**Project: NASA Path**

**Software Development Plan (SDP)**

University of Maryland University College

SWEN 670 2178

Team: André Akinyele, Jeffery Dovan, Erika Guthrie, Tracy Hill, Pyae Kyaw, & Derek Stine

**Table of Contents**

[**Revision History**](#_gjdgxs) **3**

[**Introduction and Overview**](#_30j0zll) **4**

[Purpose](#_1fob9te) 4

[Scope](#_3znysh7) 4

[**Objectives**](#_2et92p0) **4**

[Determining Requirements](#_tyjcwt) 4

[Measuring Accomplishments](#_3dy6vkm) 4

[**Statement of Work (SOW) for NASA Path**](#_4d34og8) **5**

[**Assumptions and Constraints**](#_2s8eyo1) **5**

[Assumptions](#_17dp8vu) 5

[Constraints](#_3rdcrjn) 5

[**Work Breakdown Structure**](#_26in1rg) **7**

[**Software Project Description**](#_lnxbz9) **7**

[Principal Needs](#_35nkun2) 8

[**Software Requirements Specifications (SRS) Overview**](#_1ksv4uv) **8**

[Software Functionality](#_44sinio) 8

[Hardware/Software Specification](#_2jxsxqh) 9

[Minimum Hardware Specification](#_z337ya) 9

[Minimum Software Specification](#_3j2qqm3) 9

[Performance Specification](#_1y810tw) 9

[User Interface Requirements](#_4i7ojhp) 9

[Objects](#_2xcytpi) 10

[ISS Model](#_1ci93xb) 10

[**Software Development Process**](#_3whwml4) **13**

[**Software development & Implementation**](#_2bn6wsx) **13**

[Programming Language – JavaScript/Java](#_qsh70q) 13

[Development Environment – Docker Development Environment](#_3as4poj) 13

[Verification & Validation](#_1pxezwc) 13

[Codebase Control](#_49x2ik5) 14

[**Software Development Approaches**](#_2p2csry) **14**

[**Software Development Method: Waterfall Methodology**](#_147n2zr) **15**

[Waterfall Methodology Process](#_3o7alnk) 15

[Waterfall Phases](#_23ckvvd) 16

[**Test Plan**](#_ihv636) **16**

[**Maintenance & Documentation Plan**](#_32hioqz) **17**

[**Deliverables**](#_1hmsyys) **17**

[**References**](#_41mghml) **17**

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Author | Date | Description |
| 1.0 | André Akinyele | 9/22/2017 | Created document |
| 2.0 | André Akinyele | 9/23/2017 | Added information about path color |
| 3.0 | André Akinyele | 9/25/2017 | Edited specification information |
| 4.0 | André Akinyele | 9/29/2017 | Fixed document format |
| 5.0 | Erika Guthrie | 9/30/2017 | Created headers, table of contents, and added revision history table |
| 6.0 | Jeffrey Dovan | 10/1/2017 | Fixed document format |
| 7.0 | Derek Stine | 10/22/2017 | Updated after milestone 1 and client feedback |
| 8.0 | André Akinyele | 11/8/2017 | Updated and fixed document format after milestone 2 and feedback |
| 9.0 | Derek Stine | 11/12/2017 | Added new interface screenshots. Highlighted changes for milestone 3 |

**Software Development Plan**

# Introduction and Overview

## Purpose

The purpose of this software development plan (SDP) is to design a comprehensive strategy that is required in developing, modifying, or upgrading the NASA Path software program. Also, this SDP provides aspects of each step and its required deliverables. Furthermore, the SDP covers every aspect from the project’s process of software development implementation, documentation, through the software testing lifecycle and deployment.

## Scope

The scope of the NASA Path project will include an algorithm that finds the shortest exterior paths of the International Space Station (ISS) using a 3D model. It will also include navigation mapping for the exterior of the ISS using handrails and structural beams that states which path is the shortest distance, which path encounters the least hazards, and which path requires longer arm reach. It will include a design similar to Google Maps with navigation.

# Objectives

## Determining Requirements

* Analyzing the software requirements and interface software architecture
* Decomposing the software requirements and interface software architecture
* Determining requirements and deliverables
* Forming development teams based on skill set and resource availability
* Employing communications management via weekly meetings

## Measuring Accomplishments

The software program is to be completed in 4 Milestones:

* Milestone 1: Project Plan and PowerPoint Presentation
* Milestone 2: Software Requirements Specification, Test Plan, and Pseudocode
* Milestone 3: Prototype (Demo) and Testing
* Milestone 4: Regression, Quality Assurance, Signoff, PowerPoint Presentation, and Final Application Delivery

# Statement of Work (SOW) for NASA Path

The final project program is to inform the client as to which path is the shortest distance, which path encounters the least hazards, and which path requires longer arm reach when a crewmember is at a particular location and needs to get to another place on the other end of ISS. The program is to consist of:

a. Creating an algorithm that finds shortest paths for the exterior of the International Space Station using a 3D model. The program is to utilize the Dijkstra Algorithm, as it   
is used to find the shortest path between two points in a graph and could be modified   
for 3D.

b. Creating maps and navigation for the exterior of the ISS (using handrails and   
structural beams).

c. Design something like Google Maps with navigation. The Client has a 3D model of the exterior of the Space Station, complete with handrails and other interfaces useful to a space walker. Completed in 4 Milestones, the deliverables are due:

* Milestone 1 delivered on October 1, 2017
* Milestone 2 delivered on October 22, 2017
* Milestone 3 delivered on November 12, 2017
* Milestone 4 delivered on December 3, 2017

# Assumptions and Constraints

## Assumptions

The development can be web-based. There are no software language or technology preferences other than those that provide for open-source development and future modifications. The current “DOUG” program provides client with a 3D model, which allows highlighting handrail paths and hardware, and a “ruler” feature to estimate distances between two points (sample data is shared on GitHub repo at <https://github.com/darenwelsh/EVANav>). If team formats the data as shown, “DOUG” developers can import it into the program but the core requirement is to provide a 3D representation of the calculated paths for users, either in “DOUG” or in the software. The team can estimate any path that requires a “plane change” by providing more time (explained later).

## Constraints

The constraints are that, if the software developed is web-based, the software must support more than one browser (Chrome, Firefox, and Internet Explorer (IE)) because “DOUG” program is an executable application that runs natively on the user’s computer (PC, OS X, and Linux supported, with PC as the largest customer). The chosen software and/or technology must foster long-term support from collaborative development, although there are no software language or technology preferences. The client does not want a “black box” delivery, as the client will need to fix bugs and add new features.

The user input interface must be clear and simple. Also, the user input interface should include start and end points, allow optional waypoints in the middle of the route, and allow options for route determination. This includes avoiding crew hazards (things that hurt the crew or their suit), hardware hazards (things that could be hurt by the crew), minimize rotations and plane changes, and field to enter crew member’s wingspan. Eventually, the client would like to support two simultaneous routes (one for each crew member) and deconflicting a partner’s route (i.e., don’t use the same handrails) but those are not core requirements for this project.

The output interface should be as transparent as possible. Output should include a visual representation of the available paths. Eventually, a sequential list of handrail numbers should be output, including distance between each pair of handrails, and be in a format readable by “DOUG” but that is not a core requirement for this project.

The format of the objects for path calculation should be represented using nodes on a graph (a scene-graph), illustrated below in *Fig. 1*, with each node in the graph containing a transformation of the frame that its model and child nodes would be relative to. Each node is presented with 5 consecutive lines of text in the file with the following format:

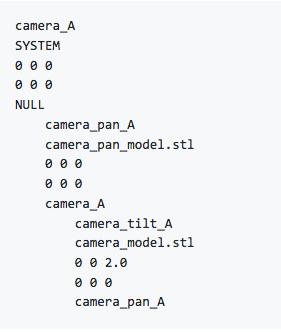
<unique\_node\_name>

<geometry\_file\_name> or "SYSTEM"

<x> <y> <z>

<pitch> <yaw> <roll>

<parent\_node\_name> or "NULL"

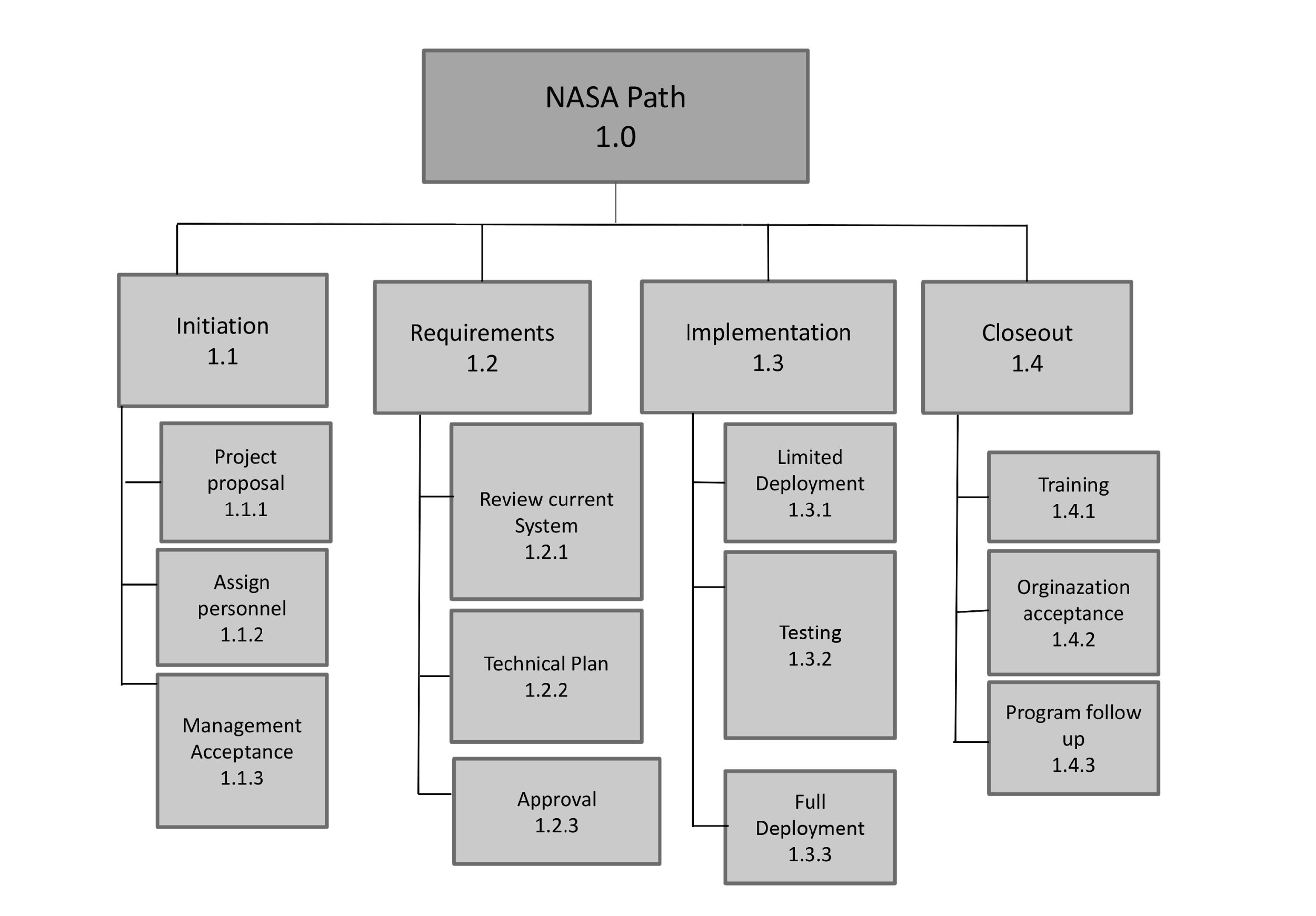


*Fig. 1*: Example of using nodes on a graph (a scene-graph).

In “DOUG”, all units are in inches and degrees. Also, “DOUG” uses a right-handed coordinate system where rotation is about the ‘Y-axis’, ‘Z-axis’, and ‘X-axis’, typically applied in Pitch-Yaw-Roll order. The software should operate in formats similarly to “DOUG” to provide for better compatibility.

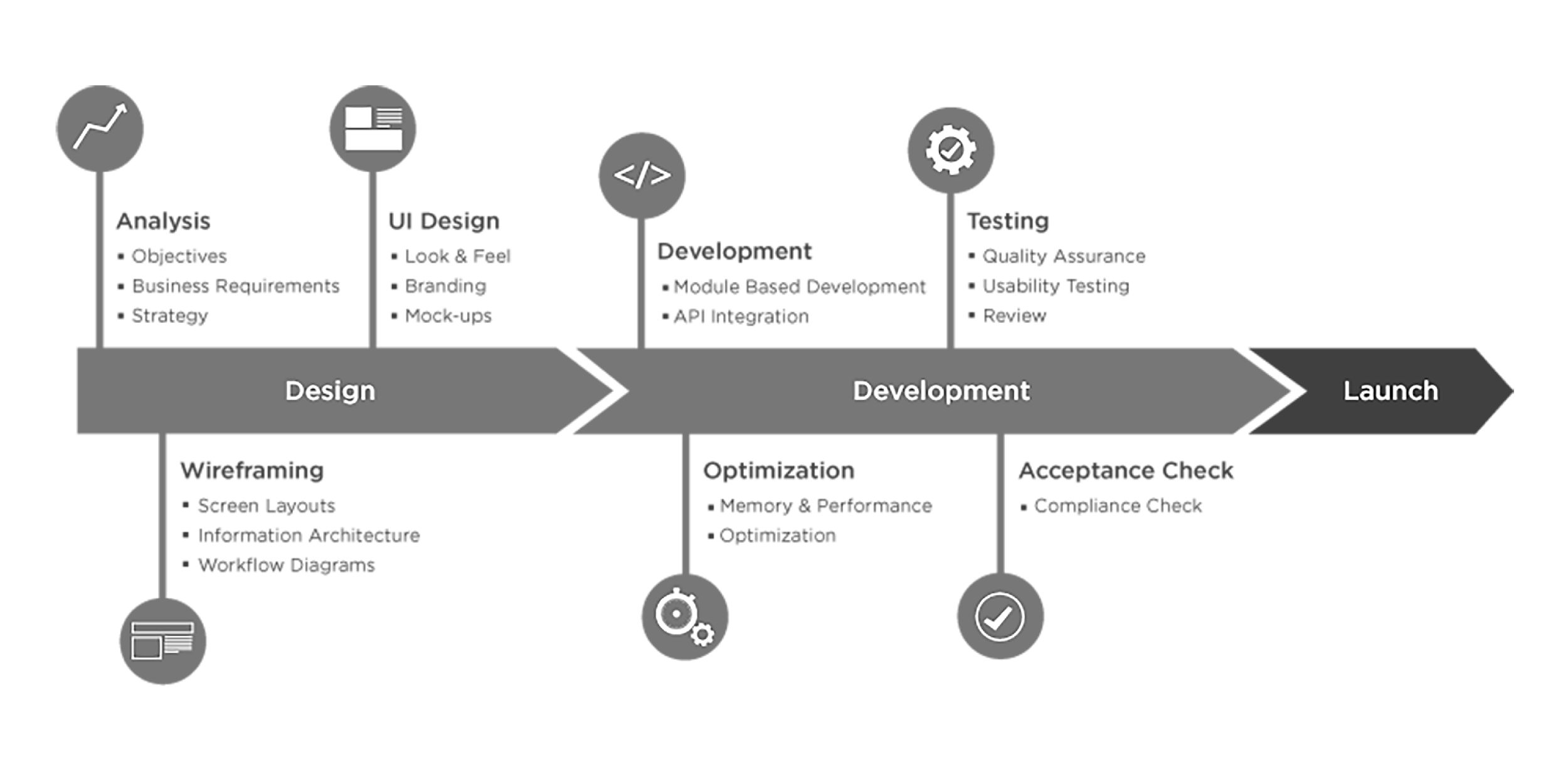
Additional constraints include designing the software similar to Google Maps with Navigation using a 3D Model; estimating any path that requires a “plane change” will require more time, meaning when the body rotates from one translation plane to another (going around a corner or edge). If any of the model names include the word “antenna”, it has some volume around it designated as a “Keep Out Zone.” For example – one antenna might have high levels of radiation in a zone up to 1 foot around it.

# Work Breakdown Structure



*Fig. 2*: Work Breakdown Structure.

# Software Project Description



*Fig. 3*: NASA Path design process.

## Principal Needs

Users of the software are astronauts and the people who train them and write their spacewalk procedures. The software will be used to train astronauts on how to perform spacewalks, plan astronaut’s spacewalks, and write astronauts procedures. Also, the software will be used to map out translation paths and choreograph maneuvers involving robotic arm (SSRMS). The software will also be used to avoid hazards, such as radiating antennas and sharp edges. Finally, the software will be used real-time when deviations to the timeline must be made, whether due   
to contingencies or timeline constraints. The software data input takes input from the user to establish the ideal path(s) for the astronaut. The software data output displays output data of the determined ideal path(s) for the astronaut, then outputs the data so that “DOUG” can import it. Finally, the software data output displays with the application.

# Software Requirements Specifications (SRS) Overview

## Software Functionality

The main functionalities of the software include an algorithm that finds the shortest path for the exterior of the ISS using a 3D model. Also, a 3D model that states which path is the shortest distance, which path encounters the least hazards, and which path requires longer arm reach. Finally, a display of multiple paths between destinations; i.e., the first path displays the shortest path, the second path displays the path with the fewest hazards, and the third path displays the fastest path.

## Hardware/Software Specification

The architecture hardware consists of client’s PCs and/or devices and servers. The software consists of Web browsers, such as Chrome, Firefox, and IE. The network consists of TCP/IP with five layers:

* Application
* Host-to-Host or Transport
* Internet
* Network Access
* Physical layers

## Minimum Hardware Specification

* Monitor, keyboard and mouse
* Either Internet connection or data connection to upload/download files
* Need at least 3.4 GHz central processing unit (CPU), at least 8GB random access memory (RAM), and at least 1GB graphics processing unit (GPU) for smooth operation of 3D software
* 10 GB free disk space

## Minimum Software Specification

* Chrome/Firefox/IE 11 browser
* Windows 7+/Mac OS X+/Ubuntu 14+

## Performance Specification

Software performance includes using a 3D Model that tells the client, in 3D, where a crewmember is at a particular location, how to navigate a crewmember when that member needs to get to another place on the other end of ISS, which path is the shortest distance, path encounters the least hazards, and path requires longer arm reach.

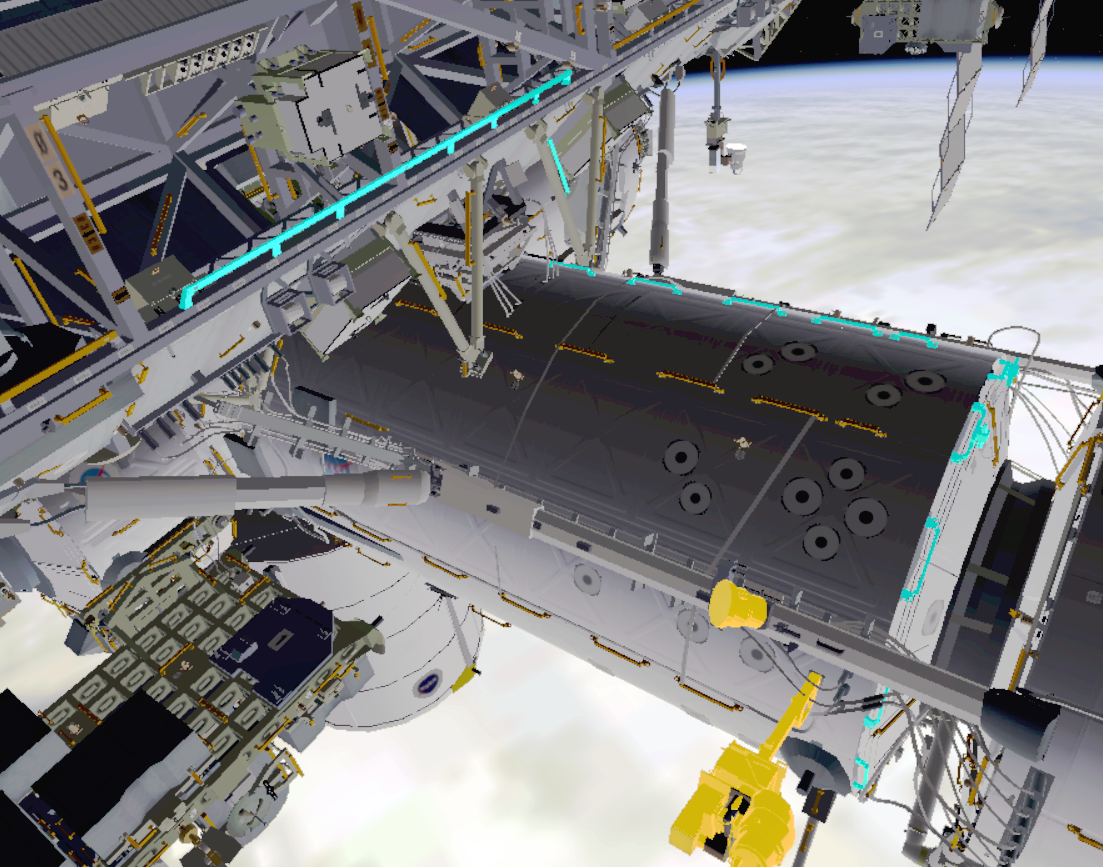
## User Interface Requirements

The software interface input should be simple, using a form with some pull down and slider fields. Variables should start with (1) Start handrail, (2) End handrail, (3) Optional middle waypoint handrails, (4) Representation of crew hazards and sensitive hardware to avoid as well as traversing around obstacles rather than through them, and (5) Integer value for astronaut’s wingspan (the distance this particular person can reach between handrails). The software interface output is provided in the GitHub repo. If data is formatted as shown, the “DOUG” developers can import it into the “DOUG” application for 3D viewing. The team should consider providing some preliminary sample output data to the “DOUG” developers to ensure requirements are met. However, realizing that compatibility with “DOUG” is not a core requirement, client would like to leave room for import/export compatibility with “DOUG” to be developed in the future. At the least, output should include a visual model of the calculated paths in the interface within the delivered software.

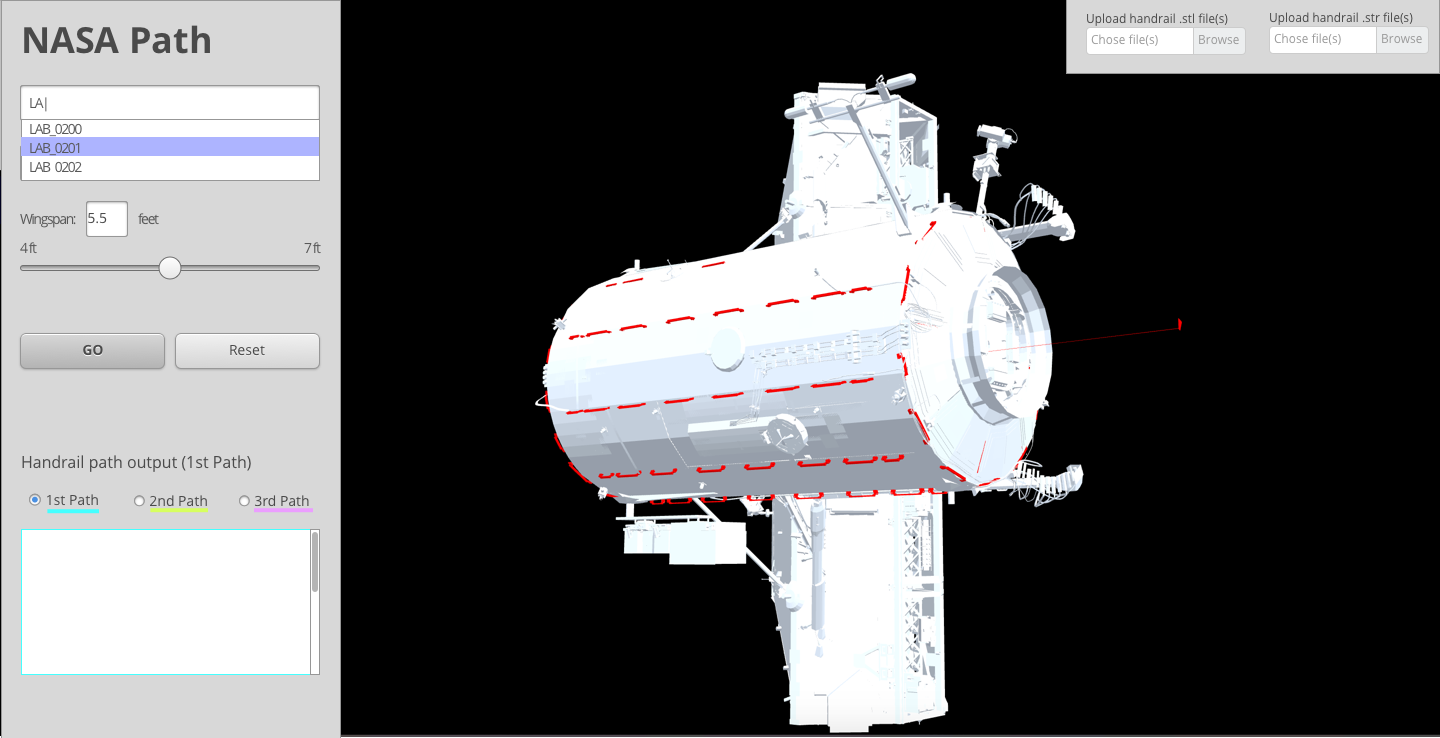
## Objects

The Model Objects include the handrails (all the yellow/gold handrails are to be used), arm reach (user selects from a slider that ranges from 4 ft. to 7ft.), and hazards. Hazards are to be represented in the software by the development team; i.e., make a list of some objects that protrude from the structural surface to showcase the functionality. If any of the model names include the word “antenna”, it has some volume around it designated as a “keep Out Zone.”

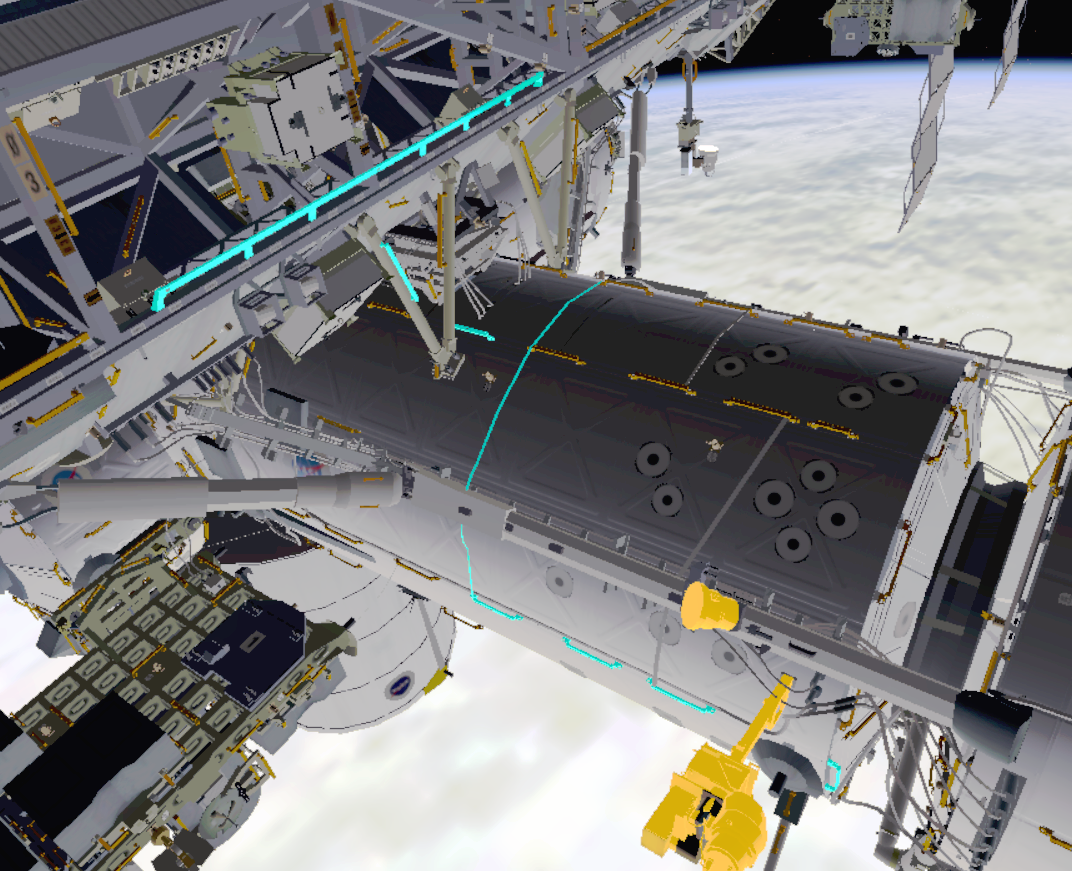
## ISS Model



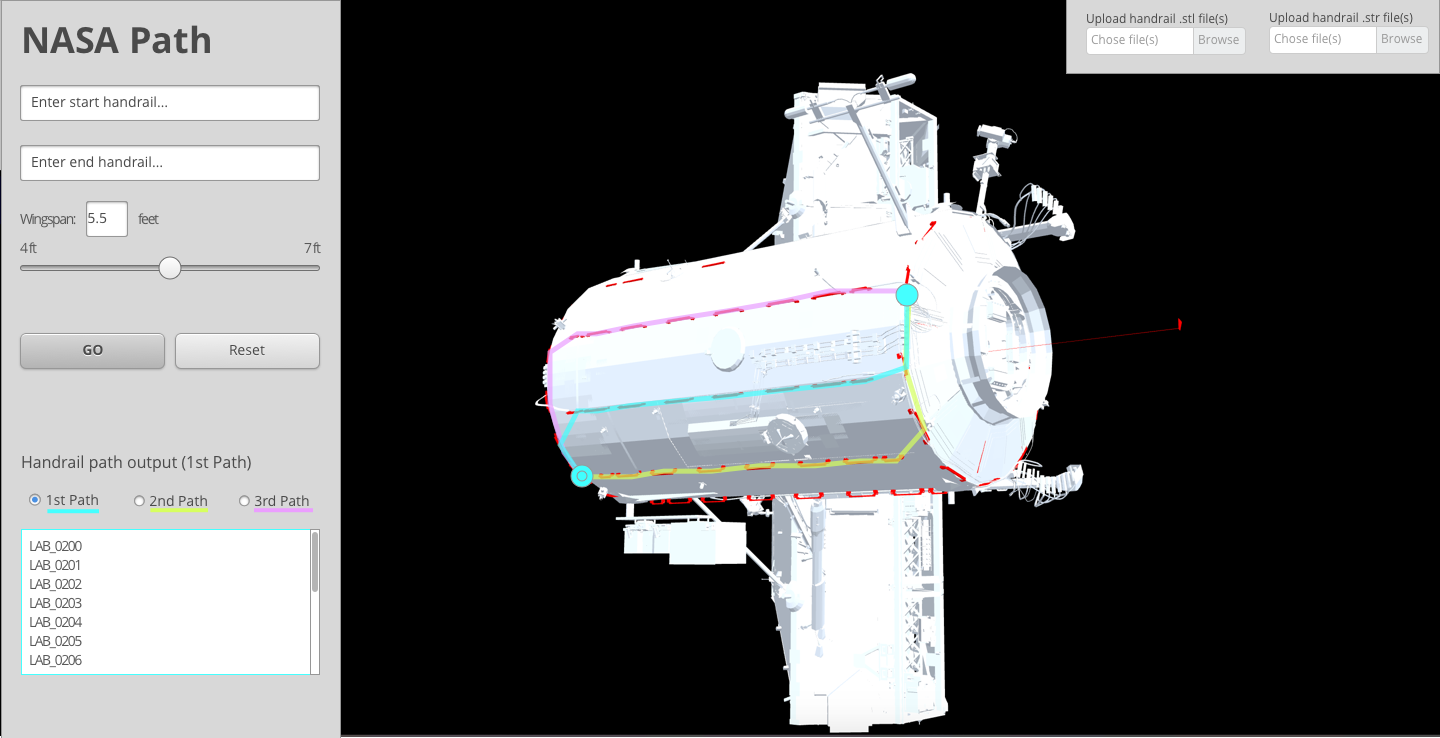
*Fig. 4*: ISS path 1. Crewmember must maneuver around sensitive hardware highlighted in yellow.



*Fig. 4.5*: ISS path 1. Handrails in NASA Path application model highlighted in red.



*Fig. 5*: ISS path 2. The crewmember routed away from the sensitive hardware, but the path is longer.



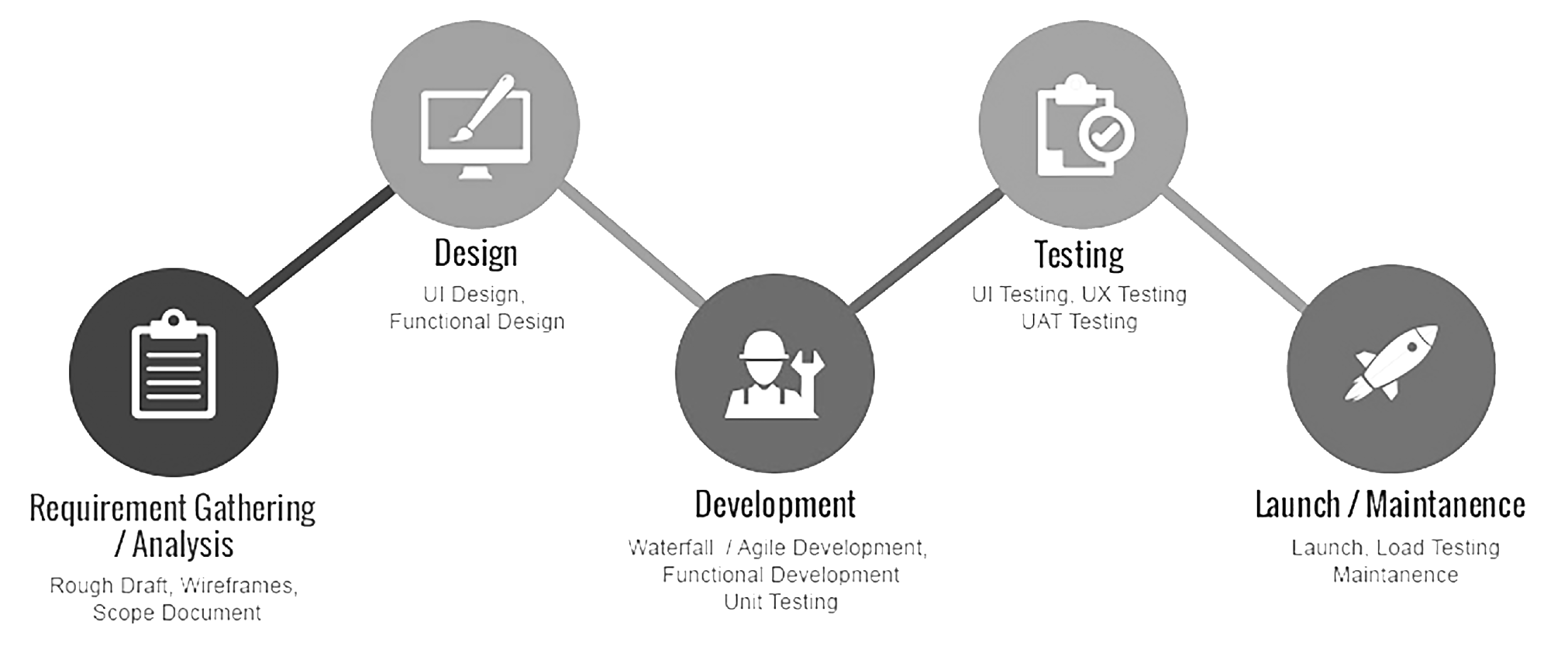
*Fig. 5.5*: ISS path 2. Top three paths calculated highlighted in 1) cyan, 2) magenta and 3) yellow.

The ISS 3D model in *figs. 4–5* were provided by the client from the GitHub repo at <https://github.com/darenwelsh/EVANav>.

The 3D model comes from a program named “DOUG”. Unfortunately, the client can’t share the program, but can be the liaise between the team and developers of the “DOUG” program. Due to these limitations, the NASA Path application will provide a browser-based 3D rendering (using the ThreeJS JavaScript library) for depicting the handrails and paths provided shown in *fig. 4.5* and *fig 5.5* respectively. From this model, Path 1, shown in *fig. 4* above, illustrates how the crewmember must maneuver around the sensitive hardware highlighted in yellow and the paths highlighted in cyan.

From the 3D model program named “DOUG”, Path 2, shown in *fig. 5* above, illustrates how the crewmember is routed away from the sensitive hardware, but the path is longer, highlighted in cyan.

# Software Development Process



*Fig. 6*: NASA Path development process.

# Software development & Implementation

## Programming Language – JavaScript/Java

The JavaScript and Java programming languages will be utilized to implement a front-end and back-end, also using an API or services architecture to communicate between the two.

## Development Environment – Docker Development Environment

The software environment will consist of Docker, a software container platform, where developers use the environment to eliminate problems when collaborating on code (Docker, Inc., 2017). The environment allows developers to use the Docker repository and its Dockerfile to create a Docker image, run a Docker container, and develop code in the container. Docker builds, tests, and releases new Docker versions using this container. On each software commit, a new Docker container will be built and tests will be executed. Only when the tests are successful, the new changes will be deployed to the stable release.

## Verification & Validation

Software verification will ensure that the output of each phase of the software development lifecycle has effectively carried out what the corresponding input artifact specifies (requirement > design > software product). Software validation will ensure that the software meets the needs of the client. Thus, software verification will ensure that the team “built it right” and software validation will ensure that the team ”built the right software" and confirm that the software, as provided, fulfilled the intended use and goals of the client.

Verification & Validation for this project will include a review with the client at each milestone to ensure that the software is developed according to expectations. A test plan for the project will be submitted as part of Milestone 2. Milestone 3 will include a demonstration of the working software. Additionally, the client will have access to the Github demonstration link to review and provide feedback at any time throughout development. Adjustments based on feedback can be made accordingly before final delivery provided they are received up to 2 weeks prior to Milestone 4 and fit within the agreed upon scope of the project. Additionally, two weeks of testing by the development team will be concluded prior to handoff to the client in Milestone 4 to ensure the software meets the stated requirements before delivery.

## Codebase Control

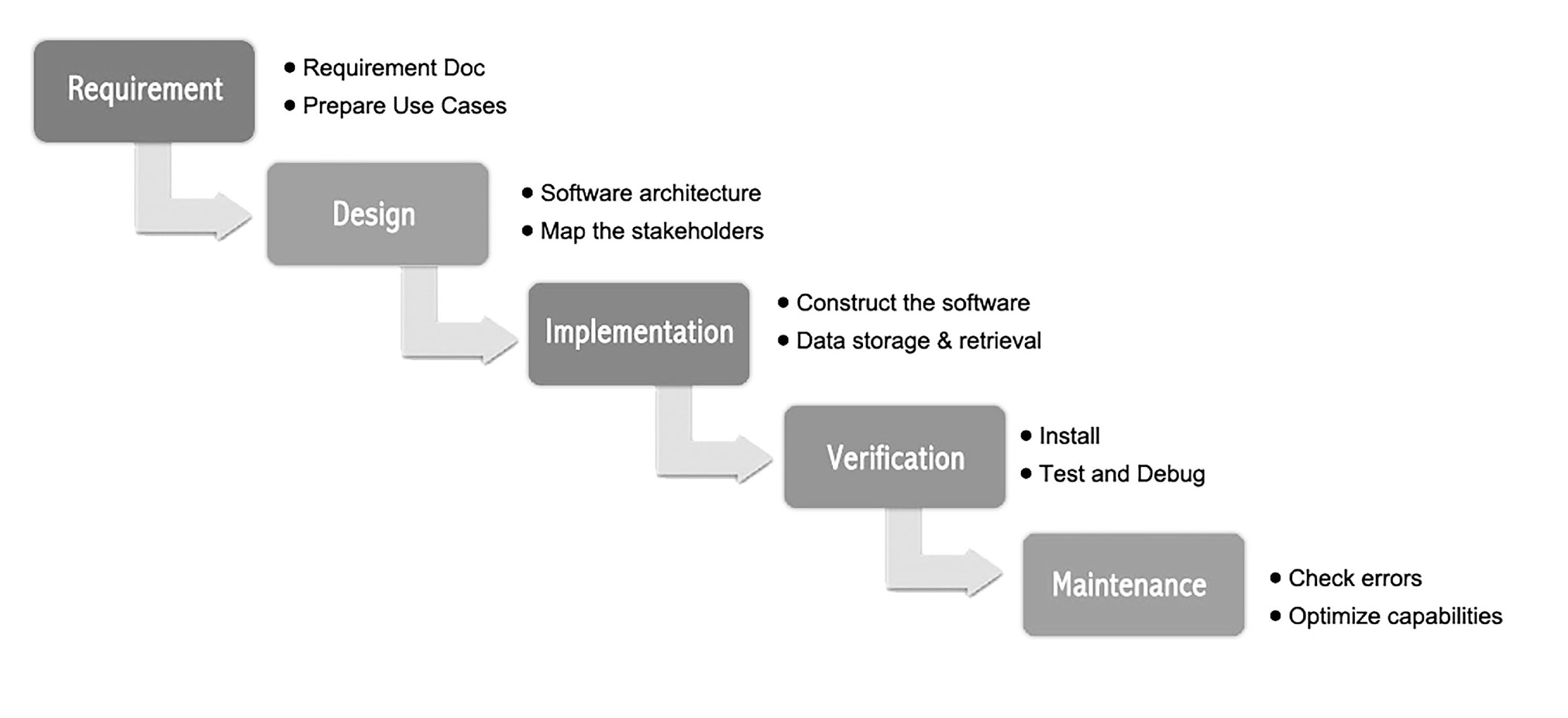
The team will utilize the Git codebase version control since the client already has some experience with the system. As the client wants the software to be released under a free and open source license to continue development and maintenance, we will be housing the software on GitHub, which is an online platform for Git repositories, designed for small to large projects. Also, Git offers local branching, staging areas, and multiple workflows.

# Software Development Approaches

The software architecture will be Web-based, used over the Internet with a Web browser. The client side scripting/coding will consist of JavaScript/Java and the Dijkstra Algorithm (modified for 3D). The server side will consist of Docker Development Environment and Git.

Dijkstra’s algorithm is an analogous scheme to compute a shortest paths tree (SPT). The algorithm solves the single-source shortest-paths problem in edge-weighted digraphs with nonnegative weights. Dijkstra’s algorithm uses extra space proportional to V and time proportional to E log V (in worst case) to solve the single-source shortest paths problem in an edge-weighted digraph with E edges and V vertices. The algorithm builds a rooted tree by adding an edge to a growing tree; thus, adding next the non-tree vertex that is closest to the source (Sedgewick & Wayne, 2011). All-pairs shortest paths: Given an edge-weighted digraph, support queries of the form given a source vertex s and a target vertex t. This finds the shortest path from s to t with one whose total weight is minimal, using time and space proportional to E V log V. It builds an array of DijkstraSP objects, one for each vertex as the source. Thus, it answers a client query, uses the source to access the corresponding single-source shortest-paths object and then passes the target as argument to the query.

# Software Development Method: Waterfall Methodology



*Fig. 8*: Waterfall Methodology

The Waterfall methodology is one of the basic approaches applied to software development methodology frameworks, as it’s a linear sequential framework and focuses on complete and correct planning to guide large projects and risks to successful and predictable results. The methodology involves the following development steps: Conception, Initiation, Requirements (defined up front and detailed), Design (text and diagrammatic descriptions of the software and hardware elements), Implementation (system programming), Verification (integration and testing components), and Maintenance (maintaining the system).

## Waterfall Methodology Process

The Waterfall methodology is a sequential approach, in which development is seen as flowing steadily downwards through its phases, as it shows a process, where developers follow these phases in order, such as the requirements specification (requirements analysis), software design, implementation and integration, testing (or validation), deployment (or installation), and maintenance, as demonstrated above in *fig. 8*. The simple principles involve projects that are divided into sequential phases, with some overlapping between phases, as emphasis is on planning, time schedules, target dates, budgets, and implementation of a complete system at once. Thus, after each phase has ended, it proceeds to the next phase.

Although the Waterfall methodology discourages revisiting and revising any prior phase once it’s complete, reviews may occur before moving to the next phase, thus, allowing for possible changes or to ensure a phase has indeed completed. However, iteration is not generally part of the Waterfall methodology, but can usually occur at the testing phase.

Therefore, the Waterfall methodology process offers tight control, maintained over the lifecycle of the project, utilizing extensive documentation, formal reviews, and obtaining the approval/signoff by the user and IT management, which occurs at the end of each phase, but before the commencement of the next phase.

## Waterfall Phases

First Phase: Preliminary Analysis involves conducting a preliminary analysis, proposing alternative solutions, describing benefits, and then submitting a preliminary plan with recommendations.

Second Phase: Requirements defines the project objectives into well-defined functions and operation of the intended application, while analyzing end-user information needs.

Third Phase: Design describes the desired features and operations in detail, which includes screen layouts, business rules, process diagrams, pseudocode and other documentation.

Fourth Phase: Implementation, Integration, Testing or Validation implements the real code and brings all the phases together to form a special testing environment, which thereafter, checking for errors bugs and interoperability commence.

Fifth Phase: Deployment or Installation represents the final stage of the initial development, as the software is put into production and runs.

Finally, Sixth Phase: Maintenance involves modifications, improvements, add-ons, and the movement to different processing platforms. The maintenance phase is often the longest of the stages because maintenance takes place during the remainder of the software’s lifecycle (Modern Software Development Methodologies for Software Engineering, 2016).

# Test Plan

The testing strategy includes some of the most common testing needed for any Web application development process: Quality Assurance and Bug Testing (Unit and Functional), Multiple Browser Compatibility, Application Security, and Performance - Load and Stress Testing. The deployment strategy involves usability, as the software will be stored publicly on GitHub for use.

Both unit-testing and functional testing will be automated and executed on each developer’s code commit. Also, we will automate some laborious parts of user interface (UI) testing, but do manual testing for the rest of the UI to test user experience (UX).

# Maintenance & Documentation Plan

For continued development and maintenance, the software will be housed on GitHub (publicly) and released under a free, open source license. Documentation will consist of a README file.

# Deliverables

The team will use Git and host the software on GitHub (publicly) which will also serve as the final delivery format for the client to download. The software should be released under a free and open source license. It is currently hosted at <https://github.com/lovetostrike/nasa-path-finder> but should be downloaded and relocated to an official NASA repository after final delivery.

# References

Asahi Technologies (2016). *Waterfall methodology* [Image]. Retrieved from https://www.asahitechnologies.com/wp-content/uploads/2016/02/Waterfall.jpg

Docker, Inc. (2017). What is Docker? Retrieved from https://www.docker.com/what-docker

Duckett, J. (2014). *JavaScript & Jquery: Interactive front-end web development*. Indianapolis, IN: John Wiley & Sons, Inc.

Infoicon Technologies. (2016). *Development process* [Image]. Retrieved from http://www.infoicontechnologies.com/assets/images/mobile-app-new.jpg

Pelosi, M. (2016). Modern software methodologies for software engineering. Document posted in University of Maryland University College SWEN 603 9040 online classroom, archived at: http://campus.umuc.edu

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Upper Saddle River, NJ: Pearson Education, Inc.

Stallings, W., & Case, T. (2013). *Business Data Communications: Infrastructure, Networking, and Security* (7th ed.). Upper Saddle River, NJ: Pearson Education, Inc.

Vogel, L. (2009, 2016). Dijkstra’s shortest path algorithm in Java – Tutorial: Vogella GmbH version 1.2, 29.09.2016. Retrieved from http://www.vogella.com/tutorials/JavaAlgorithmsDijkstra/article.html

Welsh, D. (2017). Navigation for space walks. Retrieved from https://github.com/darenwelsh/EVANav

WisdmLabs. (2016). *Design process* [Image]. Retrieved from https://wisdmlabs.com/site/wp-content/uploads/2016/02/app-design-process.png