Shortqut

Concept of Operations

COP 4331 Fall 2010

Modification history:

Version	Date	Who	Comment
v0.0	08/15/00	G. H. Walton	Template
v1.0	9/19/10	Dana Orlando	Main information sections filled in
v1.1	9/22/10	Steven Nichols	Fix minor formatting issues

Team Name: Shortqut

Team Members:

- Robby Ronk Email Website
- Laura Barton Email Website
- Dana Orlando Email Website
- Eric Olson Email Website
- Steven Nichols Email Website
- Richie Manis Email Website

Contents of this Document

The Current System

The Proposed System

- Needs
- Users and Modes of Operation
- Operational Scenarios
- Operational Features
- Expected Impacts
- Analysis

The Current System

The standard GPS (Global Positioning System) navigation software uses space satellites to determine its current location. Those that are built for use inside a vehicle have the ability to take in additional location information, and determine a path between the two points. They are built for use on the road, and will therefore find the shortest route between the two locations based on the shortest distance. The information used to determine the route is stored in an internal database in the form of geographical points (latitude and longitude) and is used in coordination with a map to display the route on the GPS monitor, and using text-to-speech, verbally tell the driver where the next point of interest is, and how soon they will reach it.

The Proposed System: Needs

What these systems do not supply is information on whether a particular road on their route gets congested during the heavy traffic times (morning, and afternoon rush hours).Â This system will be designed to calculate the route to the destination based not only on the locations, but also on the current time of day, and the traffic patterns typically experienced at that time. For example, if a road would normally become congested during the late afternoon hours, our system would find an alternate route for the user to take, so to avoid having to sit in that traffic. The system will also feature a detour routing system. This operation will allow the user to request an alternate route to their destination if the one that is supplied by the system is not sufficient for them. The system will then calculate and return the next shortest route.

The Proposed System: Users and Modes of Operation

User with little knowledge of the area: These users will be using the basic mode of operation. They will put in their destination, and the system will calculate the shortest route for them to follow. Since they are not familiar with the area, it is unlikely that they will want to change the route that they are currently on.

User knowledgeable about the area: These users will most likely be using the detour option supplied with the system. If the user does not like the shortest route that is calculated, they can request that another route be calculated, and will then be presented with the second shortest route, and so on. For example, if the user would like to avoid a particular road, or would like to go around a particular part of town instead of through it, that information can be taken into account when calculating the route.

It is assumed that all users of the system will have knowledge about driving.

The Proposed System: Operational Scenarios

Scenario 1:
The user is using the GPS navigation device during a time block when the roads are typically busy. After the user inputs the destination, the GPS will search for the shortest route to get the user from the current location to the destination. If any of the roads along that route are typically busy during the time block that the user is due to arrive at that road, the GPS will find another route around the congestion, but that will also have the next shortest travel time.

Scenario 2:

The user is using the GPS navigation device during a time block when the roads are not typically congested. After the user inputs the destination, the GPS will search for the shortest route to get the user from the current location to the destination. Since the roads are not typically busy at that time, it will keep the user travelling along the calculated route until they reach their destination.

Scenario 3:
The user is using the GPS navigation device, and the road that they are currently traveling on is taking too long, perhaps due to an accident or construction. When the user inputs a destination, the GPS calculates the three fastest routes to that location. If the current route is insufficient, the user can choose a detour option. This option will choose another one of the routes that were calculated in the beginning of the trip, and redirect them from their current location.

The Proposed System: Operational Features

Must Have:

-Give accurate directions from one geographical location to another

-Find the shortest path using a combination of shortest distance and shortest time

-Use the current time, and the estimated time for each road to determine which roads will take the least amount of time

-Calculate the three shortest routes

-Turn by turn directions

-Visual display of the map, and the highlighted route

-Traffic patterns, intersections, and roads stored in local and central databases

-Zoom and pan map

-Center map around current location

-Text-to-speech

Would Like to Have:

-Rotating map

-Ability to avoid certain roads (such as highways)

The Proposed System: Expected Impacts

We feel that this system is a great improvement over traditional GPS systems, and will be both useful and practical for all drivers, whether they drive during rush hour, or during times when traffic is not as heavy. We hope that it will not only lead to faster travel times, but will eventually be able to improve traffic congestion on some of the major roads.
The development of this project will also have an impact on the developers. It will give us more experience in dealing with a real world issue, and in being part of a development team that will help to improve that issue.

The Proposed System: Analysis

Expected Improvements: <list them>

Disadvantages:

-System needs to "learn" traffic patterns

-If the system is taken to a place it has never been used before, it will need to learn the traffic patterns before it can be used to avoid congested roads

Limitations:

-System will be programmed with information about traffic patterns around the UCF area, however the a map will be supplied for the entire state of Florida

-Normal GPS navigation system can be used to look up directions for the state of Florida, but the traffic data will only be supplied for the UCF area

Risks:

-High challenge level of the project

-Most team members have not used the decided programming language before, could cause some programming delays

Alternatives and Tradeoffs: <list them>

Template created by G. Walton (GWalton@mail.ucf.edu) on August 30, 1999 and last modified on August 15, 2000.

This page last modified by Dana Orlando(kittymage@knights.ucf.edu) on September 20, 2010

ShortQut

Project Management Plan

COP 4331C, Fall, 2010

Modification history:

Version	Date	Who	Comment
v0.0	08/15/00	G. H. Walton	Template
v1.0	09/16/10	Steven Nichols	Update header, add links in Applicable Standards section
v1.1	09/19/10	Steven Nichols	Updated Quality Assurance, Risk Management, Technical Progress Metrics, Plan for tracking, control, and reporting progress
v1.2	09/20/10	Steven Nichols	Updated Team Members, Added link to Artifact Size Metric
v1.3	09/20/10	Laura Barton	Updated Project Overview, Project Team Organization, and Deliverables
v1.4	09/21/10	Laura Barton	Updated Life Cycle Process, Table of Work Packages, etc., and Configuration Management
v1.5	09/21/10	Steven Nichols	Updated Tools and Computing Environment
v1.6	09/21/10	Laura Barton	Updated V model diagram, PERT Chart, and some links

Team Name: Team 1, Inc.

Team Members:

- Eric Olsen : EOlsen@knights.ucf.edu
- Steven Nichols : steven.nichols@knights.ucf.edu
- Robby Ronk - rjr@knights.ucf.edu
- Richie Manis - r@knights.ucf.edu
- Dana Orlando - kittymage@knights.ucf.edu
- Laura Barton - lbarton@knights.ucf.edu

Contents of this Document

Project Overview

Reference Documents

Applicable Standards

Project Team Organization

Deliverables

Software Life Cycle Process

Tools and Computing Environment

Configuration Management

Quality Assurance

Risk Management

Table of Work Packages, Time Estimates, and Assignments

PERT Chart

Technical Progress Metrics

Plan for tracking, control, and reporting of progress

Project Overview

Team 1, Inc. will implement a Global Positioning System, nicknamed ShortQut, which will have the ability to learn about and adapt to road congestion from collected data as well as suggest the quickest route given the provided data. ShortQut will be run via any portable computer connected with a USB GPS device and the database in which the archived data is stored. Unlike ordinary GPS devices, ShortQut will have stored documentation of traffic patterns from which it will be able to derive the optimal route.

Reference Documents

- Concept of Operations

Applicable Standards

- Coding Standards
- Document Standards
- Artifact Size Metric Standard

Project Team Organization

Members of this development group include Robby Ronk, Eric Olsen, Steven Nichols, Dana Orlando, Richie Manis, and Laura Barton. The developers communicate through weekly face-to-face meetings, group emails, a Trac site, and GitHub. Ronk leads the team in the role of project manager. He oversees the development of each module as determined by the group as well as maintains the group web site. Each group member is responsible for a different section of the project that, when completed, shall work together seamlessly. The following is a list of functional modules, their managers and a brief description of the responsibilities involved with that part:

- **Interpreter/Retriever:** Manis shall manage the communication between the GPS device and the ShortQut program.
- **Map Factory:** Olsen shall be responsible for the creation of the map and plotting of data
- **Route Factory:** The generation of the top 3 quickest routes using the A* algorithm shall be overseen by Nichols.
- **The Stiqy (i.e. GUI):** Orlando shall compile the user interface which will display the map and the suggested routes as well as use text-to-speech to convey the directions.
- **Database Communication:** Barton shall develop the database tables and procedures needed to pass data between both the local and server databases and the local database and ShortQut.

Deliverables

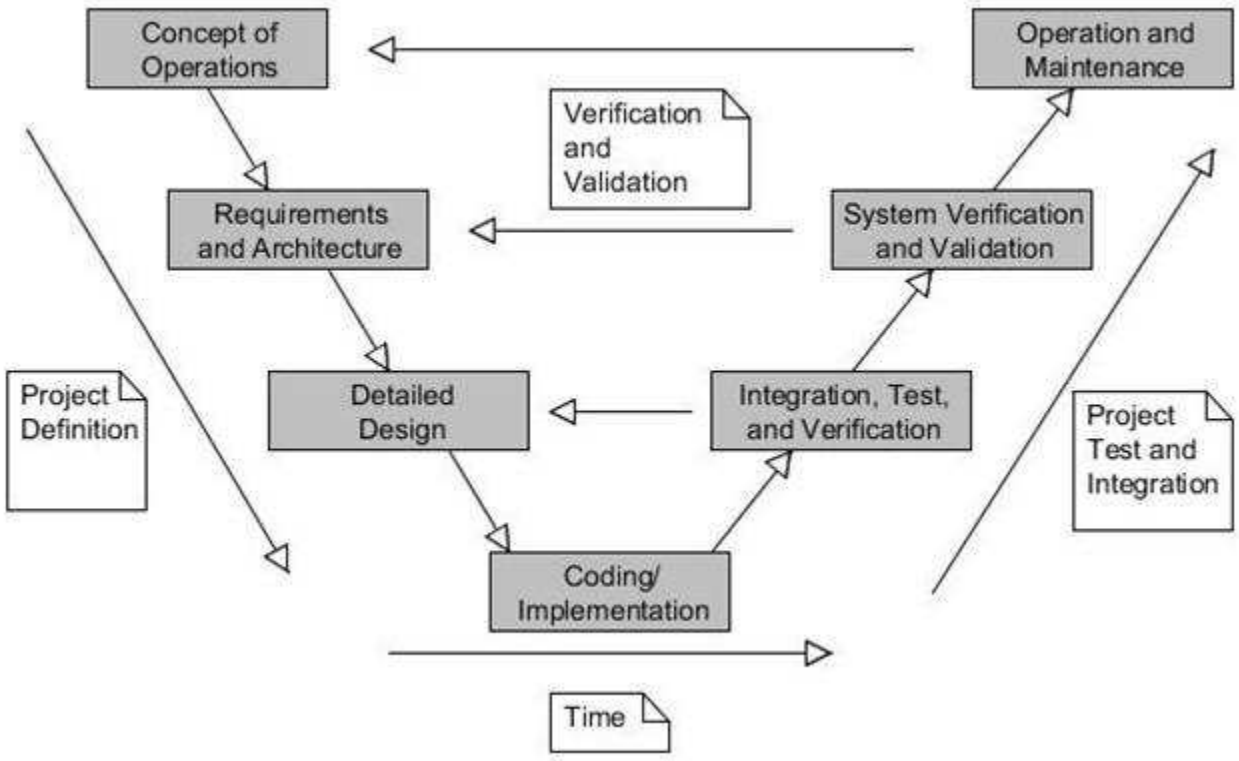
Artifact	**Due Dates**
Meeting Minutes	Due 2 days after each meeting
Individual Logs	Due every Friday
Group Project Management Reports	Due biweekly on Fridays
ConOps	Due 9/21 to Project Manager Due 9/23 for Deliverables I
Project Plan	Due 9/21 to Project Manager Due 9/23 for Deliverables I
SRS	Due 9/21 to Project Manager Due 9/23 for Deliverables I
High-Level Design	Due 10/24 to Project Manager Due 10/26 for Deliverables II
Detailed Design	Due 10/24 to Project Manager Due 10/26 for Deliverables II
Test Plan	Due 9/21 to Project Manager

	Due 9/23 for Deliverables I
User's Manual	Due 11/28 to Project Manager Due 11/30 for Final Presentation
Final Test Results	Due 11/28 to Project Manager Due 11/30 for Final Presentation
Source, Executable, Build Instructions	Due 11/28 to Project Manager Due 11/30 for Final Presentation
Project Legacy	Due 11/28 to Project Manager Due 11/30 for Final Presentation

---

Software Life Cycle Process

Team 1, Inc. developers will follow the V life cycle process model. The V model incorporates the same steps that the group will take in developing the product while also allowing for the verification of design decisions and validation of requirements, which the developers shall monitor during the testing phases, as is indicated in the V model diagram below.



Tools and Computing Environment

The majority of the code for the project will be written in the Python programming language. Python is an object oriented language designed to be easy to read, understand and maintain. It's main advantages are cross-platform compatibility, a large standard library and many open-source third-party libraries. Python is an interpreted language, although it is also compiled after the first use for later runs.

The project will also require two databases, a local database for each instance of the GPS software and a global database to allow individuals to share and aggregate traffic data. These databases will be created with SQL. The global database will use the MySQL implementation while the local database may be either MySQL or SQLite. MySQL has an extension designed to store geographical records. This newly added Spatial Extensions library may prove to be a deciding factor in favor of MySQL on both the global and local databases.

Since all the tools and languages we need for the project are cross-platform compatible and since our team is split between Windows and Linux users, our program will be designed to also work in Windows and in Linux. In those instances where platform specific code is required, such as in the text-to-speech module, we can easily detect which platform the code is running on and run the appropriate function.

Configuration Management

Version and change control will be handled through Git, which is a repository storing all of our project files. Group members have access to shared files which they can update from their own computers. Git has a function which monitors all changes made to each of the shared files, so developers need only comment about changes made.

Quality Assurance

Developers shall be responsible for testing the functionality of their own modules and are encouraged to do so as they write them through techniques such as Unit Tests and Integration tests. New modules should be tested against sample data and be free of major issues before being committed to the version control system.

Developers are responsible for making sure their modules work, but may coordinate among themselves to QA each other's modules if they so desire. Negative test results should be reported through the Issues tracker on GitHub which is integrated with source control. Subsequent positive results should be reported by marking the Issue as resolved.

Risk Management

Group member decides to drop class
The workload assigned to the member will be split up among the remaining students. Some students may have to re-learn information that was known only to the member who is leaving

A group member's computer breaks
Since documents and source code will always be stored in the repository on a remote machine, the loss of data should be minimal. That unlucky group member will need to get their computer fixed ASAP and recreate any changes they made between the last commit and the computer failure.

The source control site goes down
Since the source control system we are using is open source, we can install the software one of the servers that group members have access to and easily recreate the repository from the most recent copy between all group members.

The wiki goes down
The worst case scenario is that all the personal web pages are lost along with the information they contained. We would have to start a new Wiki on another server and accept that the individual progress logs are lost. The project would be able to proceed, but our logs would be incomplete.

The chosen programming language is incompatible with the system design
The sooner this is discovered the better. Best case, we find a serious flaw immediate after starting to code and are forced to fall back to another language. We would lose a little bit of time porting the code to a new language setting our efforts back by a day or so. In a worst case scenario, the flaw isn't discovered until near the end of the project. In this case we would keep most of the project in the current language and port only those modules which had to be written in another language.

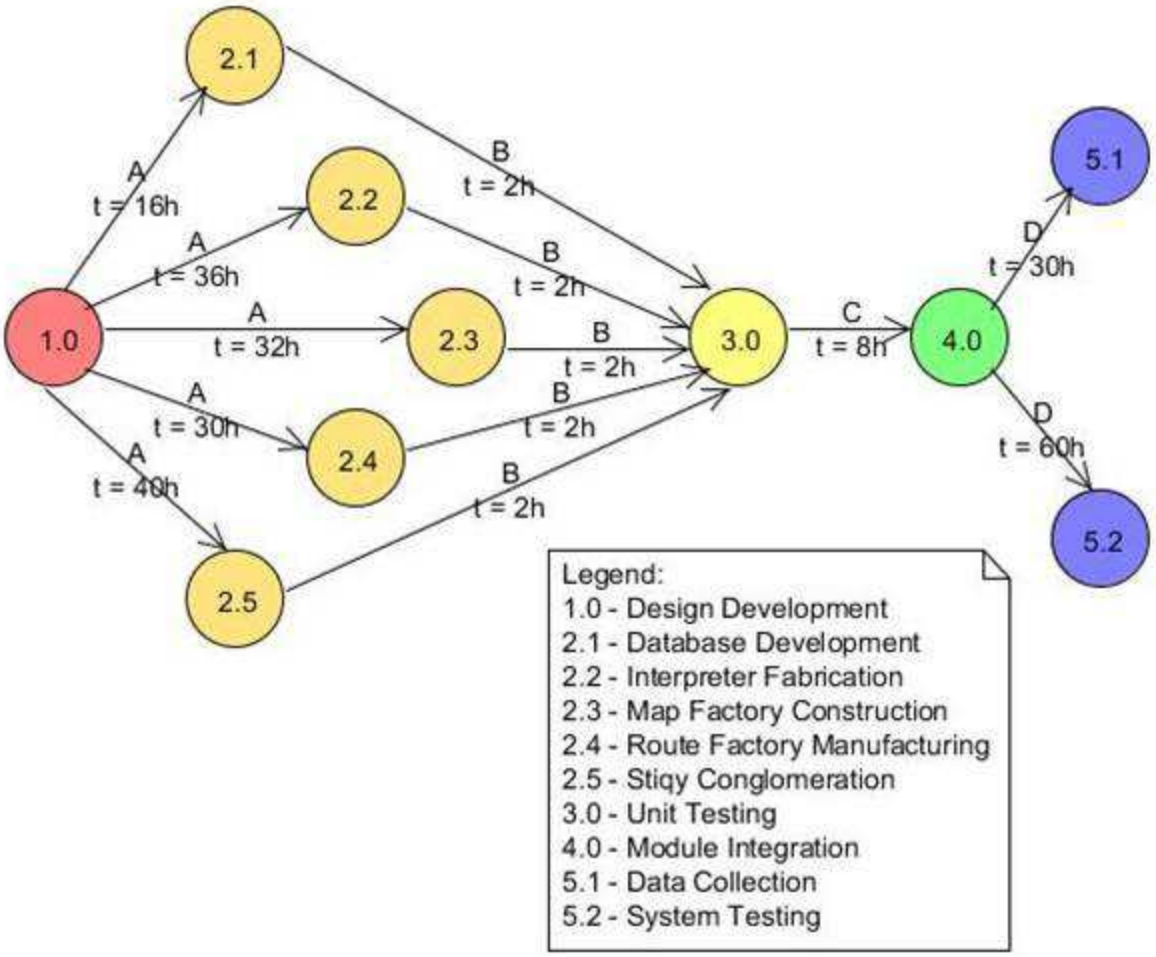
Someone accidentally types "import skynet" and the project becomes self-aware
This is one of the downsides to using Python as the primary language. The danger can be somewhat mitigated by always adding "thisAlgorithmBecomingSkynetCost = 999999999999" to all genetic algorithms.

Table of Work Packages, Time Estimates, and Assignments

The following table is a high-level breakdown of the different modules or steps needed in order to complete the ShortQut system. Note: All time estimates are tentative. Also, listed managers are only who is responsible for making sure each module is completed.

Work Package	Assignee	Estimated Work Time
Design Development	Robby Ronk	10 hours
Database Creation	Laura Barton	16 hours
Interpreter Fabrication	Richie Manis	36 hours
Map Factory Construction	Eric Olsen	32 hours
Route Factory Manufacturing	Steven Nichols	30 hours
Stiqy Conglomeration	Dana Orlando	40 hours
Unit Testing	Robby Ronk	10 hours
Module Integration	Robby Ronk	8 hours
Data Collection	Robby Ronk	30 hours
System Testing	Robby Ronk	60 hours

PERT Chart



Technical Progress Metrics

Requirements phase
In the requirements phase, we will use the number of documents completed verses the number of documents to be submitted to judge the current progress. When a document contains many independent subsections, the number of subsections completed vs the number of total subsections may also be used. These metrics are easy to collect and interpret and should be sufficient.

Analysis phase
In the analysis phase, we will track the number of modules that have been specified. The most natural approach would be to count the number of completed UML diagrams. This metric can be used to track progress from week to week but will be insufficient to measure how close the

analysis phase is to completion. For that, we will need individuals to estimate how many diagrams remain.

Design phase
Lines of Code (LoC) -- This metric is easy to collected and report. It is useful if interpreted correctly. The change in LoC from week to week will allow the project leader to get a sense of when the low hanging fruit in the design begins to be exhausted. As the programmers are forced to tackle more difficult functions, their weekly LoC will decrease and may even be negative if entire functions need to be re-written in light of new information. To encourage proper documentation and readability, comments and whitespace will be included in the LoC metric.

Number of Functions (NoF) -- A more useful, through slightly more difficult to collect, metric, the NoF should give an indication of the number of tasks that have been completed as each function should be simple bock of code corresponding to a single task.

Number of Integrated Modules (NoIM) -- This metric is focused on how many modules have been tested and are known to work together vs the number of modules remaining to be integrated. This metric will only become useful as the project nears the final stages.

Number of Bugs Squashed (NoBS) -- This metric is more about keeping programmers encouraged than tracking actual progress. Humans have a tendency to forget how far they have come, and so this metric can remind them of the gains which have been made.

Plan for tracking, control, and reporting of progress

Each team member will post weekly updates to their individual pages detailing their status, any issues or problems they are working to address, and their individual activity log. Reporting of progress will be automated as much as possible to free team members to focus on writing software/documentation.

Each week the project manager will read the individual logs, collected automated progress reports and examine the work completed. Based on these results the project manager will take corrective action if necessary.

The project manager will issue a revised Project Management Report every two weeks which will include a brief, ~1 sentence description of the project status, a brief description of any planned changes to the project, an updated graph of the technical progress metric (planned vs. actual) for the previous two weeks and an updated PERT chart.

Template created by G. Walton (GWalton@mail.ucf.edu) on Aug 30, 1999 and last updated Aug 15, 2000

This page last modified by Laura Barton on September 21, 2010

ShortQut

Coding Standards

COP 4331C, Fall, 2010

Modification history:

Version	Date	Who	Comment
v0.0	09/16/10	Steven Nichols	Initial version
v0.1	09/17/10	Steven Nichols	Fix formatting
v0.1.1	09/18/10	Steven Nichols	Added webpage Steven's Webpage link
v0.1.2	09/22/10	Steven Nichols	Updated Team Members

Team Name: Team 1, Inc.

Team Members:

- Eric Olsen : EOlsen@knights.ucf.edu
- Steven Nichols : steven.nichols@knights.ucf.edu
- Robby Ronk - rjr@knights.ucf.edu
- Richie Manis - r@knights.ucf.edu
- Dana Orlando - kittymage@knights.ucf.edu
- Laura Barton - lbarton@knights.ucf.edu

Contents of this Document

Document Overview

Reference Documents

Document Conventions

Comments

Naming Conventions

Other Conventions

Programming Language Specific Conventions

Document Overview

This document specifies the standards that will be used when writing the source code files for the project. The goal of specifying these standards is to ease maintenance by improving code readability. As per the requirements in the Applicable Standards section of the Project Management Plan, all requirements herein must be followed to the letter.

Reference Documents

- Project Management Plan -- Applicable Standards

Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Comments

In General

- Multiple comments within the same block SHOULD be delimitated from one another by a newline character.
- In-line comments SHOULD NOT be used as they impair readability. Instead a single-line comment MAY be used directly above the line to be commented.

Modules

Programmer declared modules MUST have comments describing:

- the purpose of the function

Module comments SHOULD be placed directly above the start of the module UNLESS the best-practices of the programming language specify another format, such as with the docstring in Python.

Module comments SHOULD state:

- the author of the module and the date of the function was originally created
- the dates of any modifications along with the names author of the modification and the purpose of the modification

Functions

Programmer declared functions MUST have comments describing:

- the purpose of the function
- the expected type and meaning of any parameters

Function comments SHOULD be placed directly above the start of the function UNLESS the best-practices of the programming language specify another format, such as with the docstring in Python.

Function comments SHOULD state:

- the author of the function and the date of the function was originally created

- the dates of any modifications along with the names author of the modification and the purpose of the modification

---

Naming Conventions

- File, variable, and function names MUST NOT use special characters beyond those provided by the ASCII character set. Such names SHOULD be limited to upper and lowercase Roman character, the ten (10) Arabic digits and possibly the underscore character.
- Variable names MUST be descriptive and SHOULD be as short as possible without loss of meaning.

Other Conventions

- Lines longer than 80 character SHOULD be split into two lines.
- When using languages like Python which are whitespace dependent, TAB characters MUST be converted into four (4) SPACE characters.

Programming Language Specific Standards

Python

- Any code written in the Python programming language SHOULD adhere to the style guide in PEP 8, available at http://www.python.org/dev/peps/pep-0008 .
- Module and Function comments SHOULD use the docstring format described in PEP 257, available at http://www.python.org/dev/peps/pep-0257/

Template created by G. Walton (GWalton@mail.ucf.edu) on Aug 30, 1999 and last updated Aug 15, 2000

This page last modified by Steven Nichols (steven.nichols@knights.ucf.edu) on September 22, 2010

ShortQut

Document Standards

COP 4331C, Fall, 2010

Modification history:

Version	Date	Who	Comment
v1.0	09/18/10	Steven Nichols	Initial version
v1.0.1	09/18/10	Steven Nichols	Updated header with link to Steven's Webpage
v1.1	09/19/10	Steven Nichols	Updated General Formatting section to address possibility of using word processor to write documents
v1.1.1	09/22/10	Steven Nichols	Updated Team Members

Team Name: Team 1, Inc.

Team Members:

- Eric Olsen : EOlsen@knights.ucf.edu
- Steven Nichols : steven.nichols@knights.ucf.edu
- Robby Ronk - rjr@knights.ucf.edu
- Richie Manis - r@knights.ucf.edu
- Dana Orlando - kittymage@knights.ucf.edu
- Laura Barton - lbarton@knights.ucf.edu

Contents of this Document

Document Overview

Reference Documents

Document Conventions

Fonts and General Formatting

Section Formatting

Tracking Changes

Miscellaneous

Document Overview

This document specifies the standards that will be used when writing the documents (including web pages) for the project. The goal of specifying these standards is to enhance consistency and flow across multiple documents written by different authors. The standards described bellow address such questions as font size, headings, spacing, spelling and grammar checking, Table of Contents, figures and tables, authors' names and modification history. As per the requirements in the Applicable Standards section of the Project Management Plan, all requirements herein must be followed to the letter.

Reference Documents

- Project Management Plan -- Applicable Standards

Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

General Formatting

All non-code documents MUST be in HTML format. As HTML presents difficulties to spell/grammar checking, it is acceptable to use a traditional word processor like Microsoft Word to write the documents. If a Microsoft Word is used, writers SHOULD save the document as "Web Page, Filtered" to avoid cluttering the HTML with unnecessary tags and Microsoft specific formatting.

Fonts SHOULD be limited to **Times New Roman, 10-12pt** for body content and **Times New Roman, 12-14pt** for headings. In such places where font-size is automatically set or font-size is specified in a format other than the Point scale, authors MAY use the default font sizes.

Font styles such as **bold**, **italic** or **strikethrough** MAY be used to improve readability with the EXCEPTION of **underline** which is reserved for denoting links and SHOULD NOT be used to format non-hyperlinked text.

The use of indentation and bullets where appropriate is RECOMMENDED.

Section Formatting

Sections within documents SHOULD be separated from one another by a **horizontal line**. Sections SHOULD begin with a section title separated from the horizontal line above and the body text below by the height of a single line.

Each section SHOULD be included in a **Table of Contents** before the start of the first section. These Table of Contents entries SHOULD link to the section title.

Tracking Changes

Each document MUST contain a **modification history** field describing what changes were made, the date of the modification and the author who made the changes. This modification history need not be embedded in the document itself, as long as a modification log is recorded, say with Version Control software.

Miscellaneous

Spell/grammar checking is RECOMMENDED.

Tables SHOULD be labeled with the title of the table appearing above the table.

Figures SHOULD be labeled with the title of the figure appearing below the figure.

Template created by G. Walton (GWalton@mail.ucf.edu) on Aug 30, 1999 and last updated Aug 15, 2000

This page last modified by Steven Nichols (steven.nichols@knights.ucf.edu) on September 22, 2010

ShortQut

Software Requirements Specification

COP4331C, Fall, 2010

Modification history:

Version	Date	Who	Comment
v0.0	8/15/00	G. H. Walton	Template
v1.0	9/16/10	Eric Olsen	Filled in information up to the body/contents.
v2.0	9/21/10	Eric Olsen	All done through Section 2
v3.0	9/21/10	Charles Manis	Added the requirements tables for Section 3
v3.1	9/22/10	Steven Nichols	Updated team members and added links to Reference Documents section

Team Name: Team 1 Inc

Team Members:

- Eric Olsen : EOlsen@knights.ucf.edu
- Steven Nichols : steven.nichols@knights.ucf.edu
- Robby Ronk - rjr@knights.ucf.edu
- Richie Manis - r@knights.ucf.edu
- Dana Orlando - kittymage@knights.ucf.edu
- Laura Barton - lbarton@knights.ucf.edu

Contents of this Document

Introduction

- Software to be Produced
- Reference Documents
- Applicable Standards

Definition, Acronyms, and Abbreviations

Product Overview

- Assumptions
- Stakeholders
- Event Table
- Use Case Diagram
- Use Case Descriptions

Specific Requirements

- Functional Requirements
- Interface Requirements
- Physical Environment Requirements
- Users and Human Factors Requirements
- Documentation Requirements
- Data Requirements
- Resource Requirements
- Security Requirements
- Quality Assurance Requirements

Supporting Material

SECTION 1: Introduction

Software to be Produced:

- A GPS navigation program that learns about rush hour travel times and reroutes accordingly. An external USB GPS will connect to a netbook which will run the program. The software will store the time of travel and the time of day for each road segment and store it in a local database. There will also be an internet accessible central database; this database will hold all information from all users and allow the users to sync to it and use other user's data as well.

Reference Documents:

- Concept of Operations
- Project Plan

Applicable Standards:

- A USB GPS unit
- a laptop with Python 2.6

Definitions, Acronyms, and Abbreviations:

- USB: Universal Serial Bus
- GPS: Global Positioning System

SECTION 2: Product Overview

Assumptions:

- We assume the GPS will connect to the computer via USB
- GPS and computer will communicate using the areNMEA 0183 protocol
- The computer must have at least 1 GB of memory and a 2Ghz CPU.

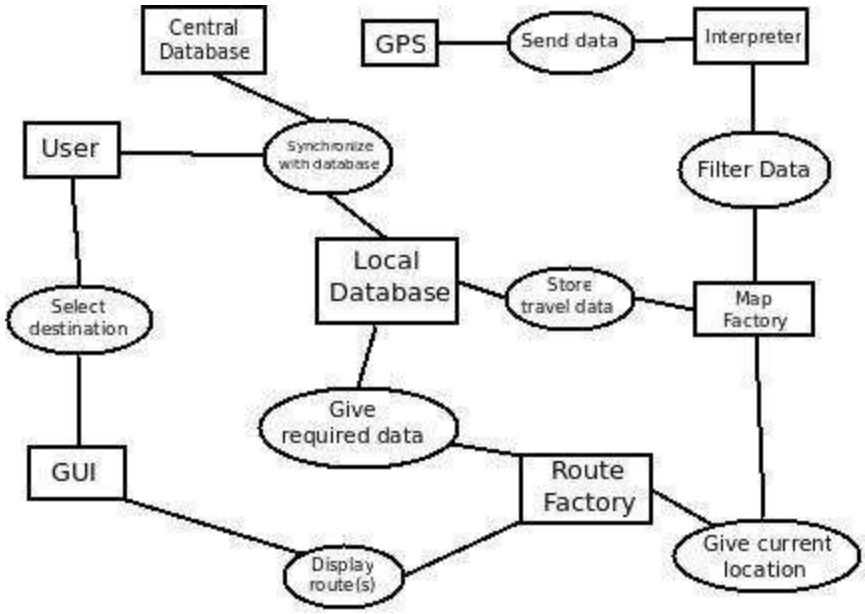
Stakeholders:

- All the members of our team
- The client (aka the TA)
- The professor, Dr. Turgut
- GPS navigation companies (Magellan, Garmin)
- Geoff Huston

Event Table:

Event Name	External Stimuli	External Responses	Internal data and state
Receive GPS data	USB GPS sends data to the computer	Update location and route	Filter data, update local database, reroute if necessary
Create Route	User chooses to find route	Route(s) is displayed on the GUI	A* calculates best routes based on data
Zoom In/Out	User pushes zoom in or out button	View of map zooms in or out depending on input	View distance is changed to reflect the user's input
Synchronize Database	User connects local program to the central database	Display "Update Complete"	Add user's local data to central database and update user's data from central database

Use Case Diagram:



Use Case Descriptions:

- Select Destination - The user will use the GUI to select a destination to navigate to
- Display Route(s) - After A* calculates the best route(s), they are highlighted on the map in the GUI
- Give Current Location - The Map Factory sends the GPS's current coordinates to the Route Factory
- Give Required Data - The Local Database must give the road segments, time traveled, and travel duration to the Route Factory
- Filter Data - The Map Factory takes the raw data from the Interpreter and filters out outliers and other unwanted information
- Store Travel Data - The Map Factory takes the filtered data and stores the relevant information in the Local Database
- Synchronize with Database - When the user connects to the Central Database, the data collected in the Local Database must be uploaded to the Central Database, and the Local Database must be updated with any new data from the Central Database
- Send Data - The GPS must send the data to the Interpreter using the areNMEA 0183 protocol

---

SECTION 3: Specific Requirements

<The following template must be used for each requirement: >

No: <unique requirement number>
Statement: <the "shall" statement of the requirement>
Source: <source of the requirement>
Dependency: <list (with link) to each other requirement on which satisfaction of this requirement depends. (May be "None")>

Conflicts: <list (with link) to each other requirements with which this requirement conflicts. (May be "None")>
Supporting Materials: <list (with link) to supporting diagrams, lists, memos, etc.>
Evaluation Method: <How can you tell if the completed system satisfies this requirement? >
Revision History: <who, when, what>

3.1 Functional Requirements:

No: 1
Statement: The software shall collect data from the GPS device.
Source: The team.
Dependency: None
Conflicts: None
Supporting Materials: • NMEA 0183 • Python serial interface library
Evaluation Method: Does the software know where on Earth it is and report an accurate location?
Revision History: Charles Manis, 2010-09-21, added requirement
No: 2
Statement: The software shall calculate the fastest route from the current location to the target destination using A*. Weights will be retreived from the database.
Source: The team
Dependency: • The current location • The target destination • Local database connection
Conflicts: None

Supporting Materials:

- A* search algorithm

Evaluation Method: The software calculates the shortest path from the current location to the target.

Revision History: Charles Manis, 2010-09-21, added requirement

No: 3

Statement: The software shall get the target destination from the user through the GUI interface.

Source: The team

Dependency:

- GUI

Conflicts: None

Supporting Materials:

- User interaction to get target

Evaluation Method: Can the user set the target destination?

Revision History: Charles Manis, 2010-09-21, added requirement

No: 4

Statement: The software shall provide the user with a GUI centered on the current location with the ability to zoom, pan, and recenter the view.

Source: The team

Dependency:

- The current location
- GUI

Conflicts: None

Supporting Materials:

- User interacting with the software

Evaluation Method: Does the software center the view? Can the user pan? Can the user zoom? Can the user restore the view?
Revision History: Charles Manis, 2010-09-21, added requirement
No: 5
Statement: The software shall synchronize the local database with the central database.
Source: The team
Dependency: <list (with link) to each other requirement on which satisfaction of this requirement depends. (May be "None")>
Conflicts: <list (with link) to each other requirements with which this requirement conflicts. (May be "None")>
Supporting Materials: • Database Interaction
Evaluation Method: Is the software aware of roads? Does the software update the central database's tables?
Revision History: Charles Manis, 2010-09-21, added requirement
No: 6
Statement: The software shall filter (removing outliers) the data from the GPS and determine the road it is on and how long it is taking, noting the current time.
Source: The team
Dependency: • The current location • Local Database
Conflicts: None
Supporting Materials: <list (with link) to supporting diagrams, lists, memos, etc.>
Evaluation Method: Does the software accurately display the road it is on? Does the software correctly calculate speed? Does the software log travel time to the local database?
Revision History: Charles Manis, 2010-09-21, added requirement
No: 7
Statement: The software shall speak directions on the route to the user using a text-to-speech interface.

Source: The team
Dependency: • The next turn to make, from the current route
Conflicts: None
Supporting Materials: • Text to speech
Evaluation Method: Does it talk? Does it say the right things?
Revision History: Charles Manis, 2010-09-21, added requirement

No: 8
Statement: The software shall display route(s) on the map with street names.
Source: The team
Dependency: • Route calculation • The current location
Conflicts: None
Supporting Materials: <list (with link) to supporting diagrams, lists, memos, etc.>
Evaluation Method: Are route(s) displayed on the map as well as street names?
Revision History: Charles Manis, 2010-09-21, added requirement

- < Describe the fundamental actions that the software must perform. Functional requirements can be partitioned into subfunctions or subprocesses. Note: the software design partition does not have to correspond with the functional requirements partition. Functional requirements include:
    - validity checks on the inputs,
    - exact sequence of operations,
    - responses to abnormal situations, including: overflow, communication facilities, and error handling and recovery
    - effect of parameters
    - relationship of outputs to inputs, including: input/output sequences, formulas for input to output conversion
    - ...>

3.2 Interface Requirements:

- < Describe the interactions of the software with other entities. Interface requirements include a precise description of the protocol for each interface:
    - what data items are input
    - what data items are output
    - what is the data type, the format, and the possible range of values for each data item? (i.e. what is the "domain" of this data item?)
    - how accurate must each data item be?
    - how often will each data item be received or sent?
    - timing issues (synchronous/asynchronous)>
    - how many will be received or sent in a particular time period?
    - how accurate must the data be?
    - ...>

3.3Physical Environment Requirements:

- < Describe the environment in which the software must run. Physical environment requirements include:
    - type of equipment on which the software must run
    - location of the equipment
    - environmental considerations: temperature, humidity, ...
    - ...>

3.4 Users and Human Factors Requirements:

- <Describe the users and their constraints:
    - What different types of users must the system support?
    - What is the skill level of each type of user? What type of training and documentation must be provided for each user?
    - Do any users require special accommodations (large font size, ...)
    - Must the system detect and prevent misuse? If so, what types of potential misuse must the system detect and prevent?
    - ...>

3.5 Documentation Requirements:

- <Describe what documentation is required:
    - on-line, printed, or both?
    - what is the assumed skill level of the audience of each component of documentation?
    - ...>

3.6 Data Requirements:

- <Describe any data calculations: what formula will be used? to what degree of precision must the calculations be made? >
- <Describe any retained data requirements: exactly what must be retained?
- ...>

3.7 Resource Requirements:

- <Describe the system resources:
  - skilled personnel required to build, use, and maintain the system?
  - physical space, power, heating, air conditioning, ...?
  - schedule?
  - funding?
  - hardware/software/tools?
  - ...>

3.8 Security Requirements:

- <Describe any security requirements:
  - must access to the system or information be controlled?
  - must one user's data be isolated from others?
  - how will user programs be isolated from other programs and from the operating system?
  - how often will the system be backed up?
  - must the backup copies be stored at a different location?
  - should precautions be taken against fire, water damage, theft, ...?
  - what are the recovery requirements?
  - ...>

3.9 Quality Assurance Requirements:

- <Describe quality attributes:
  - What are the requirements for reliability, availability, maintainability, security, portability ...?
  - How must these quality attributes be demonstrated?
  - Must the system detect and isolate faults? If so, what types of faults?
  - Is there a prescribed mean time between failures?
  - Is there a prescribed time the system must be available?
  - Is there a maximum time allowed for restarting the system after a failure?
  - What are the requirements for resource usage and response times?
  - ...>

SECTION 4: Supporting Material

- <Here is where you put all your analysis work from which you derived the above requirements. It may include UML or other diagrams, notes, memos, etc.)

Template created by G. Walton (GWalton@mail.ucf.edu) on Aug 30, 1999 and last updated Aug 15, 2000

This page last modified by Steven Nichols (steven.nichols@knights.ucf.edu) on September 22, 2010

ShortQut

Test Plan

COP 4331 - Fall 2010

Modification history:

Version	Date	Who	Comment
v0.0	08/15/00	G. H. Walton	Template
v1.0	9/20/10	Robby Ronk	Created Sections 1, 2, 3 and 4. Added 8 test cases.

Team Name: Team1

Team Members:

- Robby Ronk - rjr@knights.ucf.edu
- Richie Manis - r@knights.ucf.edu
- Steven Nichols - steven.nichols@knights.ucf.edu
- Dana Orlando - kittymage@knights.ucf.edu
- Eric Olsen - eolsen@knights.ucf.edu
- Laura Barton - lbarton@knights.ucf.edu

Contents of this Document

1. Introduction:

   Overall Objective for Software Test Activity

   Reference Documents

2. Description of Test Environment
3. Overall Stopping Criteria
4. Description of Individual Test Cases

SECTION 1: Introduction

- Overall Objective for Software Test Activity:
- The software test is expected to find bugs and ineffective processes before the software is released. The software test is also expected to find as many bugs as possible every time it is performed. As many tests as possible are to be automated.

Reference Documents:

- Concept of Operations
- Project Plan
- SRS

SECTION 2: Description of Test Environment

The testing environment will first be simulated on the developers computers. Python has an environment called the toplevel, also known in other languages as a REPL (Read Eval Print Loop). The toplevel allows for interactive development which speeds up production and reveals bugs quickly. After the toplevel, the developers will test the software on its intended platform, which is a netbook with an external GPS device. When the developers are confident that all major bugs are eliminated, users will test the software on a netbook.

SECTION 3: Stopping Criteria

- Testing will continue until a fatal error. All errors will be recorded and sent to the team members responsible for solving the issue along with relevant information. Unit test cases will be created when possible.
- All test cases must pass before the software is ready for release.

SECTION 4: Description of Individual Test Cases

- Test Objective: Verifying the Interpreter
- Test Description: Feeds the Interpreter a set of data sourced from the GPS. The source data will come from a program which records raw data from the GPS.
- Test Conditions: Python toplevel.
- Expected Results: Latitude, Longitude points.

- Test Objective: Verify the Point Filter
- Test Description: The GPS isn't perfectly accurate and so the Map Factory needs to filter the input from the Interpreter. This test will feed the Map Factory with a set of data from the Interpreter with some wild points.
- Test Conditions: Python toplevel.
- Expected Results: Points with outliers removed.

- Test Objective: Map Points to Roads
- Test Description: Filtered points won't be exactly on roads and need to be mapped to roads.
- Test Conditions: Python toplevel.
- Expected Results: Correct roads.

- Test Objective: Test Intersections
- Test Description: The Map Factory is responsible for roads AND intersections. This test will test going across an intersection and making right and left turns.
- Test Conditions: Python toplevel.
- Expected Results: Correct road segments.

- Test Objective: Find time taken of Road Segment
- Test Description: The Map Factory determines the time taken across road segments.
- Test Conditions: Python toplevel.
- Expected Results: The time taken within a %5 tolerance.

- Test Objective: Test A*
- Test Description: A* is a pathfinding algorithm and should be tested for correctness. Many (10 or more) different unit tests will be run on A*.
- Test Conditions: Python toplevel.
- Expected Results: Uniquely correct paths.

- Test Objective: Test Route Generation
- Test Description: This will test the ability of the Route Factory to generate a route using actual roads, using predetermined costs for the roads. Several (3 or more) different tests of this function will be created.
- Test Conditions: Python toplevel.
- Expected Results: Uniquely correct routes.

- Test Objective: GUI Route Display
- Test Description: The GUI will need to display a route to the user in a visually intuitive way. The GUI will be fed a predetermined route and it will be expected to highlight it on the screen.
- Test Conditions: GUI on its own. No other modules necessary.
- Expected Results: A highlighted route.

Template created by G. Walton (GWalton@mail.ucf.edu) on March 28, 1999 and last modified on August 15, 2000.

This page last modified by Robby Ronk Robby Ronk - rjr@knights.ucf.edu