Lecture 18 — Software Requirements & Specifications

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

December 29, 2014

What are Software Requirements?

Something that a piece of software is expected to do.

Do not describe how the software should do it.

They can be formal or informal.

Why Create Requirements?

Creating requirements doesn’t seem like an XP activity...

It’s necessary to have goals for development.

Types of Requirements

1. Functional Requirements
2. Non-Functional Requirements
3. Constraints

Imagine we are making a piece of software for the university so student grades can be entered quickly and easily.

Functional Requirements

Functional: something the software does - a function or task.

Functional Requirements

- Professors must be able to input grades.
- Professors must be able to update grades once entered.
- The system must compute class averages.
- Administrators must be able to search for a student's history.

Non-Functional Requirements

Non-Functional: describe the behaviour of the software.

Non-Functional Requirements

- Professors should be easily able to learn the system.
- A student may not see the grades of other students.
- A record search must take less than five seconds.
- The system downtime must be less than eight hours per month.

Constraints

Constraints: a specialized type of non-functional requirement that restricts the design or implementation of the system.

Constraints

- The server must run on Linux.
- The database must be MySQL.
- The server must be implemented using C++.

Categorizing Requirements

It’s not always obvious whether a requirement is functional or non-functional.
Example: Security

Sometimes we have “mixed” requirements.
Example: “The user must be able to search for music on his phone and see results within 10 seconds”.

Controversy about Requirements

If a system is designed to search songs on your phone, and the search takes one full week to complete...

Stories

Behaviour Driven Development features Stories.

Story describes requirement in a natural way for all parties.

Helps everyone have a common definition of “done”.

How much detail to include? When to split a story up?

Story Template

Narrative:
As a [role]
I want [feature]
So that [benefit]

Acceptance Criteria: (presented as Scenarios)

Scenario 1: Title
Given [context]
And [some more context]...
When [event]
Then [outcome]
And [another outcome]...
Scenario 2: ...

Stories: Title

Title should give a summary of user behaviour.

Specific is better than vague:
“Module X Behaviour” vs “User Prints Invoice”

Stories: Narrative

Answers to questions:

Who will interact with the system? $\rightarrow$ Role.

What to implement? $\rightarrow$ Feature.

Why do this? $\rightarrow$ Benefit.

Stories: Scenarios

Scenario Title says what is different.

Three elements:

Given (Context), When (Event), and Then (Outcome).

How many scenarios is too many?

Stories: Example Title

Obligatory Automatic Teller Machine (ATM) example.
...All discussions of Software Requirements need one.

Title: “Account Holder Withdraws Cash”

Stories: Example Narrative

As an Account Holder (Role)
I want to withdraw cash from an ATM (Feature)
So that I can get money when the bank is closed (Benefit)

Stories: Example Scenario

Scenario 1: Account has sufficient funds

Given the account balance is $100
And the card is valid
And the machine contains enough money
When the Account Holder requests $20
Then the ATM should dispense $20
And the account balance should be $80
And the card should be returned

Stories: Example Scenario

Scenario 2: Account has insufficient funds

Given the account balance is \$10
And the card is valid
And the machine contains enough money
When the Account Holder requests \$20
Then the ATM should not dispense any money
And the ATM should say there are insufficient funds
And the account balance should be \$10
And the card should be returned

Stories: Example Scenario

Scenario 3: Card has been disabled

Given the card is disabled
When the Account Holder requests $20
Then the ATM should retain the card
And the ATM should say the card has been retained

Stories: Example Scenario

Scenario 4: The ATM has insufficient funds...

There is no limit to the number of scenarios in a story.
But at some point it makes sense to split it up.

Stories: Splitting Up

One possible decomposition of the example story:

- Account holder withdraws cash
  (assumptions: ATM is working, card is valid)
- Account holder wants to withdraw cash from broken ATM
  (assumptions: ATM is malfunctioning, card is valid)
- Account holder wants to withdraw cash with invalid card
  (assumptions: ATM is working, card is invalid)

Gathering Software Requirements

Software requirements come from somewhere; we create them.

Use requirements to get customer input into the system.

How NOT to Gather Requirements

If I had asked people what they wanted, they would have said faster horses.

Henry Ford (possibly).

How to Get User Input

1. Meet with a focus group
2. Follow users in their workflow
3. Brainstorming.

Remember: what you get from customers is not requirements; it’s data.

Gathering Requirements

No matter how diligent you are, there will be incorrect, missing, and superfluous requirements.

Plan accordingly!

Case Study: Satellite Tracking Software

Case Study: Satellite Tracking Software

Imagine we are working at the Canadian Space Agency.

We are on the Telemetry, Tracking, and Control (TT&C) team.

TT&C team: ensure communication between satellites and Canadian ground stations

Case Study: Satellite Tracking Software

Video: Satellites 101

Case Study: Satellite Tracking Software

Information from the satellite:

- Status of the spacecraft (resources, attitude, operation, health)
- Scientific data and images
- Orbit and timing data for ground navigation
- Tracking object locations

Case Study: Satellite Tracking Software

Video: Real time satellite in Google Earth

Case Study: Satellite Tracking Software

Satellite Hardware:

- Main antenna
- Signal modulation and demodulation equipment
- Sensors and transducers
- Central computer

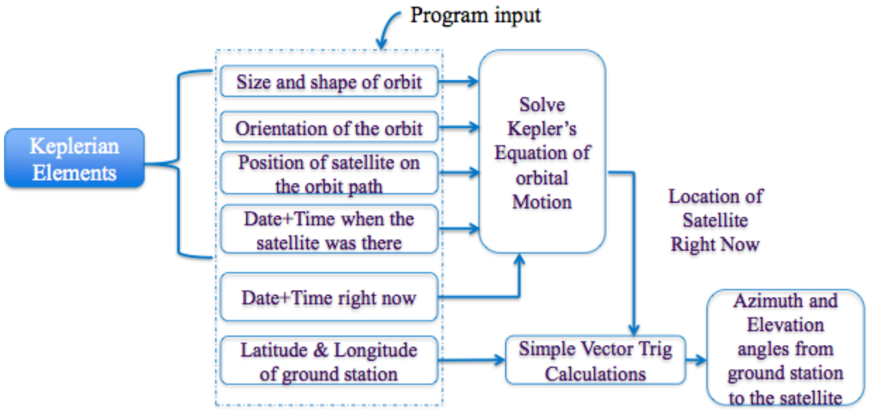
Case Study: Satellite Tracking Software

Tracking software uses a mathematical model of the orbit.

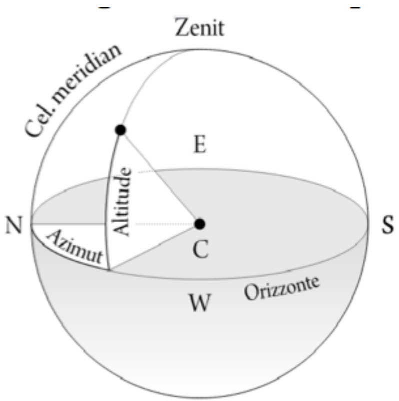
The model takes as input the current state of the satellite.

Using this data and the model, the software predicts the future state of the satellite.

Case Study: Satellite Tracking Software


Program input
Keplerian Elements
Size and shape of orbit
Orientation of the orbit
Position of satellite on the orbit path
Date+Time when the satellite was there
Date+Time right now
Latitude & Longitude of ground station
Solve Kepler's Equation of orbital Motion
Location of Satellite Right Now
Simple Vector Trig Calculations
Azimuth and Elevation angles from ground station to the satellite

Case Study: Satellite Tracking Software



The azimuth is the angle that start from North direction, indicated here as 45 degrees

Note that “Zenit” = Zenith, “Azimut” = Azimuth, “Orizzonte” = Horizon, and “Cel. meridian” = Celestial Meridian

Case Study: Satellite Tracking Software

Video: Satellite Orbit Path

Case Study: Satellite Tracking Software

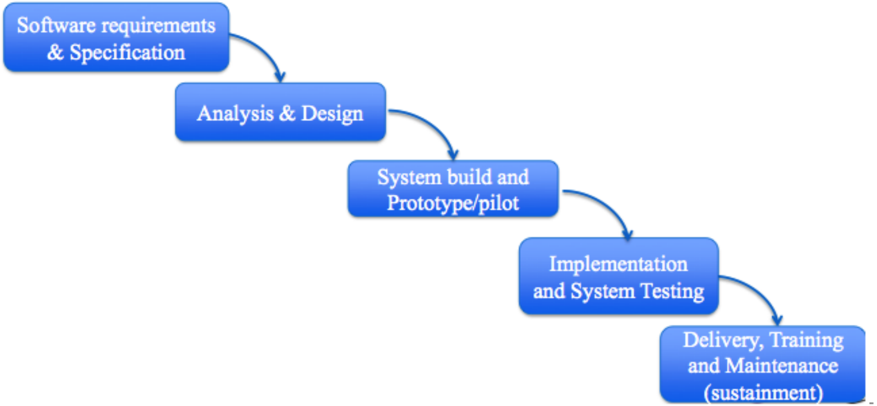
From the background, what are some shortcomings in the existing CSA software?

Case Study: Satellite Tracking Software

Shortcomings in the existing CSA software:

- Code outdated
- Source code was unavailable
- Library was a licensed product (license fees)
- Did not respond well when called by multiple programs

Case Study: Satellite Tracking Software


Software requirements & Specification
Analysis & Design
System build and Prototype/pilot
Implementation and System Testing
Delivery, Training and Maintenance (sustainment)

Case Study: Satellite Tracking Software

What are some

- Functional Requirements
- Non-Functional Requirements
- Constraints

of this system? We don't need specific numbers, but try to cover all the areas.

Discuss for a few minutes; then we'll discuss as a class.

Specifications

… specs are like flossing: everybody knows they should be writing them, but nobody does.

Joel Spolsky

Specifications

Based on software requirements, describe how the software will meet them.

Now it’s time to design the program.

Specification Motivation

Why not just start coding?

It’s faster to change a line of text in a spec than rewrite code.

Communication between different parties:

- Developers
- QA
- Marketing
- Technical Writers
- Customers

Scheduling & Planning.

Without a Spec

Most of the communication still takes place, but inefficiently.

People will get out of sync.

More chance of an error or wasted effort.

Writing a Specification

Goal: describe how the product works from a user perspective.

It will never be complete and perfect. Update as required.

Writing a Specification

Start with the requirements.

We should be able to examine the requirements and identify which sections of the specification fulfill that requirement.

Traceability of requirements.

Writing a Specification

Design should be user-focused.

Create a scenario about how a user will use the system.

Example: professor is ready to enter grades.

More detail and realism results in a better specification.

Writing a Specification

It’s all about the details.

Describe what should happen whether things go right or wrong.

You may spend more time describing error handling.

Specification: Printing

1. No printers are configured.
2. Multiple printers are configured, but no default is set.
3. The data is in colour but the default printer can only print black and white.
4. The print processor encountered an error.
5. The print job was sent, but the printer is out of paper or ink.
6. The printer memory is insufficient to handle the print job.
7. The print job was sent and is now in the print queue.
8. The print job completed successfully.

Writing a Specification

Sometimes, nothing *should* happen.

Scenario not described in the spec? Add it.

Update the specification regularly.
Otherwise it becomes useless if not misleading.

A Collection of Stories

Remember stories?

The specification can be a collection of stories.

Just assemble all the stories.

Writing a Specification

Specs are hard to write as a team.

If it's too large, split it up so one person is responsible for some sections.

Co-ordinate to keep them consistent.

Specification Tips

Tip 1: Write simply.

Don’t write “utilize” instead of “use”.

‘ The query must complete within 10 seconds”
vs.
“It is an absolute requirement that under all circumstances that should a user elect to execute a query on the system, the maximum upper bound on execution time thereof is 10 seconds”

Specification Tips

Tip 2: Review & Reread.

Should be an ongoing process.

Rewrite parts that are hard to understand.

Specification Tips

Tip 3: Avoid Templates.

Might sound like a good idea, but overly formal...

People already hate doing specs - don’t make it worse.

Specification Tips

Tip 4: Don’t be boring.

Funny is one way, but I don’t recommend it.

If your spec is an insomnia cure...

Specification Example

Our example will be *Fog Creek Copilot*
  Providing tech support over the internet.

Called in internal documentation “Aardvark”.

Copilot/Aardvark Overview

“Aardvark allows people to help their friends, relatives, and customers with their computer problems by temporarily taking over their computers over the Internet.”

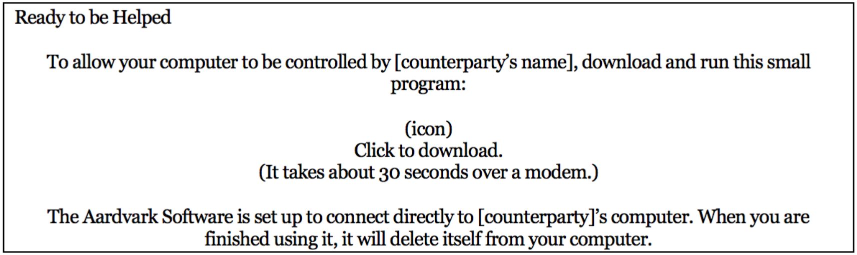
Copilot/Aardvark Major Features

- Easy to get started:
  - No software needs to be permanently installed;
  - Simple fast payment and no commitment
- Version 1.0 is Windows only and requires a reasonable broadband connection.
- Blasts through all firewalls as long as outbound connections on port 443 are allowed.
- Secure (hopefully).

Copilot/Aardvark: Making Changes

When I wrote the first draft of this spec, I had a more complicated flowchart that assumed that either party (helper or victim) could initiate the process. This became extremely complicated and convoluted for various reasons. As I worked through the screens that would be needed to allow either party to initiate the process, I realized that Aardvark would be just as useful, and radically simpler, if the helper was required to start the whole process. Making this change in the spec took an hour or two. If we had made this change in code, it would have added weeks to the schedule.

Copilot/Aardvark: UI Examples


Ready to be Helped

To allow your computer to be controlled by [counterparty's name], download and run this small program:

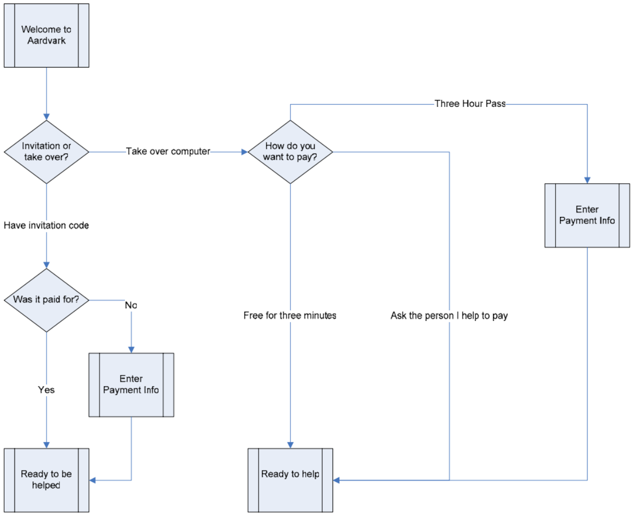
(icon)
Click to download.
(It takes about 30 seconds over a modem.)

The Aardvark Software is set up to connect directly to [counterparty]'s computer. When you are finished using it, it will delete itself from your computer.

Copilot/Aardvark Future Features

- Ability to transfer files between helper and victim, possibly with drag and drop.
- Chat features, either voice or IM style.

Copilot/Aardvark: Pictures = 1000 Words


Welcome to Aardvark
Invitation or take over?
Take over computer
How do you want to pay?
Three Hour Pass
Enter Payment Info
Have invitation code
Was it paid for?
No
Enter Payment Info
Yes
Ready to be helped
Free for three minutes
Ask the person I help to pay
Ready to help