

DILBERT[®]

BY
SCOTT ADAMS



Software Engineering 301:
Software Analysis and Design

Requirements:
A brief overview

Agenda

- What are they & why do they matter?
- Representations
- Requirements analysis

Requirements: The basics

- The requirements for the system define:
 - WHAT the system should do
 - Not HOW the system should do it (usually)
- Why do the requirements matter?
 - Without them, how do you know what to do?
 - Without them, how do you know when you've achieved the “goal”?

Functional vs. non-functional

- Functional requirements
 - The functionality/features of the system
- Non-functional requirements
 - Deal with performance, reliability, legal issues, quality, usability, maintainability, security, ...
- *Does this difference matter?*

“What the system should do”

- According to whom?
 - The people who care about what the system does are called the **stakeholders**
- When?
 - Requirements can change, and often do
- “Should” is different than “must” and “would be nice to”
 - Requirements can be of different priorities to a stakeholder
 - Priorities can differ between stakeholders

Why do requirements change?

- Stakeholders don't always understand well what they need
- Stakeholders can change their opinion about what is important
 - Especially after they see something concrete
- Business environment has changed
- Mistakes in communication
 - Hearing \neq understanding

Developers versus other stakeholders

- Different domains of expertise (often)
- Difficulty articulating what is needed
- Difficulty interpreting what has been stated
- Other stakeholders are not inherently stupid, crazy, lazy, ...
 - Of course, these are real people, and some people have such limitations
- Part of real development is overcoming the barriers



How the customer explained it



How the Project Leader understood it



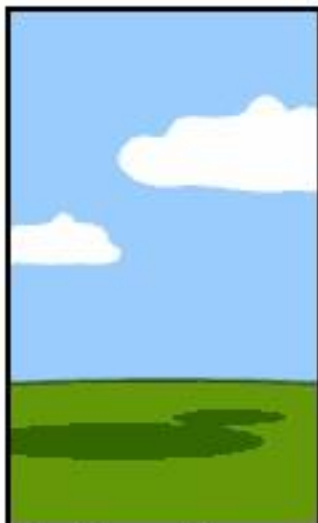
How the Analyst designed it



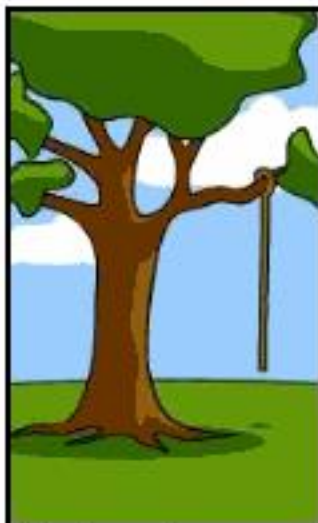
How the Programmer wrote it



How the Business Consultant described it



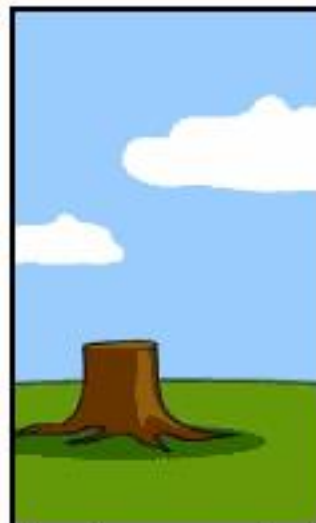
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

Agenda

- ~~What are they & why do they matter?~~
- Representations
- Requirements analysis

Our on-going example

- What are the requirements for the software to run a vending machine?



Hardware vs. software

- Are we modelling the software or the hardware?
 - At this point, it is convenient to ignore the difference
 - act as though the customer is interacting with the software
 - act as though the hardware doesn't exist as such
- Later, we will have to worry about the difference
- This is one strength of abstraction, being able to ignore details sometimes

Some possible representations

- **Natural language text**
- *Structured natural language*
- **User stories**
- *Scenarios*
- *Use cases*
- Formal specification

Vending machine:

Natural language requirements

Our vending machines let customers buy pop by entering Canadian coins and selecting the pop of their choice from a set of options. The total value of all coins entered in the machine and not used in a previous purchase shall be displayed to the customer. When the customer selects a pop and sufficient funds are available, the total will be reduced by the amount of the cost of that pop, the pop will be vended, and the change returned to the customer. If insufficient funds are present, a message will be displayed to that effect. If no pops remain for the selection, a message will be displayed to that effect. A technician will be able to set the price of each available pop, load pops into the machine, and load/empty coins to serve as change. When the machine is open for servicing, other functionality must be disabled, for safety. We are also interested in some sort of “customer loyalty card”, whereby customers can prepay amounts and receive discounts on products. And we are also considering having the machines communicating directly with the central office, to tell us when they are full of coins, out of product, etc.

Natural language descriptions

- “Easy” to read
- Potentially lots of ambiguities
- No guarantee of ...
 - completeness
 - consistency
- Potential admixture of important points and trivial details
- Hard to analyze

Vending machine:

Structured requirements

1. Coin entry

1. Vending machines shall accept Canadian coins
2. Vending machines shall track the total value of all coins entered in the machine not used in a purchase

2. Pop selection

1. Customers shall select the pop of their choice from the available kinds
2. A pop shall be vended when it is selected and its cost is less than the current total available
 1. The current total available will be reduced by the amount of the cost of the selected pop
 2. The remainder shall be returned to the customer
 3. The pop shall be delivered via the delivery chute
3. When the cost of a selected pop is greater than the current total available, it shall not be vended
 1. An error message will be displayed
 2. The total available will not change
4. When a pop is selected which is currently out of stock, no pop shall be vended
 1. An error message will be displayed
 2. The total available will not change

Vending machine:

Structured requirements

3. Display

1. The currently unused total shall be displayed when other information is not being displayed
2. Error messages and informational messages will be displayed temporarily [duration not indicated] and then the currently unused total shall be redisplayed.

4. Service

1. A technician will be able to set the price of each available pop
2. A technician will load pops into the machine
 1. The totals available will be automatically updated
3. A technician will load/empty coins to serve as change
4. When the machine is open for servicing, other functionality must be disabled, for safety.

5. Customer loyalty card

1. Customers can prepay amounts and receive discounts on products.
2. Prepaid amounts shall be recorded on a card [technology details not specified]

6. Networked communication

1. Vending machines will communicate directly with the central office, to inform when they are full of coins, out of product, etc.

Structured requirements: Issues

- Is this really any clearer?
 - True, it is formatted a little more nicely
- What is the user experience like?
- What unusual conditions can arise that have been overlooked and how should these be handled?
- Are there any self-contradictions?
- Are all terms clear and unambiguous?
 - After all, this is still natural language!
- Structured requirements often don't abstract well into useful views

Scenarios

- A simple means of describing desired specific interaction sequences

“Ming inserts \$2 in Canadian coins in the vending machine. He decides which of the available kinds of pop he would like, and selects it. The pop costs less than \$2, so the pop is dropped into the delivery chute, and his change is returned through the coin return.”
- Multiple scenarios needed for other possibilities

Other scenarios (condensed)

- Ming changes his mind altogether and wants his money back
- Ming realizes that he doesn't have enough change, and wants his money back
- The machine doesn't have enough of the pop that he chose
- The machine doesn't have enough change (when does he find out?)
- Ming decides that he would like to return his pop purchase

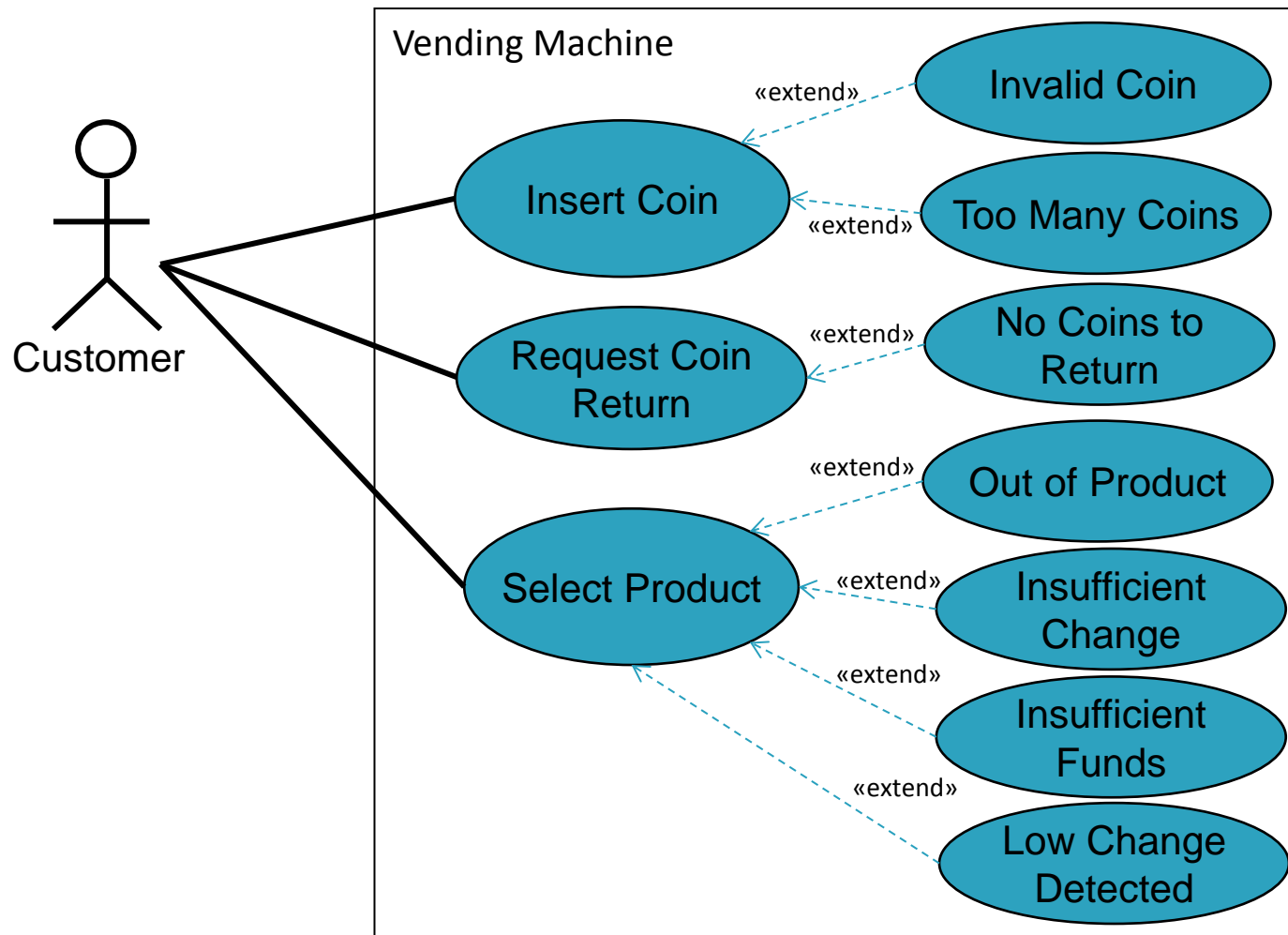
Issues with scenarios

- Positive:
 - Non-technical people can read and write them
 - Easy to see whether there are problems with individual scenarios
- Negative:
 - Voluminous, hence, a lot of work to write down carefully
 - Difficult to analyze as a whole
 - Complete? overlap? contradictions?

Use cases

- A use case is an abstraction of one or more scenarios
 - A use case does not mention specific individuals
 - A scenario is an instance of a use case, as an object is an instance of a class
- Each use case describes a set of scenarios, in terms of **externally visible** behaviour
 - Basically, information going into the system and information coming out
- Use cases model the functional requirements of the system: diagram and descriptions

Vending machine: Use case diagram



Vending machine:

Use case description for Insert Coin

Name: Insert Coin

Participating actor(s): Customer

Precondition: none

Main flow of events:

1. Customer inserts a coin
2. Value of coin is added to current total
3. Revised current total is displayed for the Customer

Postcondition: Current total is updated

Alternative flow (Invalid Coin):

2. Coin is returned to Customer
3. Abort use case

Postcondition: Current total remains unchanged

A snippet of a formal specification (not for the vending machine)

```
\begin{axdef}
amenu: (v_arrow \cross SELECTION \cross selection) \fun SELECTION
\where
    \forall s: selection @ (let n == # (valid s) @
        \forall a: v_arrow; i: SELECTION @ default_selection \leq n \land amenu(a,i,s) \leq n)
\end{axdef}

\begin{schema}{ GetMenuArrow}
    EventUnlocked
    \Delta Menu
\where
    input? \in v_arrow
    menu_item' = amenu(input?,menu_item,item)
    Continue
\end{schema}
```

User stories

- Descriptions about what a user wants from the system
 - Usually written by the customer
 - Usually structured as:
As a <role>, I want <desire> so that <benefit>.
 - The rationale part is sometimes considered optional when obvious.
- For a kind of user (**role**),
- Describes a desired goal or ability to do something (**desire**)
- For a particular reason (**benefit**)

Vending machine:

User stories (a sample)

- As a customer, I want to insert coins to be able to pay for my item.
- As a customer, I want to select my item.
- As a customer, I want to know the cost of an item.
- As a technician, I want to set the prices of items.
- As a technician, I want to open the machine to service it, without injuring anyone.

Epic user stories

- If user stories are too big (called **epics**), they can be split into smaller parts
- For example:

As a customer, I want to purchase pop so that I can drink it.
- Not very useful, as that describes the entire system

Agenda

- ~~What are they & why do they matter?~~
- ~~Representations~~
- Requirements analysis

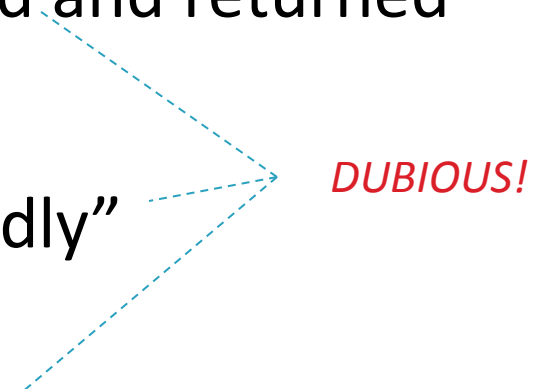
Requirements analysis

- All these issues matter to evaluate problems with requirements:
 - Correctness: Are we trying to do the right thing?
 - Completeness: What is missing?
 - Consistency: Do any details conflict?
 - Reality: Are any details unattainable?
 - Verifiability: Are any details impossible to check for?
 - Cost: How hard will a detail be to attain?
 - Priority: What matters the most?
- } *help define scope*

Example: Realism

- Stories:
 - As a customer, I want to insert coins to pay for a product.
 - As a customer, I want to select a product to purchase.
- Issues:
 - Vending machines run out of product
 - Vending machines fill up with coins
 - Invalid coins are inserted

Problematic statements

- Realism
 - “All invalid coins will be detected and returned”
 - Verifiability
 - “The interface will be user friendly”
 - Cost/verifiability
 - “The system will be error free”
- 
- DUBIOUS!*

Priorities

- Which requirements are the most important ones to deal with?
 - inserting coins, selecting pop, service
 - Because otherwise, the basic purpose of the system is not served
- Next ...
 - requesting coin return
 - Because customers will be annoyed otherwise
 - entering prepaid card
 - Because details are up in the air and these will complicate other functionality
 - networking aspects, setting prices remotely
 - Because these are not clearly necessary to achieve basic purpose, and details are unclear

Priorities and Design

- Lower priority requirements tell us something about likely changes in the future
- Your design should worry about what is to come
 - i.e., don't just worry about what has to be done now, or you may cause yourself trouble later

Remember:

Functional vs. non-functional

- Functional requirements
 - The functionality/features of the system
- Non-functional requirements
 - Deal with performance, reliability, legal issues, quality, usability, maintainability, security, ...

Vending machine: NFRs

- Exercise: Express some NFRs for the vending machine
- If you are tempted to say “safety” ... no!
- Better: “The vending machine must not injure anyone while it is unlocked”
 - This is specific, and phrased as a requirement
 - It may be difficult to test conformance to this requirement, so additional refinement may be needed

Next time

- Testing