A practical approach to Agile Software Development

Mahesh Prabhu
Head - Microsoft Technology Practice
INTRODUCTION

With the ever changing requirements, technology & innovations, it is imperative for software companies to be able to leverage the change and pass the benefits to its customers. A static model like the Waterfall – which has been widely practiced in the software houses over the last couple of decades – though accommodates changes through the Change Management processes; the overall experience is not pleasant for both developers & customers.

Agile software development enhances the customer & developer experiences in adapting to changes all through the project life cycle.

In this document, I have described our adaptation of Agile in Microsoft Technology software projects.

Agile@ITC Infotech

Agile@ITC Infotech is an iterative and incremental approach to software development fuelled by collective ownership of closely knit team that strives for superior quality output in aggressive timelines.

Our adaptation of agile incorporates the salient features of
Scrum
Extreme Programming
Continuous build and integration
Test Driven Development

Once the high-level specifications, architecture and design are drawn out, the technical tasks are classified into (a) Complex tasks, (b) Common tasks, and (c) Monotonous tasks.
Each type of task is accomplished through a Sprint. Expert Sprint and Extreme Sprint should be accomplished before starting the Expected (habitual) Sprint. Most of the elements of the Sprint Cycle remain the same for all types of Sprints. The minor variations are described below.

(a) Complex tasks
- Requires research on how to do
- Calls for a Proof of concept (POC)
- Requires expert coders
- Typically accounts for 10% to 15% of all tasks

Expert Sprint: these are taken up by the expert developers. Most of the times the technical architects or tech leads themselves take up this sprint.

(b) Common tasks
- Need to be completed before others
- Reusable components, objects and services
- Unavailability of these would result in duplication of code & effort
- Typically accounts for 15% to 25% of all tasks

Extreme Sprint: these activities are best addressed by Pair Programming or Extreme Programming. The output – in terms of speed, quality & innovation – is expected to be more than that of 2 programmers writing code individually. These units should be coded quickly as many other units will be dependent upon them.

(c) Monotonous tasks
- Repetitive in nature
- Developers have done such work before
- Not much of research involved
- Typically accounts for 60% to 75% of all tasks

Expected (habitual) Sprint: all project members participate in this Sprint individually. The lack of complexity in the units does not warrant pair programming or high degree of expertise.
The Sprint Cycle

The project life cycle comprises of iterative Sprint Cycles. A Sprint begins with identification of the Sprint Backlog or Sprint Specification – a subset of the overall specification. Typically, a Sprint lasts for 3 to 6 weeks, during which the team creates working/usable increment of the software. Every member takes ownership of a section of the Sprint backlog.

- Scrum meetings (also known as Stand up meetings) are held every day to check the progress and discuss any hurdles.
- Every member including the Project Manager does the actual work – design, code, etc
- Project Manager has additional responsibility of ensuring the team has no hurdles
- Customers & other stakeholders are always kept informed of the progress; they provide regular feedback and inputs during the planning of each Sprint
The activities carried out in a Sprint Cycle are given below:

Shells and Wire-frames: these form the low level design & program specifications. The developer creates the UI, Classes with function/event/property definitions. These are created directly using the IDE – Visual Studio. Comments are then placed into the empty shells which describe the functionality & logic (what and how) that have to be coded.

- Gives clarity to the developer as to what has to be done and how
- Reviewer can see upfront whether the developer has understood the requirements for that unit
Unit test scripts: NUnit or VSTS is used to write automated unit test scripts. Writing the test cases upfront also ensures that the functionality is understood by the developer.
- Both Valid and Invalid cases are scripted
- All tests should fail initially, when tested with the Shells and Wire-frames

Construction: construction comprises of coding, code review, unit testing, refactoring and fixing defects.

FxCop is used for automating the code review. The standards are scripted into the FxCop rules upfront. FxCop is run every time a unit is checked-in so that the developers can get early indications of their quality of code with respect to standards and best practices. Profilers and Code Analysis are also run at regular intervals to check the performance, memory/CPU usage, etc.

NUnit or VSTS is used to automate the unit testing. As the unit test scripts are written upfront, running the unit tests will hardly consume any additional time or effort.

Re-sharper or other Visual Studio plug-ins is used to re-factor the code for better readability and maintainability.

QA Test: this is done by someone other than the developer who coded the unit.

Continuous Build and Integration: achieved through tools like Cruise Control.net, VS TFS or by writing custom scripts.
**Agile @ ITC Infotech**

A practical approach to Agile Software Development

- Avoid integration issues at a later stage
- QA test can happen parallel to construction
- Customer & other stakeholders can get a sneak peek anytime and provide early feedback

**Release – working increment of the software**: There can be multiple development releases and production releases in the project life cycle.

**Benefits of Agile @ ITC Infotech**

Agile @ ITC Infotech is an evolved home-grown adaptation of the agile concepts that is practical & easy to apply to software development projects in the Microsoft Technology space. We have experienced significant benefits including:

- Higher productivity compared to the traditional project life cycle models
- Enhanced learning from each other
- Better output due to collective ownership & involvement
- No unpleasant surprises at the end – transparency of progress with all stakeholders
- Better quality of code
- Minimize technical risks & challenges as Expert Sprints are carried out upfront
- Lastly, agility!!