VirX.Net Software Innovations Inc.

Software Development Process Specification

Version 1.1

September 2nd, 2017

The Waterslide Methodology

Draft 1

Antony Peiris

Table of Contents

1. Revision History

Date	Version	Author	Note
Jul 22, 2017	1.0	Antony Peiris	Initial Draft
Sept 2, 2017	1.1	Antony Peiris	Grammar Fix

2. Introduction

2.1. Document Format

This document adheres to the VIRXTDOC-11.112 documenting standard. The VIRXTDOC-11.112 format is a simplified technical documenting scheme developed by VirX.Net Software Innovations Inc. It has been designed to help simplify complicated technical specifications so that they can be easily understood by both non-technical (non-tech savvy) stakeholders and developers alike. Please see the Document Format Specification section at the end of this document for more details.

2.2. Purpose

The purpose of this document is to define the specifications of The Waterslide Software Development Process. It will explain the purpose of the methodology and it's uses as well as how it may be implemented. The document is intended for both technical and non-technical individuals to have a clear idea of how to effectively utilize this method in software development projects.

2.3. Scope of Methodology

The Waterslide method is intended for start-up software projects which have the potential (and the need) to evolve into large systems with long term maintenance and scalability requirements.

2.4. Definitions

3. Overall Description

3.1. Introduction

The Waterslide model is focused on rapid development cycles, strict specification standards, and thorough quality assurance policies while still being flexible enough to allow natural software evolution, scalability, and growth. The Waterslide method aims to simplify large projects by breaking them down into well-defined Waterslide cycles. The end goal of the methodology is to achieve a high-quality professional product. The method is geared to accomplish these goals quickly, efficiently and cost-effectively while ensuring continuous to growth through rapid cycle iteration. While the Waterslide method draws its inspiration from existing software development processes such as the Waterfall method, Agile, etc… it is quite unique and different in itself. Some of the key aspects which make the Waterslide method different is that it insists on simplicity and transparency of the software development process itself. In addition, it also demands that all related paperwork strictly adheres to the VIRXTDOC-11.112 documenting standard. Most of all, the Waterslide method demands that the complete development life cycle is enjoyable and fun for all parties involved. Just like a water-slide should be.

The name “Waterslide” is symbolic to the development principles and philosophy that this methodology is built upon. In short, it demands a quick and controlled life cycles with minimal surprises. It is meant to be simple, fun, fast, fluid and cost-effective. With the Waterslide method, time is never wasted on too much planning or too much development. Once a cycle is complete, the entire process is repeated again with new specifications. Everyone involved would know what to expect and where they are headed.

3.2. The Model

3.2.1. Software Requirements Specification (SRS)

The Software Requirements Specification (SRS) drafting phase is the most vital part of the process. This is where the developer(s), designer(s), project owner(s) and all stakeholders(s) get together and discuss what needs to be done. This phase is broken down into its own sub-cycle which involves the following stages:

1. **Discussion and Planning (Consultation)**
2. **Document Drafting**
3. **Document Review**
4. **Repeat or Finalize**

This cycle must always adhere to the following policies:

1. **The SRS document must adhere to the VIRXTDOC-11.112 documenting standard.**
2. **The initial SRS does not have to be perfect.** It is meant to serve as a starting point only. All subsequent SRS documents must aim for perfection.
3. **A fixed *development period* must be defined when going through the initial SRS iteration**. The *development period* should be short (depending on how all involved parties agree to define "short"). For example, 7 days, 14 days, 30 days, etc…
4. **The *development period* should never exceed 30 days**. Keeping the development period as short as possible is recommended to make the best of the Waterslide methodology.
5. **The SRS must never represent development work estimated to exceed the previously defined *development period*.** The recommended amount of actual development time the SRS should represent is 75% or less of the *development*

period value.

6. **The initial SRS must include core essential features and functionality only**. If all required features and functionality can not be fit into the *development period*, a new SRS must be created for the next Waterslide cycle iteration and added there instead.

7. **Upon *Document Review*, decide quickly if the sub-cycle must be repeated or be finalized**. If all parties agree on and clearly understand the final result of the SRS, it may be finalized and the SRS phase completed. Otherwise, the SRS sub-cycle must be repeated.

Note: references above to "initial SRS" refer to the SRS which is drafted on the very first Waterslide life cycle iteration. Therefore, "subsequent SRS" refer to SRS documents for other Waterslide iterations which may follow.

To explain the SRS phase further, the general goal is to have a well-defined specification which represents the results which are to be achieved within the *development period*. When starting a new project, the very first Waterslide life cycle will attempt to build the core system with as many of the essential functionality that can be fit in as possible within the defined *development period*. It does not have to be perfect but must represent the generally desired outcome of the final result. This is because the stakeholder(s) may not always know exactly what they want at this stage yet. Once this first cycle is complete, the entire process is repeated. However, on subsequent iterations the stakeholders and developers must get together and discuss how to improve upon the initial cycle. Therefore, for each Waterslide cycle iteration, a new SRS is required. However, each new SRS will also consist of a complete picture of the whole product consisting of all changes that have been decided after the previous iteration. So basically, each new SRS is simply an updated version of the previous SRS which came before it.

3.2.2. Rapid Development

The Rapid Development phase is extremely simple. During this stage, the developers and designers will work together to create the product defined within the SRS document and then release the results to the stakeholder(s) accordingly. This phase is broken down into the following steps:

1. Create a Milestone Roadmap

2. Develop each milestone and release to stakeholder(s)

The Milestone Roadmap can be created collaboratively by breaking down the SRS into logical segments. In some cases, this Roadmap may be optional if dealing with an extremely small Waterslide cycle. In such situations, the Milestone Roadmap may not be necessary at all. However, if we take a Waterslide cycle that consists of a 30 day *development period,* for example, it would be beneficial for the stakeholder(s) to see the progress that is being made periodically instead of waiting a whole 30 days. Therefore, once each milestone is completed, the developer can release the results.

The best part about this phase is that it can progress simultaneously along with the *Thorough Testing* phase as soon as the first milestone is released. In some cases, the developer(s) may even make quick changes to fix minor issues simultaneously. However, ideally, all fixes should be made during the *Verified Fixes* phase once the *Rapid Development* phase has been completed.

3.2.3. Thorough Testing

The *Thorough Testing* phase is extremely important. During this phase, the stakeholder(s) will be responsible for testing the system thoroughly to ensure that it looks and behaves exactly as defined in the SRS. If any inconsistencies are discovered, a report must be logged for the developer(s) to review. This phase consists of the following policies:

1. **Strict quality assurance verification by comparing SRS with the end result.** The stakeholder is primarily responsible for this task. However, it must be done collaboratively among all involved parties.

2. **Thorough testing to ensure the end result looks and behaves exactly as described in the SRS**. The stakeholder is primarily responsible for this task. However, it must be done collaboratively among all involved parties.

3. **Log detailed reports of inconsistencies and bugs**. The individual making the report is responsible for clearly describing the problem, how it was encountered, if steps need to be taken in order to replicate the problem (and if so what they are), and referencing specific areas of the SRS which are relevant to the issue.

4. **New features and changes may not be requested during the testing phase.** If changes are required, they can be added to the next Waterslide cycle during the SRS phase. They may not be logged as fix reports during this phase.

5. **A collaborative effort must be taken to discuss and resolve the issues at hand**.

If the *Rapid Development* phase has been completed, the *Verified Fixes* phase can commence even if the *Thorough Testing* phase is still ongoing. Being able to process both phases simultaneously will save a lot of time. However, it should be noteworthy that there is indeed is a risk of collisions between multiple reports made during the *Thorough Testing* phase. Therefore, clear communication, collaboration, and care must be taken in order to avoid any potential wastage of time.

3.2.4. Verified Fixes

During this phase, the developer(s) will work collaboratively with the stakeholder(s) to make the required fixes towards the end result. It would be similar to the *Rapid Development* phase with the exception of stakeholders being able to offer feedback and final verification. This phase consists of the following sub-cycle:

1. **The developer(s) will analyze a logged report by order of priority (or submission).**

2. **The developer(s) will respond to a faulty report with evidence (or) begin the fixing process.**

3. **The stakeholder(s) will verify the response and finalize the task if satisfied with the outcome (or) submit a new report with additional information.**

4. **Repeat.**

Once all logged reports have been processed and verified, the *Solid Launch* phase can commence.

3.2.5. Solid Launch

This phase consists of taking the final results and placing them in a production

environment for active production use (or further testing such as a BETA instance).

3.2.6. Repeat Cycle

The Waterslide cycle may now repeat to address new requirements. These new requirements will always be based on the existing results of the process. Any drastic changes (such as requirements that may demand a complete reworking of the results) will constitute a new forked project.

3.3. Environment

The Waterslide method requires three environments to function as intended. They are:

1. **Development Environment**

   The *Development Environment* is used during the *Rapid Development* phase (and *Verified Fixes* phase) by the developer(s). Typically only the developer(s) would have access to this area. This could be the personal computer of a developer for example (but may not always be the case).

2. **Staging Environment**

   The *Staging Environment* is used during milestone releases. The *Thorough Testing* phase will be conducted in this environment, and fixes made during the *Verified Fixes* phase will also be released to this environment. The developer(s) will move results from the *Development Environment* to the *Staging Environment* where stakeholders may interact with the results.

3. **Production Environment**

   The *Production Environment* is the final destination for the results once the *Stable Launch* phase is initiated.

3.4. Document Format Specification

This document adheres to the VIRXTDOC-11.112 documenting standard. The VIRXTDOC-11.112 format is a simplified technical documenting scheme developed by VirX.Net Software Innovations Inc. It has been designed to help simplify complicated technical specifications so that they can be easily understood by both non-technical (non-tech savvy) stakeholders and developers alike.

3.4.1. Philosophy

The purpose of any technical document is to communicate, record and exchange information among associated parties to achieve (or maintain) a predefined goal. In order to achieve its purpose efficiently, the document must be clear, structured and easy to understand by all parties involved. Especially by individuals who are non-technical. This can be achieved by compiling technical information in a fluid story-like format which describes processes step-by-step. Descriptive explanations of practical real world use cases bring abstract specifications and theoretical data to life. Attributes and characteristics may be outlined in multilevel ordered lists. A simple, easy to replicate format is fundamental.

3.4.2. Structure

*Must contain a global header with a unique/descriptive title and footer with page info.

1. Cover page (header and footer optional)
    1. Document Title (must match header title)
    2. Document Author
    3. Document Date
    4. Other Details (any)
2. Table of Contents
3. Revision History

4. Introduction Section
    1. Document Format: a section to educate the reader about VIRXTDOC-11.112
    2. Purpose: purpose of the document and it's aspired goal(s)
    3. Scope: description of the subject matter and scope of the document
    4. Definitions: brief list of important word definitions
5. Overall Description: detailed description of subject matter hierarchically divided by numbered headers (and correctly referenced via the table of contents).
6. Other Sections (as required)
7. Document Format Specification: a copy of the document format specification you are reading right now.

3.4.3. Content Style Guidelines

1. Explain technical information in a fluid story-like format to elaborate abstract specifications and theoretical data.
2. Describe processes, functions and procedures step-by-step using real world practical use cases and examples.
3. Outline attributes and characteristics using multi-level ordered lists.
4. Maintain clear and consistent hierarchal organization of content
5. Keep it simple. Do not unnecessarily over-complicate.
6. Avoid generalizations and abstruse descriptions/specifications.
7. Sequential progressions (i.e. data that belong in flow-charts) should be represented in written format first and foremost.
8. Written explanations are compulsory. Diagrams and illustrations are secondary.