a great opportunity as it allowed me to watch and participate in the creation of a brand new product from (almost) the start to end of its very first development cycle.

Results

For SDETs there are roughly four main responsibilities: 1) writing test cases and test specs, 2) writing testing tools and preparing the test/build infrastructures, and 3) executing the tests and following up with bug reports to the SDEs (Software Development Engineers, i.e. developers). During my summer internship as an SDET Intern, I was able to experience two of those: writing test cases and writing testing tools. For writing test case, I was responsible for testing some of the basic features of the iPhone SkyDrive app. For writing testing tools, I was responsible for writing a test framework for automated testing on Android.

For the first quarter of the summer, the team was finishing up with the planning phase so I was able to contribute a bit in the design process and get a glimpse of the specifications review process at Microsoft. In the second quarter of the summer, I had written the automated test cases for verifying the "Sign in/Sign out" process and the "Settings" page. The last half of the summer was spent writing a full and complete Android test automation framework and the 36-page documentation for it (see the "Intern Project" section of this report). The iPhone and Windows Phone 7 SkyDrive apps are expected to be publically released by the end of this September, with the final builds tested thoroughly against my test cases. The Android SkyDrive app is expected to be ready by the next milestone and will be tested using my test automation framework. By the end of the summer, I had fulfilled both my responsibilities and was given a full-time job offer as an SDET at Microsoft.

Microsoft Structure

SDEs, SDETs and PMs

In Microsoft's software development structure, there are generally two types of positions: Individual Contributor (IC) and Manager. ICs are the "developers" of Microsoft; they create the products and write the code. Managers focus on the bigger vision and teamwork aspects, writing reports and deciding on the general direction of product development. As an SDET Intern, I would be considered an IC as I wrote actual code. In general, there are three IC positions at Microsoft: Software Development Engineer (SDE), Software Development Engineer in Test (SDET), and Project Manager (PM). Their roles are roughly as following:

Software Development Engineer (SDE):

- MSW Glossary definition: *Individuals who write or debug computer programs and may specialize in one or more methods of creating computer programs, Web pages, or programming languages.*
- Responsibilities include:
    - Prototyping and investigation feature implementations
    - Writing feature implementation specifications
    - Implementing features following PM design specs
    - Focus on scaled stability and performance
    - Write unit tests and fix reported bugs

Software Development Engineer in Test (SDET):

- MSW Glossary definition: *Individuals who test and critique software components and interfaces, write test programs to assure quality, and develop test tools in order to increase effectiveness.*
- Responsibilities include:
    - Preparing and writing automated testing framework/tools
    - Writing test cases and scenario specifications
    - Implementing test cases to test against PM design specs

- Running automated tests and occasionally manual verifications
- Reporting bugs and checking fixes
- Maintaining automated daily build environment

Project Manager (PM):

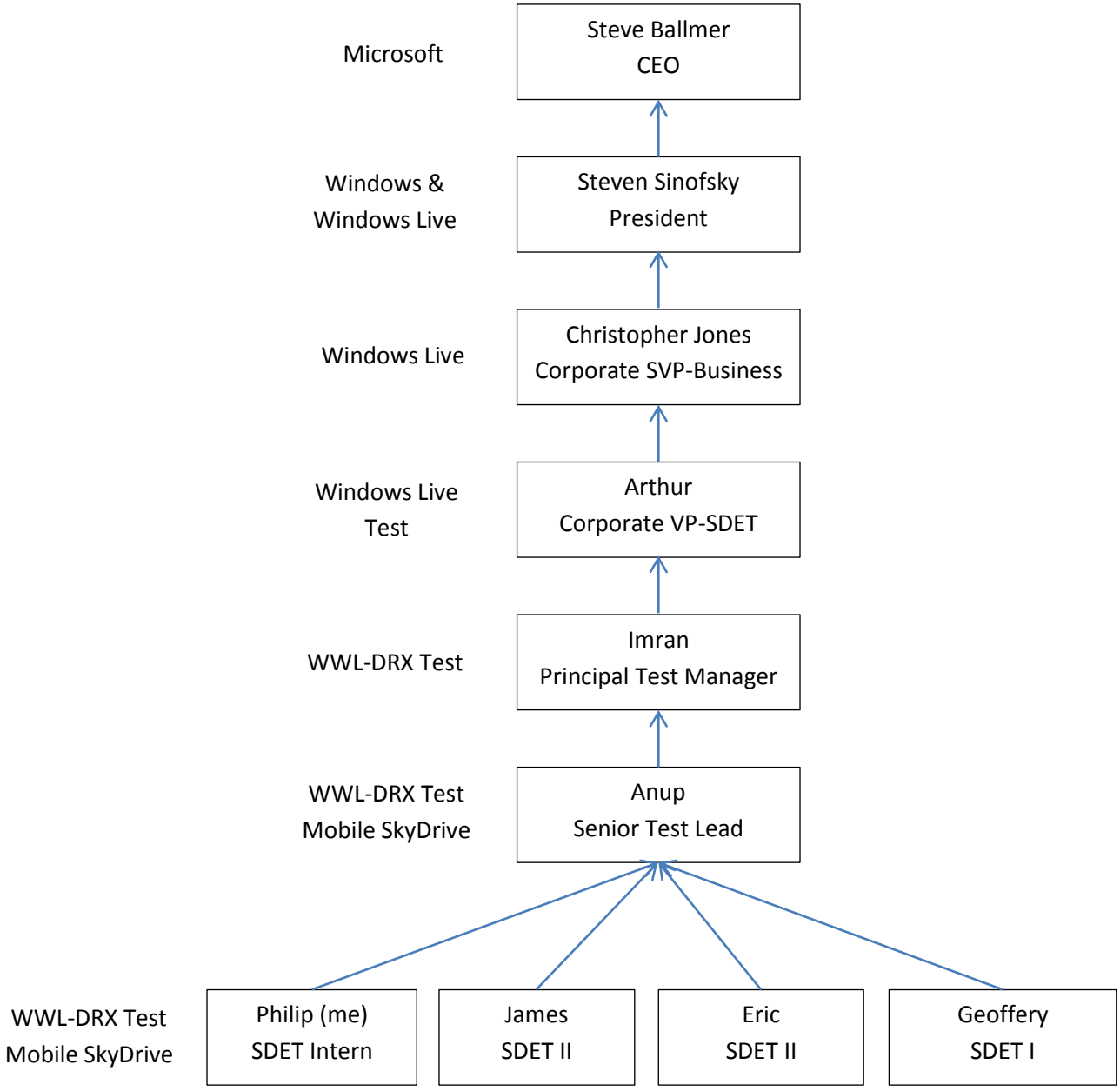
- MSW Glossary definition: *Individuals who are responsible for pulling together and facilitating internal project team communication, driving trade-off decisions, and owning budget and resource planning for multiple projects or programs.*
- Responsibilities include:
  - Facilitate communications with other involved teams
  - Design UI prototypes and interaction behaviour/workflows
  - Writing design specifications and scenarios
  - Managing feature implementation priorities and timelines
  - Organizing internal “dogfood” testing and writing usage documentation

Each feature team consists of SDEs, SDETs and PMs in a rough ratio of 3:3:2, with each role having equal importance and influence on the final product. While PMs may be involved in multiple projects, SDEs and SDETs usually focus on one product/feature line and work closely to follow the PM design specs.

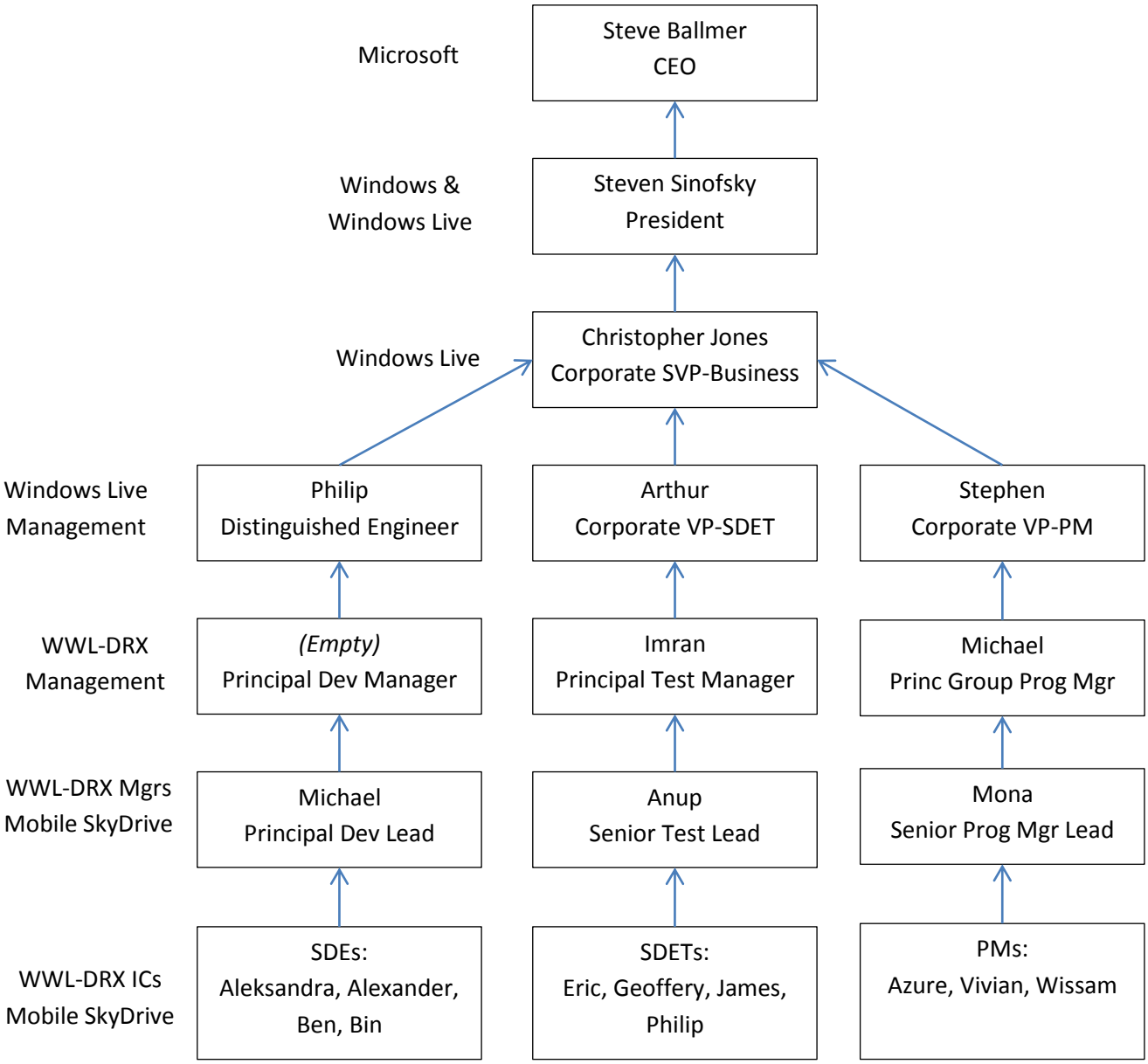
Hierarchy

The "chain of command" at Microsoft is different from most companies. Instead of reporting to a central "team lead", ICs report to the lead of their specific position. As an SDET Intern, I am under the guidance of an SDET II (James) and report directly to the Senior Test Lead (Anup), who reports to the Principal Test Manager (Imran), and so on, only converging with the other IC roles near the very top of the tree and ending with Steve Ballmer.

DRX-WWL Mobile SkyDrive SDET Tree:



WWL-DRX Mobile SkyDrive Tree:



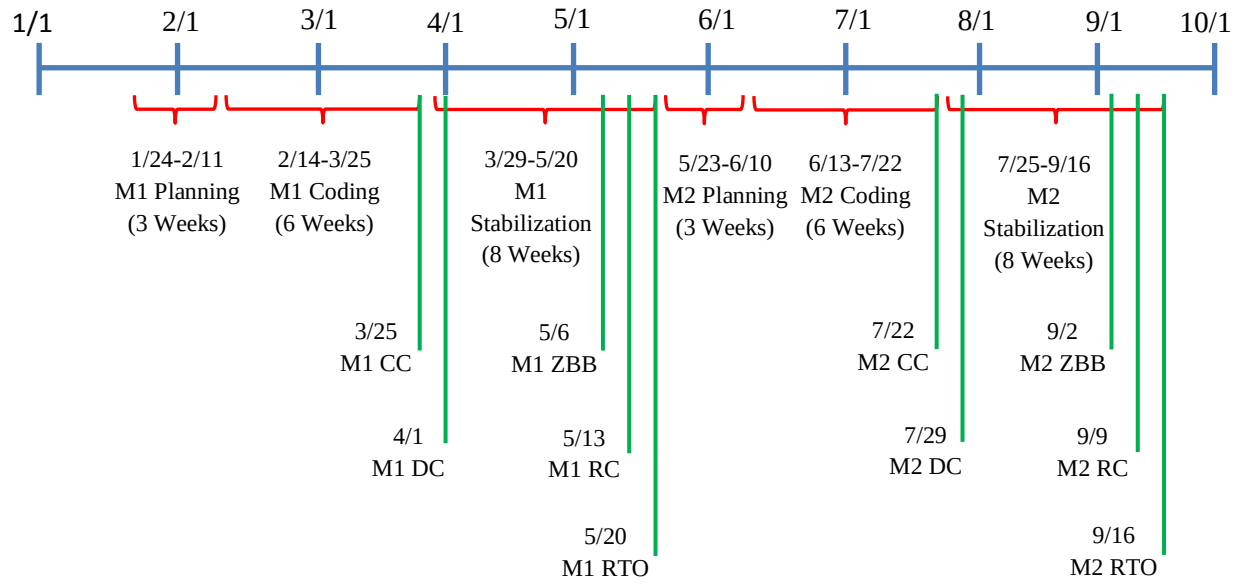
Although the three branches (dev, test, and PM) do not merge until Christopher Jones, there is much interaction on every level of the tree. For our team, the SDEs, SDETs and PMs interacted on a daily basis through Scrum meetings, office drop-bys, demos, and occasionally over lunch. At least one of the leads would be present at every Scrum meeting and the three leads would meet weekly to evaluate the current project progress and schedule. The test lead Anup would also hold a weekly Scrum-like meeting just for his directs (e.g. SDETs that report to him). Once every few weeks, the entire DRX test team under Imran (there were three DRX teams) would also gather to give demos, go over progress statistics of each team, and discuss employee feedback and/or concerns.

Microsoft Development Process

Timeline and Milestones

In Windows Live, milestones are grouped together by “Waves”. A Windows Live Wave can be thought of as an iteration of the Windows Live platform; the Wave 4 update released last year introduced numerous improvements in it online services such as a built-in Web Messenger and Essentials. Each Wave containing four major Milestones: MQ, M1, M2 and M3. MQ is known as the “engineering phase” with little or no PM involvement. During MQ, the SDEs and SDETs set up development infrastructure, design and inspect the architecture, and prepare the hardware. M1 is the first development milestone and consists of planning, coding, and stabilization phases. M2 and M3 are repeats of M1 with the inclusion of user feedback and the addition of work items not completed in previous milestones. An example early version of the Wave 5 timeline is shown below. When I joined Microsoft at the beginning of the summer, we were just at the start of M2 of Wave 5.

Windows Live Wave 5 Example Timeline:



Planning:

This is the organization and spec-writing phase. During this phase, the feature list is brainstormed and prioritized. User feedback or marketing research is heavily consulted if available. The most important features are selected for the current milestone and ownerships of features are assigned to each team member. The PMs write design specs detailing the exact user interaction and response flow for each feature as well as create mock UIs. The SDEs write implementation specs following the PM specs for each feature. The SDETs write test methodology specs and user scenarios that would test out the target features. Throughout the entire process, prototyping is done to give rough estimates of the "cost" of the feature in "days" (equivalent to Brook's "man-day") and scheduling adjusted accordingly. In total, the planning phase is allotted three weeks.

Coding:

This is the pure code-cranking phase. During this phase, the SDEs turn their implementation specs into actual code. The SDETs set up the daily building process and write code for running automated test cases, then run the test cases or engage in manual testing as SDEs commit code. The PMs communicate with other related teams (such as, in our case, the SkyDrive Backend team) for updates and prepare market data and documentation for "dogfooding" (see below). The coding phase concludes when all features are complete (or moved to the next milestone) and "code complete" is declared. In total, the coding phase is allotted six weeks.

Stabilization:

This is the break-and-fix phase. This phase starts with the internal "dogfood" testing. During this phase, the SDEs work with the SDETs to resolve all the bugs caught by the automated and manual testing. PMs work on release documents and setting up the market for the release, as well as start planning for the next milestone with feedback from the dogfooding in mind. While all feature designs are frozen during the phase, implementations are improved for stability and performance. Once all bugs have been declared as closed or a as a feature moved to the next milestone, Zero Bug Bounce is declared and the final product goes through as a Release Candidate, then Release To Operations. In total, the stabilization phase is allotted eight weeks.

Code Complete (CC):

When all dev code has been written and tested for basic functionality, the project has reached "Code Complete". This is usually a celebratory day and indicative that the product is now ready for dogfooding. The MSW Glossary definition of code complete is as

following: “A development milestone marking the point at which all features for the release are implemented and functionality has been verified against the functional specification.”

Dogfood Complete (DC):

After “Code Complete” has been declared, the product is almost ready for the dogfood process. MSW Glossary defines dogfood as: “Software code not fit for public consumption but good enough for internal purposes, very unrefined and buggy (that is, full of bugs), but containing the basic nutrients.” In other words, dogfooding is the process of internal testing and feedback, usually first starting with the feature team itself, then expanding to the product team and finally the entire division. “Dogfood Complete” is declared once the completed build is available and dogfooding instructions are ready.

Zero Bug Bounce (ZBB):

Zero Bug Bounce is the target state at the end of the stabilization phase. This is the stage where all bugs have either been resolved or turned into features for the next milestone. MSW Glossary more formally defines this as: “The first point in time after code complete when there are no active bugs older than a certain amount of time (typically several days, pre-defined based on the end-to-end time required to resolve a bug).”

Release Candidate (RC):

The Release Candidate is the first build ready for public consumption. MSW Glossary defines RC as: “Builds of products produced with no known issues that the product team believes should prevent them from being released to manufacturing or to the Web.”

Release to Operations (RTO):

Once the Release Candidate is ready, the next step would be Release to Operations, referring to Microsoft Operations (SMSG), the group that handles the public release process. MSW Glossary defines RTO as: “The point in the product development process at which the software and documentation of the product are released to operations groups.” Once the Operations group receives the build, they will make the product available to the relevant consumers (e.g. OEMs, commercial licensers, public website, etc.). Immediately following RTO is the start of the next Milestone, beginning again with the Planning phase.

Agile Development Methodology

Mobile development is a relatively new phenomenon in the software development industry. For in particular, entering the mobile market requires a lot of change in the software development process. Large scale projects such as Windows or Office usually have development cycles lasting many months or years resulting in a largely stable product that is only periodically updated with patches. In the mobile world, however, fixes and new updates are expected on a monthly or sometimes weekly basis. To adapt to these different expectations, many processes had to be heavily modified or, like in our team's case, scraped and replaced with something new. For the Mobile SkyDrive team, the decision was made to use the agile development methodology with the scrum framework.

MSW Glossary defines "agile" as "A people-oriented, adaptive methodology for application development that focuses on short iterations and customer interactions." At The official "Principles behind the Agile Manifesto" document reads as follows:

1) *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software*
2) *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
3) *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
4) *Business people and developers must work together daily throughout the project.*
5) *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
6) *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
7) *Working software is the primary measure of progress.*
8) *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
9) *Continuous attention to technical excellence and good design enhances agility.*
10) *Simplicity--the art of maximizing the amount of work not done--is essential.*
11) *The best architectures, requirements, and designs emerge from self-organizing teams.*

12) *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.*

Our team's main focus was to create a free app that will allow users to effectively use their mobile devices to access the free SkyDrive service. To accomplish this, we would be very customer-based and frequently release updates based on user feedback of their experiences with the app (after the first release). If the users decided that they preferred a different feature or implementation or some technical roadblock appeared preventing us from executing our original plan, we would need to be able to react fast. The agile development methodology would allow us to do just that.

Scrum Process

In addition to adhering to agile development principles, we kept on-track by integrating parts of the Scrum process into our development cycle. MSW Glossary defines "scrum" as "An agile, lightweight process that can be used to manage and control software and product development using iterative, incremental practices." In rugby, a "scrum" is a method used to decide on restarting a play after the ball goes out of bounds. The software development "Scrum" can be seen similarly; the Scrum process is used to restart the development cycle when an unexpected change in plan or user expectations happens. We implemented a few key aspects of the Scrum process: backlog, burn-down chart, sprints, daily stand-ups, and demos.

In his article titled "Agile Development: Lessons Learned from the First Scrum", Dr. Jeff Sutherland documented the first usage of the Scrum process in software development. Despite happening almost 20 years ago, much of his experiences and strategies still work today and were applied with visible effects by our team. In his article, Sutherland wrote:

The first Scrum started with a half day planning session that outlined the feature set we wanted to achieve in a six month period. We then broke it into six pieces which were achievable in 30 day sprints. This was the product backlog. For the first sprint, the product backlog was transformed into development tasks that could be done in less than a day.

Instead of just a “half day planning session”, our team expanded this portion to the entire planning phase. User expectations and target features were brainstormed, researched, filtered, and listed as requirements in our “product backlog”. This feature count would then be placed onto a downward-sloping “burn-down chart” that would be updated weekly to reflect the current project “code complete” status. These backlog items were also categorized into four priority levels: P0, P1, P2 and P3 with P0 being required “user stories” and P3 being less important “stretch goals”. The PMs drafted UI mock-ups, designed feature usage workflows, and wrote specs describing the user behaviours that would achieve desired results. Through prototyping and investigation, the SDEs recorded the estimated “cost” and relayed back to the PMs on whether or not the feature should be changed or scrapped if the cost was found to be too high. In the meanwhile, the SDETs went through the PM specs and wrote test specifications and test cases covering expected and unexpected user behaviours and interactions and their expected results. At the end of these investigations and spec writings was the final sprint-planning phase in which each backlog item was assigned to either Sprint 1 or Sprint 2 of our Milestone 2 timeline (we divided M2 into two “mini-milestones”/sprints called S1 and S2).

Another important aspect of the Scrum process that we followed was the daily stand-up meetings, also known in MicroSpeak as “brown bags” (due to the historical inclusion of a brown bag containing cookies at such meetings). MSW Glossary defines the term as: “Short, informal training or informational meetings that generally occur over the lunch hour, designed to fit into the schedules of individuals who might not be able to attend at other times of the day.” Sutherland outlined the following requirements for Scrum meetings:

The meetings were kept short, typically under 30 minutes and discussion was restricted to the three SCRUM questions:

1. What did you do yesterday?

2. What will you do today?

3. What obstacles got in your way?

To further encourage the brevity of the meetings, our team's daily brown bags were scheduled at 11:30 am such that we would walk out half an hour later ready to go for lunch together. Despite their brevity, however, I found the daily meetings to be the highlight and winning point of the Scrum process. Going around the table, each member of the team was able to become informed of the latest updates and changes through just a handful of sentences. As an SDET, it was immensely helpful as it allowed me (or my fellow SDETs) know of when a changeset had been or will be checked in such that I could plan my test case writing or running. For SDEs, the face time allowed them to discuss implementation issues or challenges with their fellow developers as well as request clarifications or suggest changes to the PMs. For the PMs, the meetings were a chance to verify that the project was still on track and plan for arranging meetings with other teams if necessary. It was very common for the brief 30 minute meeting to be far more productive than hours of back-forth email and instant message conversations.

The final and most anticipated part of the Scrum process, however, was the demos. Sutherland described his Scrum demo experience as followed: Every Friday during the first Scrum, we held a demo and brought in development experts from other companies in to look at the product. As a result our developers had to do the demo for their peers in other companies. This was one of the best accelerators I have seen in software development. An outside expert would say, "That's terrible; look at Borland's Product X to see how it should be done" or "How could you possibly [sic] have a dumb bug like that?" As a result of this outside input, all problems or bugs would be fixed the following week. Developers refused to be embarrassed a second time in front of their peers.

Rather than having a regular scheduled demo time, our team gave quick demos to each other either during our daily brown bags, over lunch, or whenever we dropped by each other's offices. Since Mobile SkyDrive was a Version 1 product, demos during the first half of the summer were mostly prototypes of specific feature implementations. I remember one meeting where one of our Windows Phone 7 SDEs complained about the lagginess of the default picture transition animation and showed it to us. The next day, he came in smiling and demoed off his "slideshow" hack (loading adjacent pictures and just applying a translation to the expanded canvas, then reload on animation completion). Being able to immediately see the improvement definitely gave a rewarding feeling and

morale boost to the entire team – one less worry for the SDEs, one less design change for PMs to consider, and one less "bug" for SDETs to test.

Sometime during second half of the summer, we had our first "working" iPhone SkyDrive build. It was a huge celebratory moment one of our SDEs took out his iPhone and showed us a picture on his phone that he had just uploaded to SkyDrive a few minutes before the meeting. It was equally interesting when I dropped by one of my fellow SDET's office later that day to try playing around with the app myself. While randomly tapping the iPhone's screen, the app completely froze and the UI stopped responding. I showed him his iPhone, to which he responded, "Uh oh, we definitely can't ship with that." An hour later, a bug report had been filed and the SDEs were busy investigating the root cause.

Changes Since 1997

In 1997, MIT professor Michael A. Cusumano and UC-Irvine professor Richard W. Selby wrote an article titled "How Microsoft Builds Software." Using research into the development of products such as Windows 95, Windows NT, Microsoft Office, etc., Cusumano and Selby described the common Microsoft software development process and "synch-and-stabilize" methodology. Almost 15 years later, the core principles are still the same but the details have changed.

According to the article, Microsoft had 20,500 employees and annual revenues of $8.7 billion in the fiscal year ending June 1996. According to Microsoft News Center, Microsoft had 90,400 employees and annual revenues of $70 billion in the fiscal year ending June 2011. Since Windows 95, Microsoft has come out with Windows 2000, Windows XP, Windows Vista, and Windows 7; Microsoft Office has also increased numerous versions as well. The company breath has also expanded to cover more hardware, game consoles, mobile devices and phones, and recently cloud-based computing. As a result, there are numerous variations in development processes between product lines and even teams, with each development process tuned toward the team's particular needs. For comparison purposes, I will contrast the processes described in Cusumano and Selby's article with those that I saw used in Windows Live.

Cusumano and Selby describe Microsoft's organization as a "scaled up" version of the loosely structured hacker-style with multiple small teams working in parallel to build large products. The parallelism still exists with small teams being assigned features of a big product, but there are no longer signs of loose hacker-style design autonomy. Through the use of design spec templates and feature tracking/prioritizing and other organizational tools, development follows a much stricter, professional procedural path than described in the article. There is generally a greater emphasis into professionalism and stability instead of feature-richness and innovation.

The change synchronization aspect has also become more formal. All code commits must first go through a "code review" conducted by members of the same team. Only once the changeset has been "signed off" by required members is the changeset allowed to be committed. If the automatic build process is in place, the changeset will only be merged in after a clean test build with the new changes has been completed without errors (this can take an hour or longer). Consequently, the "frequent synchronizations and periodic stabilizations" model has changed to "synchronize on feature complete and stabilize immediately after". Despite this change, "milestones", "daily builds", "nightly builds" and "zero-defect" are still commonly used terminology.

The Planning and Stabilization phases have generally stayed the same, but the testing during the Development phase has become more parallelized. Rather than testing feature sets in chunks, tests are written and executed as the feature gets completed and committed. In the down time during which SDETs were "blocked" due to a specific owned feature not yet having been completed, the SDETs would work on another task, such as maintaining the build tools/process, writing testing tools, or even writing prototype test case code without having anything to test against. In my mobile team at least, our SDETs would always be waiting on the SDEs to finish a feature, but we would always have something else to do in the meanwhile.

With Product Managers handling all design aspects of development, SDEs now are far less involved in design decisions. During my brief presences at some of the planning meetings at the beginning of the summer, I saw on a few occasions SDEs or SDETs disagreeing with the PMs on what the users want most or how the user will most likely

interact with the app. In the end, however, the PMs had the final say and the SDEs were to follow the PM specs. While this may seem overly restrictive on the SDEs, it does exemplify Microsoft's shifted preference toward more professional design choices (developers are not known to be the best designers).

The development process itself has also evolved to varying degrees depending on the team. Agile development seems to be the main development model in Windows Live, with my Mobile SkyDrive team taking the extreme approach of sprints inside milestones and usage of the Scrum process (introduced roughly around the same time Cusumano and Selby's article was published). With the design-change freeze during the 6-week Coding stage, however, there is still a hint of sequential development rigidness that ignores customer demand changes during that period (e.g. no adaptation until 4 months later). This was described to me by my manager as a positive however; by locking down your design during a milestone, you are guaranteeing your customers a stable, reliable product, even if it's not cutting edge and fits their needs perfectly. Although this contributes to Microsoft's slowness in reacting to the marketplace demands, it strongly establishes Microsoft's stance and focus on professionalism and stability.

Intern Project

Process

Every year, full-time employees (FTEs) at Microsoft undergo a review process that includes an initial commitments meeting, midpoint review, and a final review. For summer interns, the process is shrunken (to 12 weeks in my case). In addition to watching a series of online videos for the "ramp-up" training process and a New Employee Orientation (NEO) involving some mixer events, presentations, and general overview of company policies, all interns sit down with their manager during the first week and draft the intern commitments document. The expected content of the document varies from manager to manager, but the Interns Commitment document roughly outlines the Intern Project goals for the summer, how they will be accomplished, and what the expected results are. At the midpoint review, the progress on each commitment is checked and the manager gives feedback on performance so far as well as expectations for the second half of the summer. The final review meeting is conducted during the last week of the internship to see if all the expected commitments were met and if performance was satisfactory. As a Windows Live intern, we also were expected to give a short presentation to test managers and executives.

Sample intern commitment:

Commitment:	**Execution Plan:**	**Accountabilities:**
Android Automation Framework Analysis of current application view	Write built-in library that is compiled with the application itself and has access to the application's main context and views: - Use code samples from public Google API + documentation - Build hierarchy of all UI elements and common/specific shared characteristics for different UI elements - Make sure all hierarchal structuring is consistent with the iPhone automation framework's layout - Write a sample test app with all testable UI elements and multiple views	Framework can perform the following: - Locating UI elements - Retrieving UI element properties - Determining layout of screen and visibility of UI elements to user

Project

At Microsoft, manual testing is kept to the bare minimum wherever possible. This is because builds are made daily and all tests need to be run against each build. These extremely comprehensive tests/verification checks can take many hours to run if done manually, but only a few hours by computers. As a result, it is far more time efficient to write automated tests that simulate user behaviour and can be run remotely and logged for inspection later. In our particular case, since we were dealing with mobile devices, all our testing was done by running test scripts against emulators running on dedicated computers that built and ran tests every evening (so we could come in the office and check the logs first thing in the morning). Manual testing was only necessary if the logs were unclear or if the testing tools was not powerful enough to cover the desired test. The combination of testing tools and scripts are often referred to as the test automation framework. The test automation framework would essentially contain all the components needed to simulate blackbox-like user interaction with the product.

My intern project for the summer was to write a test automation framework for the Android platform. Since the Mobile SkyDrive project targeted Windows Phone 7, iOS and Android, we needed to have test automation frameworks for all three platforms. Prior to the formation of the Mobile SkyDrive team, members of my team had worked on the iPhone Messenger app and had developed a test automation framework for iOS. As I was the only one on the team with experience in Android development, I was put in charge of filling in the Android component.

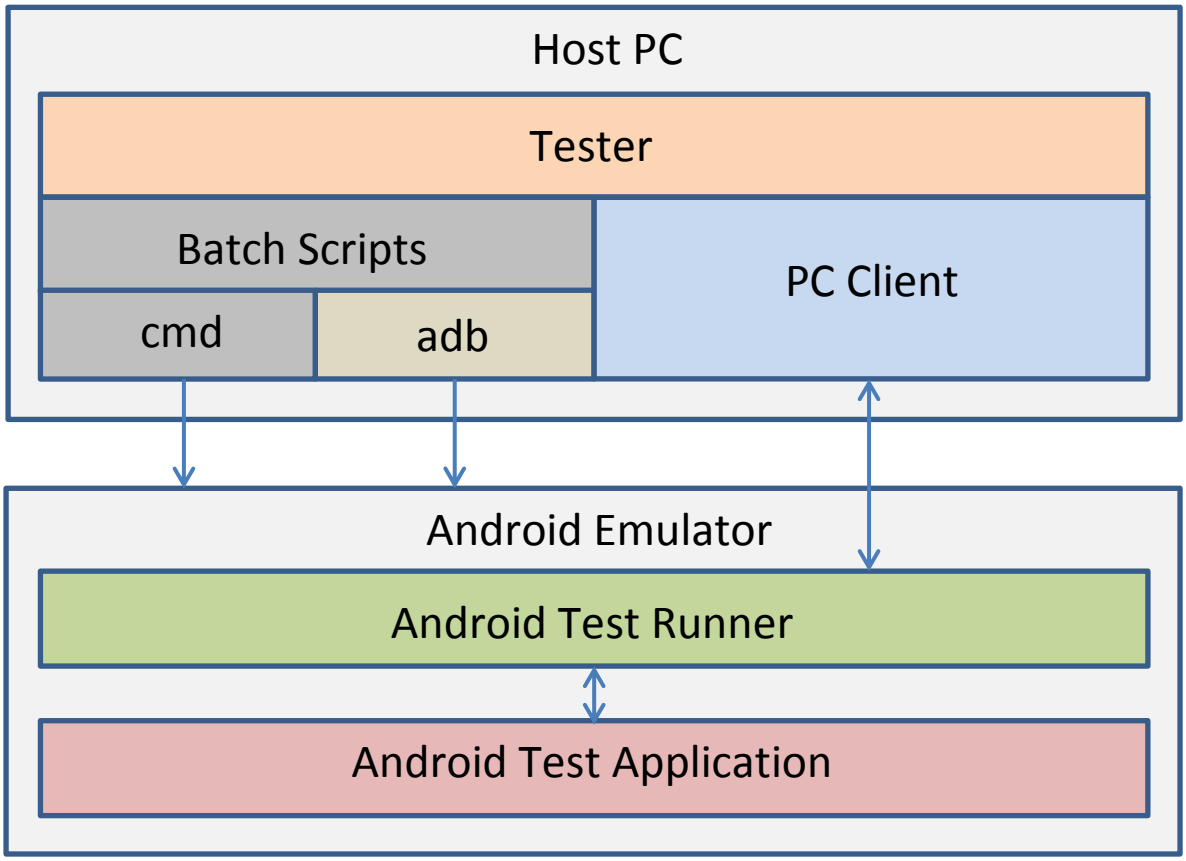
Implementation

When my manager and I met at the beginning of the internship to discuss the intern project, the Android test automation framework had four initial requirements:

- *Launch the test application*
- *Inspect the current visible screen for UI elements*
- *Verify state information of those UI elements*
- *Simulate user behaviour on those UI elements*

To achieve these goals, my Android Test Automation Framework contained four main components: 1) PC Client, 2) PC Scripts, 3) Android Test Runner, and 4) Android Test Application. The 1) PC Client is a PC program that sends command requests (e.g. test case commands) to the Android device running the Android Test Runner. The 2) PC Scripts are a series of scripts that handle Android device management (e.g. package updating and launching). The 3) Android Test Runner is a custom Android instrumentation test case, run by the Android OS's built-in InstrumentationTestRunner application, which extends testing capability by implementing an HTTP server that dynamically receives commands and executes them. The 4) Android Test Application is the production Android application that is under instrumentation and testing. Without going into details of how each aspect was implemented (apart from the Android Test Application, which would be written later by SDEs and thus is treated as a blackbox), the overall abstraction layer diagram is as following:

Android Test Automation Framework Abstraction Layers:



Results

Although the intern project was timelined to take the entire 12 weeks of my internship, due to some team reorganizations near the beginning of the summer (and subsequently the shifting of some iPhone feature testing responsibilities to me), the project did not begin until the end of the fifth week (halfway through my internship). Nevertheless, I was able to completely complete all the initial requirements as well as implement a lot of additional "stretch-goal" features. The final generalized feature list was as following:

- *Starting up the Android emulator*
- *Unlocking an Android device/emulator's lock screen*
- *Installing/uninstalling packages*
- *Starting the Android Test Runner (a specially modified JUnit test case)*
- *Starting the Android Test Application*
- *Running an HTTP server that receives requests and returns responses dynamically*
- *Parse XML messages and execute parsed commands*
- *Locating UI elements by name, ID, tag, and/or text on the Test Application*
- *Constructing a UI element layout tree for the Test Application*
- *Retrieving UI elements properties in the Test Application*
- *Performing various actions on UI elements of the Test Application*
- *Performing touchscreen and keyboard/button input actions on the Test Application*
- *End the Test Runner and Test Application*
- *Stop the Android emulator*

In addition to fully explaining and demoing off the test framework to the other SDETs in my team and my manager, I also wrote a 36-page documentation containing usage details as well as implementation notes (my manager was delighted and joked that I went a bit overboard). During my last week, I also gave presentations of my intern project overview and live demoing of the test framework, first to my Mobile SkyDrive team under Anup, then to the WWL-DRX Test team under Imran, and finally to a group of WWL test managers under Arthur, including Arthur himself. The presentations and demos were very well received, with Arthur following up with me about the sharing of the test framework with other mobile teams in the future. The internship ended with me finally committing the code and documentation to the Source Depot (centralized source code server).

Future Impact

By the end of my internship, our project was into the Stabilization phase of Milestone 2. In a few months, the Coding phase of Milestone 3 will start and the development of the Mobile SkyDrive app for Android will begin (the prototype I had included in my test framework was for testing purposes only but it had some reusable code that will probably be used as reference by the SDEs). When the first Android build appears, the SDETs on the team will need to start writing automation test code to test that build and will be using my test framework to execute that test code. At the time when I had finished my internship, my Android test automation framework was actually much more powerful and comprehensive than the one we had in place for iOS. Since the code was well commented, implementation details recorded in the documentation, and other SDETs on my team familiarized with using the framework, I am expecting it to be the foundation for Android testing in the common test framework that my team planned on developing (I had also included a detailed To-Do list in the documentation along with implementation suggestions/ideas). Seeing the rise of mobile development needs in Microsoft with other product teams also planning on creating mobile clients, there will be a need for a centralized, in-house common test framework. If my team decides to generalize certain parts of our common test framework, I envision it being used by other mobile teams outside of DRX.

Internship Reflection

When I joined my team at the beginning of the summer, my team was just coming out of the Planning phase. For the entire first week, I had the chance to sit in the design discussion meetings as well as all the final spec reviews, which essentially represent the cumulative results of the Planning phase. When I finished my internship at the end of the summer, my team had just declared code complete and was at the start of the Stabilization phase. The opportunity to be involved in almost the entire software development cycle greatly helped me understand Microsoft's development methodology.

In addition, Mobile SkyDrive was a Version 1 product. Due to the lack of a precedent as well as the fact that the team was newly formed, the schedule/timeline changed numerous times over the summer. Numerous features were added and removed between sprints and milestones, and costs estimates for the features were regularly readjusted. A few changes in the team itself also resulted in a few shifts of responsibilities and feature ownerships, some of which lead me to being unable to start my internship project until midway through the summer. Nevertheless, the entire experience of being part of a team developing a new product and having to constantly adapt to changing circumstances was very exciting and challenging.

In terms of the development methodology itself, I was left with mixed impression. As a developer who has work mostly alone in the past and also been largely responsible for the designing, developing and testing aspects of project, it was a new experience working in a team setting where the three roles were divided up. As a single person, there were no need for spending time writing specifications and changes in plans were very easily done. Being placed in a setting with rigorously enforced distinction of roles was a big change for me. For larger products such as Office or Windows where hundreds of employees are involved, I do agree that the highly structured, procedural process of design, freeze, develop, test, stabilize, and periodic large milestone release is necessary to keep organized and sane.

For much smaller independent products such as Mobile SkyDrive, I see usage of the same procedures with all the mandatory “red tape” drastically as major hindrances to development efficiency. This is especially the case in mobile markets where users expect regular updates on a weekly basis and all bugs to be fixed within a few days of discovery (instead of in the next major release). That said, I saw the usage of the Scrum process as a highly effective tool to drastically counterbalance the problem by significantly increasing and encouraging team communications. Since mobile development is still a new and growing field, I’m sure things will be very different in coming years.

Works Cited

- "Fast Facts About Microsoft." Microsoft New Center. <http://www.microsoft.com/presspass/inside_ms.mspx>.
- "MSW Glossary." Microsoft Web (Microsoft internal resource). <http://msw/AboutMicrosoft/Glossary/Pages/default.aspx>.
- Brooks, Frederick P. *The Mythical Man-Month*. Addison-Wesley: 1975.
- Colwell, Robert P. *The Pentium Chronicles*. Willey-Interscience: 2006.
- Cusumano, Michael A. and Richard W. Selby. "How Microsoft Builds Software." CACM 40(6): 1997, pp. 53-61.
- Highsmith, Jim et al. "Principles behind the Agile Manifesto." Agile Alliance: 2011. <http://agilemanifesto.org/principles.html>.
- Sutherland, Jeff. "Agile Development: Lessons Learned from the First Scrum." Scrum Alliance: 2004. <http://www.scrumalliance.org/resources/35>.