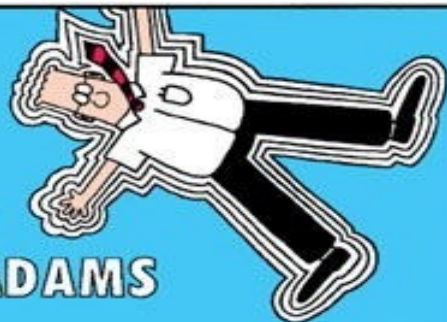




# DILBERT®

BY  
SCOTT ADAMS



E-mail: SCOTTADAMS@AOL.COM



© 2006 Scott Adams, Inc./Dist. by UFS, Inc.



1-21-06



www.dilbert.com

# CPEN 321

## *Requirement Engineering*

# *Today*

- Logistics
- Requirements
  - Why we need to identify, capture, and manage requirements
  - What a good requirement is (and what is not)
  - How to collect requirements
  - How to document requirements

# Something About Piazza... (a.k.a. Piazza Etiquette)

- Great job students answering questions!
- Course staff monitors all questions and answers
  - If we disagree with the answer or believe a clarification is needed, we clarify!
- >150 questions in 2 weeks (record high)  
*... repetitive questions, hard to find important info*



# Something About Piazza... (a.k.a. Piazza Etiquette)

- Created **Discord** server for “chatting”: <https://discord.gg/GEk9XXMM>
  - Not monitored by the course staff
- Piazza is used for important questions of relevance to the entire class
  - Piazza high-quality answers are graded:  
pro-rated from the highest (100%) to the lowest (0%)
  - No marks for asking questions
  - Marks deducted for repeatedly asking Piazza questions  
that were already answered or whose answers can  
be found in the assignment/ syllabus




# *Piazza Etiquette*

- Read/search before you ask. Piazza is not a replacement for self-learning/Googling
  - Repeating questions / follow-ups should be directed to their corresponding thread. Otherwise, they will cause mark deduction.
- Debug before you ask!
- Ask questions correctly (see next slide)
- If the problem is solved, explain how for the sake of other students (and participation marks)
- Do not ask in private (unless personal).  
Private posts that have value to the entire class will be made public



# How to ask good questions (in and outside of this course)



... “why is the server response slow in my case?” 


- What are you trying to achieve
  - What do you expect
  - What do you observe
  - **What did you try to fix the issue**
- 
- GitHub provides templates for reporting issues in a structured way:
    - <https://docs.github.com/en/communities/using-templates-to-encourage-useful-issues-and-pull-requests/about-issue-and-pull-request-templates>

Expected Behavior
Current Behavior
Possible Solution
Steps to Reproduce
1.
2.
3.
4.
Context (Environment)
Detailed Description


# ***Debugging and Troubleshooting***

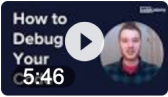
- Debugging and troubleshooting is an important skill
  - Things rarely work on the first try
  - Software developers spend 35-50 percent of their time validating and debugging software

 Videos 





**The Seven Steps of Debugging - Software Debugging**  
YouTube · Udacity  
Feb 23, 2015

9 key moments in this video 



**How to Debug Your Code**  
YouTube · Codecademy  
Apr 23, 2019

10 key moments in this video 



**Best Debugging Tips For Beginners**  
YouTube · Web Dev Simplified  
Jul 9, 2019



# ***What to do if your code is not working***

## ***(by Hartley Brody)***

- #1. Print things a lot
- #2. Start with code that already works
- #3. Run your code every time you make a small change
- #4. Read the error message
- #5. Google the error message
- #6. Guess and Check
- #7. Comment-out code
- #8. If you're not sure where the problem is, do a binary search
- #9. Take a break and walk away from the keyboard
- **#10. How to ask for help**

<https://blog.hartleybrody.com/debugging-code-beginner/>

# *Other Important Info*

1. Groups, two project ideas, and project presentations are to be submitted on Canvas by this Friday, Sept 22, 9pm
  - Instructions on what to include are on Canvas
  - If you do not have a group yet, ask around or use Piazza “search for teammates”
  - Projects will be presented in class next week!
2. We cannot read comments on assignments on Canvas
  - Notes should be added to the accompanying documents
  - Questions should be re-directed to Piazza
3. Do you have **physical** I-Clickers?

# *Today*

- Logistics
- (Summary of last class)
- Requirements
  - Why we need to identify, capture, and manage requirements
  - What a good requirement is (and what is not)
  - How to collect requirements
  - How to document requirements

# ***From Last Week: UML Summary***

- Language to express system requirements and design
- Some diagrams are used more often than others:
  - Simplified class diagrams
  - Use case diagrams
  - Sequence diagrams
  - Activity diagrams (flowcharts)
  - State machines (for full code generation, e.g., with IBM Rhapsody)
- Main benefits
  - Accurately specify design aspects to consider
  - Provide a standard language of communication

# *Requirements*

- Define what the software needs to do
  - To avoid ambiguity, misunderstandings, allow planning
- Tell the problem, not the solution: specify what to build not how
- Precise, well-defined, verifiable, and measurable
- Used to derive appropriate system design

# ***Example of Requirements (from M1)***

The main app screen should have 3 buttons:

- Button 1: Opens Google Maps and shows the location of your favorite city (excluding Vancouver), with the name of the city listed and a pin pointing to the center of the city
- Button 2: Displays on the screen the name of the city where the phone is currently located
- ...

**The software needs to do exactly that!**



How the customer explained it



How the Project Leader understood it



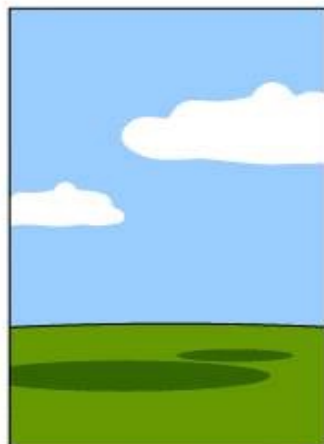
How the Analyst designed it



How the Programmer wrote it



How the Business Consultant described it



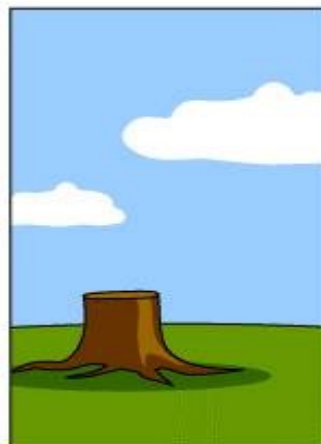
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed



How the customer explained it



How the Project Lead understood it

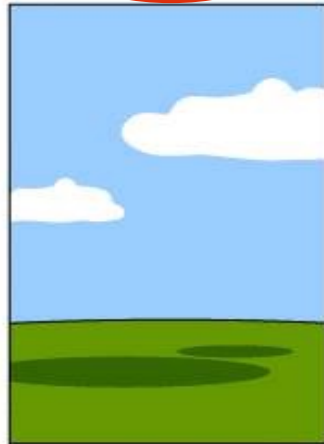


How th



ss Consultant

Failure to capture and manage requirements



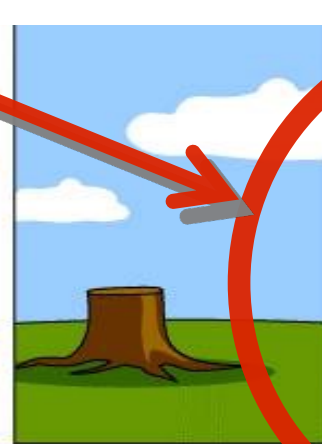
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed



# *Types of Requirements*

- **Functional:** actors and actions
  - The user can browse the catalog of items
  - The user can add an item to an order
- **Non-functional:** performance, safety, security, scalability, dependability, reusability, portability
  - The response time shall be below 100 milliseconds
  - The system shall not disclose the personal user id
  - Adding an item to an order shall not take more than 5 clicks


# ***Good vs. Bad Requirements***

- A good requirement is specific, verifiable, and measurable
- Functional requirements should include a specification of success and failure conditions

## **Not a good requirement:**

- The user should be able to log in
  - Proprietary or third-party?
  - What happens if the login fails?

## **Not a requirement:**

- A system must perform well 
  - Not specific, not measurable

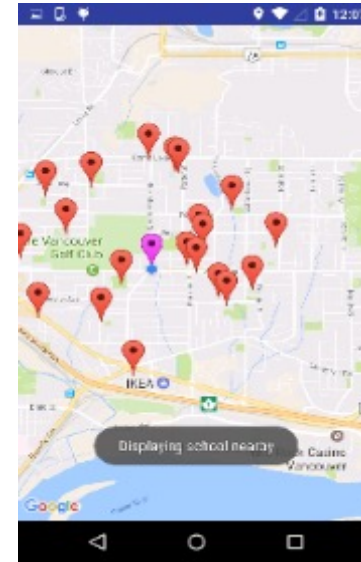
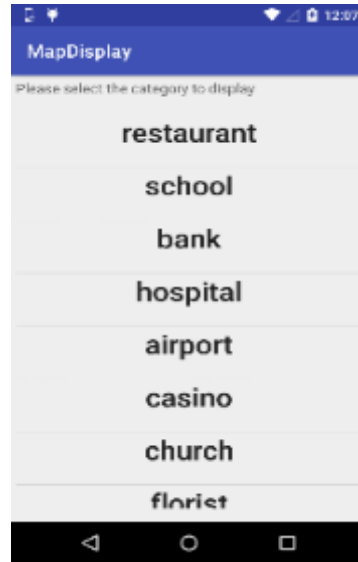
# *Requirements → Design Decisions*

- Good requirements enforce design decisions
- Good design decisions are based on meeting requirements

## Example (from M1)

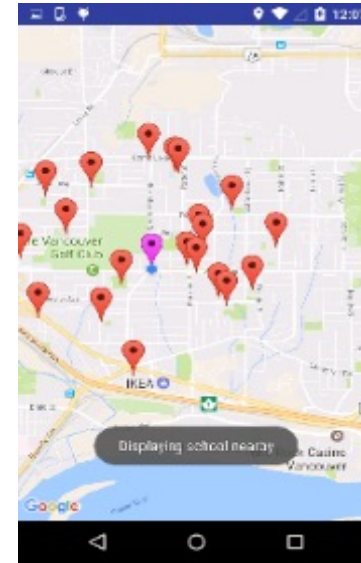
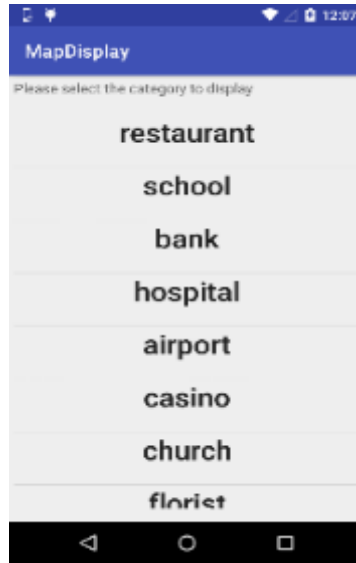
- Button 2: after I clicked the button, I had to wait for a few seconds so that the phone could fetch the location and display the current city. Is that expected behaviour or I have to come up with a method so that the city will show immediately after I clicked the button

# Trade-off Example



***GPS location  
used***

# Trade-off Example



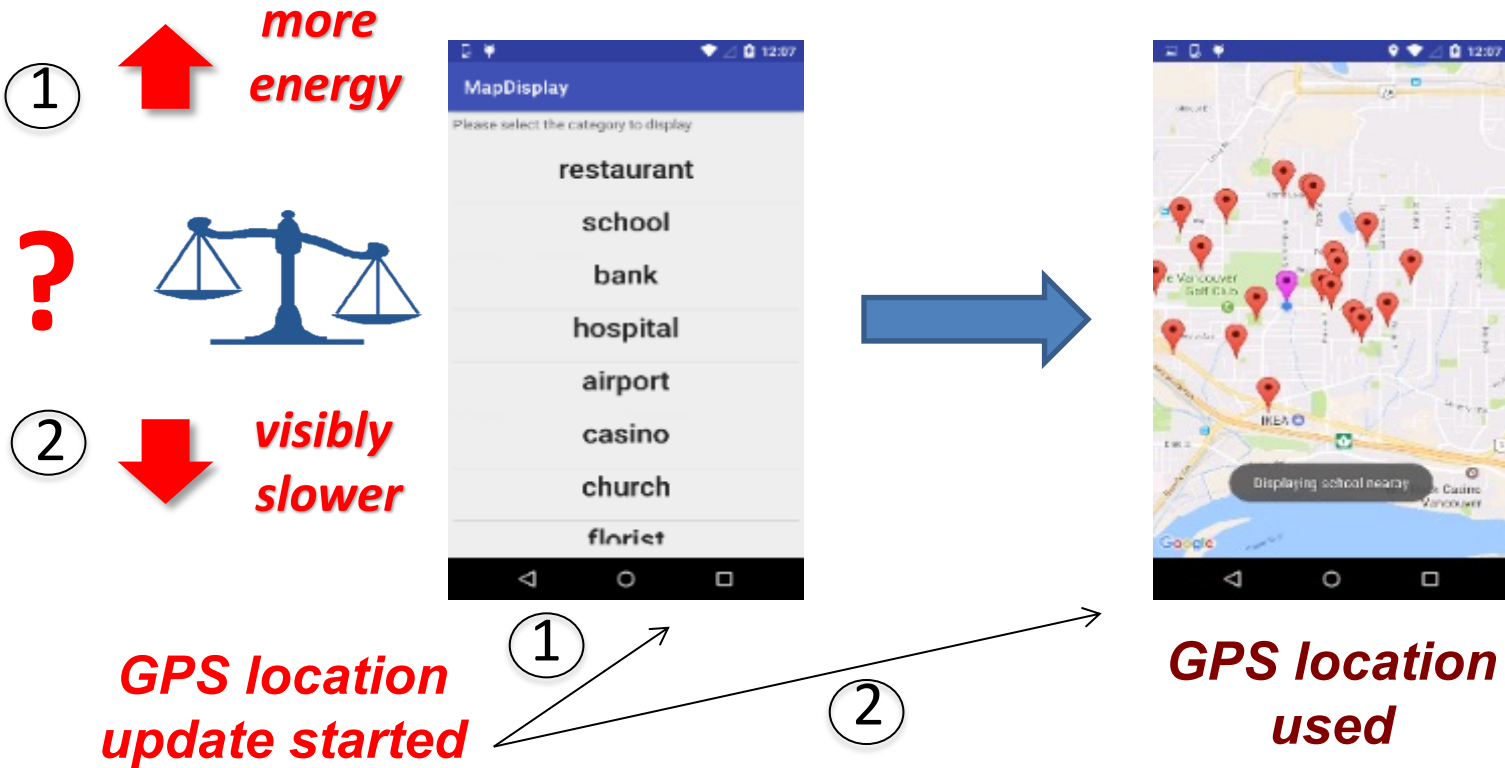
**GPS location  
update started**

①

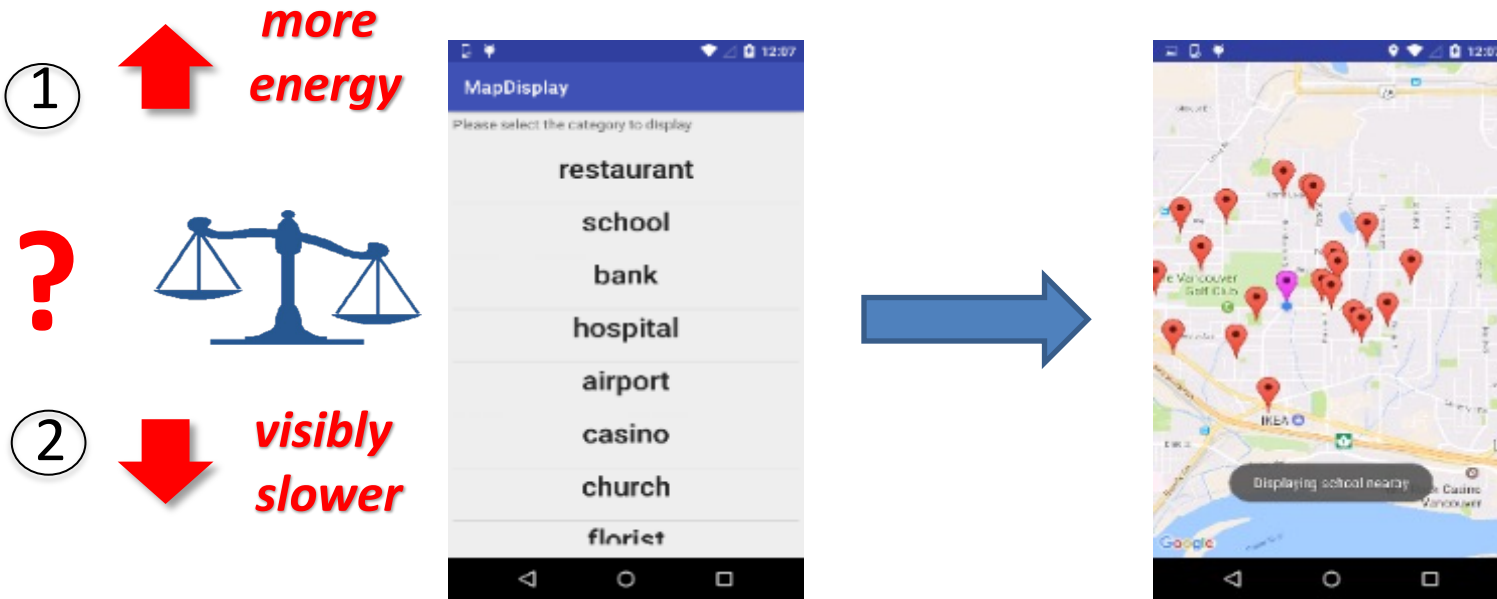
②

**GPS location  
used**

# Trade-off Example



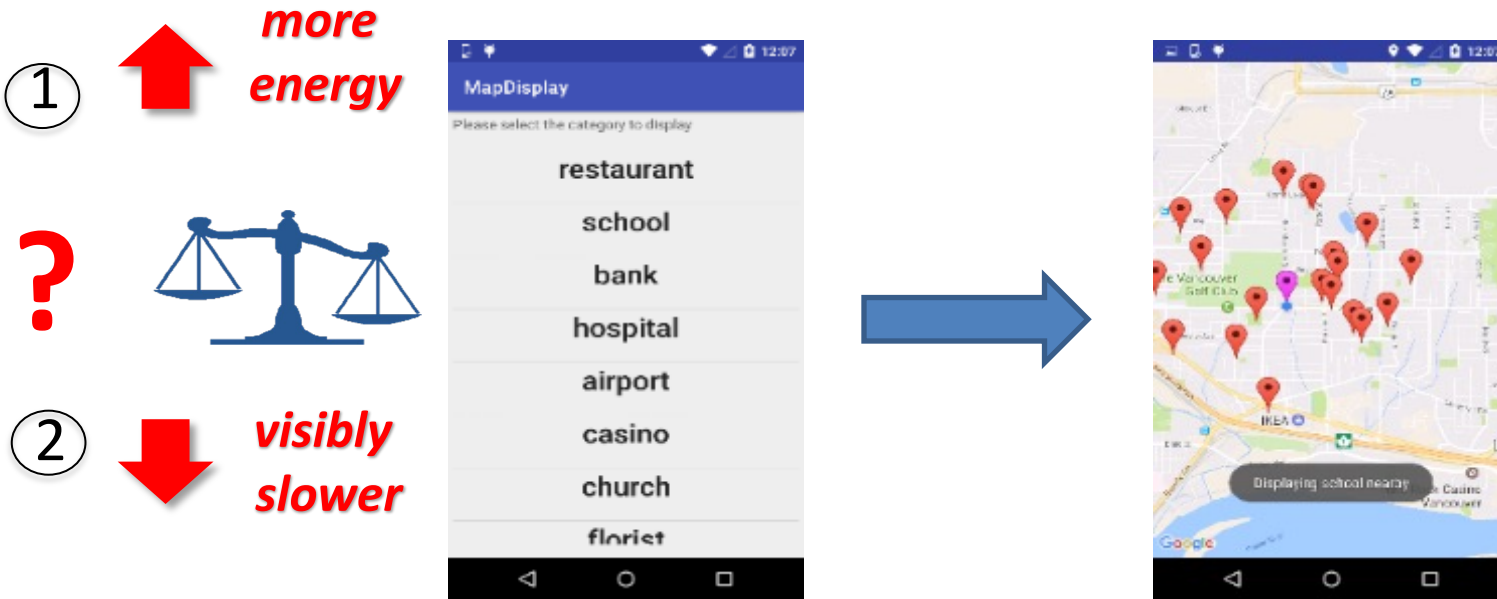
# Trade-off Example



*User-friendly or energy efficient?*



# Trade-off Example



***“Better” depends on user expectations and requirements!***



# *Today*

- Logistics
- Requirements
  - Why we need to identify, capture, and manage requirements
  - What a good requirement is (and what is not)
  - How to collect requirements
  - How to document requirements

**Requirement elicitation:**

the practice of researching and discovering the requirements  
of a system  
from users, customers, and other stakeholders

# ***How to Elicit Requirements from Users***

Access to users is very important

But...

- Users don't always know what they need / can formulate it precisely  
→ It is your job to understand what they want
- What users need changes over time  
→ Agile: Work in iterations, show working prototype, adjust to changes

How?

# *How to Elicit Requirements from Users*

- Talk to the users or (better) work with them and observe how they work
- Ask questions throughout the process to "dig" for requirements
- Think about **why** users want to do something in your app, not just what
- Allow (and expect) requirements to change later.

What if users are not (yet) available?

# ***Personas***

Personas are fictional characters, which you create in order to represent the different user types that might use your service, product, site, or brand in a similar way.

*used often in user experience, marketing, social studies*



# ***Personas***

- Typical users of a system
  - Examples of kinds of people who will use a system
- Characterizes your customers by **behaviors and desires**, not by demographics, etc.
- Should be **substantially different** from each other (unlikely to have more than 3-6!)

# Multiple Personas ...

## THE CASUAL USER



*Pete*

Uses most phone features

Uses phone to make, use contacts send texts and take pictures

Always has mobile device with him

## THE BUSINESS USER



*Jennifer*

Wants a simple phone, but functions as an integrated device

Wants to easily read email and call back the sender

Needs "Popular" mail server integration

## THE POWER USER






*Brad*

Will use almost all built-in mobile functionality

Will extend phone functionality with additional software

Will look through and change change every menu option

# Multiple Personas ...

THE CASUAL USER	THE BUSINESS USER	THE POWER USER
		
<i>Pete</i>	<i>Jennifer</i>	<i>Brad</i>
<p>Uses most phone features</p> <p>Uses phone to make, use contacts send texts and take pictures</p> <p>Always has mobile device with him</p>	<p>Wants a simple phone, but functions as an integrated device</p> <p>Wants to easily read email and call back the sender</p> <p>Needs "Popular" mail sever integration</p>	<p>Will use almost all built-in mobile functionality</p> <p>Will extend phone functionality with additional software</p> <p>Will look through and change change every menu option</p>

*Don't overdo it: 3-6 are enough in most, even very complex, cases*



# ***Personas: Pros and Cons***

## Pros:

- Give a “face” to the potential users. Help understand the customers and therefore to satisfy customer problems
- Align the stakeholder in the entire company

## Cons:

- May lead to a false sense of understanding (instead of talking to real customers)
- Biased on the developer perception: are not real and cannot produce human qualities

# ***Exercise***

- You build an online dating system.

*Identify a few (distinct) personas*

# *Identify personas*

- Alice is a college student interested in browsing profiles in order to snoop on her friends. Not willing to pay.
  - Needed to represent a population of **non-paying users**
- Bob is a software engineer looking to find a younger male date.
  - Needed to highlight **different search criteria**
- Cynthia is a retired nurse who is looking for a soul mate. She faces challenges using mobile technology but is too shy to go to a blind date.
  - Needed to represent a population **of technically-impaired people, who will not easily look for help**
- David is the owner of the system who wants to make sure the system is ethical and legally-compliant.
  - Needed to represent the requirements of the owner, e.g., to delete users

# *Personas ≠ Actors*

- Multiple personas can correspond to one type of actor (user)
- Used to elicit / gather requirements, not necessarily to document them

# *Today*

- Logistics
- Requirements
  - Why we need to identify, capture, and manage requirements
  - What a good requirement is (and what is not)
  - How to collect requirements
  - How to document requirements

# ***How To Document Requirements?***

## **Functional**

- User stories (Agile) / formal use case / structured text
- Mockups
- Prototype

## **Non-Functional**

- Structured text

# ***Option 1: User Stories***

- A high-level definition of a requirement.
- Contains just enough information so that the developers can produce a reasonable estimate of the effort to implement it.
- When “invented”, supposed to write on index cards
  - nowadays mostly in online tools

# User Story Card

Front of Card

173

As a student I want to purchase a parking pass so that I can drive to school

Priority: ~~High~~ Should  
Estimate: 4

Back of Card

Confirmations:

~~The student must pay the correct amount~~  
One pass for one month is issued at a time  
The student will not receive a pass if the payment isn't sufficient  
The person buying the pass must be a currently enrolled student.  
The student may only buy one pass per month.



# User Story Card

Front of Card	Back of Card
<p>175</p> <p>As a student I want to purchase a parking pass so that I can drive to school</p> <p>Priority: <del>High</del> Should Estimate: 4</p> <p><small>Copyright 2005-2009 Scott W. Ambler</small></p>	<p>Confirmations:</p> <p><del>The student must pay the correct amount</del> One pass for one month is issued at a time The student will not receive a pass if the payment isn't sufficient The person buying the pass must be a currently enrolled student The student may only buy one pass per month</p>

As a **role**, I want **something**, so that **benefit**

- As a student, I want to browse courses required for my program, so that I can prioritize my course choices.
- As a shopper, I would like to have a receipt that lists the price and any discounts, so that I have a record of my purchases.
- As a store manager, I want to offer a flat rate discount on items between Dec 26 and 31.

**Be specific!**

- As a store manager, I want to offer items on sale for a reduced price for a limited time, so that I can increase traffic in the store.

(Not specific enough: ☹)

# User Story Card

**Confirmations** (agreed upon discussion/  
understanding of what the team has to do)

Front of Card	Back of Card
<p>175</p> <p>As a student I want to purchase a parking pass so that I can drive to school</p> <p>Priority: <del>High</del> Should Estimate: 4</p>	<p>Confirmations:</p> <p><del>The student must pay the correct amount</del> One pass for one month is issued at a time The student will not receive a pass if the payment isn't sufficient The person buying the pass must be a currently enrolled student The student may only buy one pass per month</p>

Copyright 2005-2009 Scott W. Ambler

1. Success scenario:
  - Customer adds items to the shopping cart.
  - Customer provides credit card number, expiry date, security code, name and address to process sale.
  - System validates the payment and confirms the order by providing an order number.
  - Customer will use this order number to check the order status.
  - System will send a customer copy of order details by email.
2. Acceptance Criteria
  - VISA card must be valid
  - No less payment will accepted than grand total

# ***User Stories as Specifications***

## Pros:

- Lightweight and easy to understand
- Aligned with user intentions
- Enable and promote testing

## Cons:

- Often not descriptive enough
- Too focused on the user: maintenance and non-functional requirements cannot be adequately expressed

## *Option 2: Formal Use Cases*

- Use cases represent the activities that people do when interacting with a system to achieve their goals (remember UML)
- Use cases are an effective tool for communicating and documenting what a system is intended to accomplish
- Formal use cases use a specific structure to represent the information

# *Structure of Formal Use Cases – 1/4*

**ID:** UC-6

**Title:** Register for courses

**Description:** Student accesses the system and views the courses currently available for him to register. Then he selects the courses and registers for them.

**Primary Actor:** Student

**Preconditions:** Student is logged into system

**Postconditions:** Student is registered for courses

# *Structure of Formal Use Cases – 2/4*

## **Main Success Scenario:**

1. Student selects “Register New Courses” from the menu.
2. System displays list of courses available for registering.
3. Student selects one or more courses he wants to register for.
4. Student clicks “Submit” button.
5. System registers student for the selected courses and displays a confirmation message.

# ***Structure of Formal Use Cases – 3/4***

## **Extensions:**

### **2a. No courses are available for this student.**

- 2a1. System displays error message saying no courses are available, and provides the reason & how to rectify if possible.
- 2a2. Student backs out of this use case and tries again after rectifying the cause.

### **5a. Some courses could not be registered.**

- 5a1. System displays message showing which courses were registered, and which courses were not registered along with a reason for each failure.

### **5b. None of the courses could be registered.**

- 5b1. System displays message saying none of the courses could be registered, along with a reason for each failure.

# ***Structure of Formal Use Cases – 4/4***

<b>Frequency of Use:</b>	A few times every quarter
<b>Status:</b>	Pending Review
<b>Owner:</b>	John Smith
<b>Priority:</b>	P3 – Medium



# ***Formal Use Cases as Specifications***

Pros:

- More formal and precise
- Promote defining main and exceptional cases
- Aligned with user intentions

Cons:

- Time-consuming
- Too focused on the user: maintenance and non-functional requirements cannot be adequately expressed

## ***Option 3: Structured Text (for “Maintenance” Requirements)***

Remember that in real projects ~50% of the time is spent on maintenance rather than defining new features

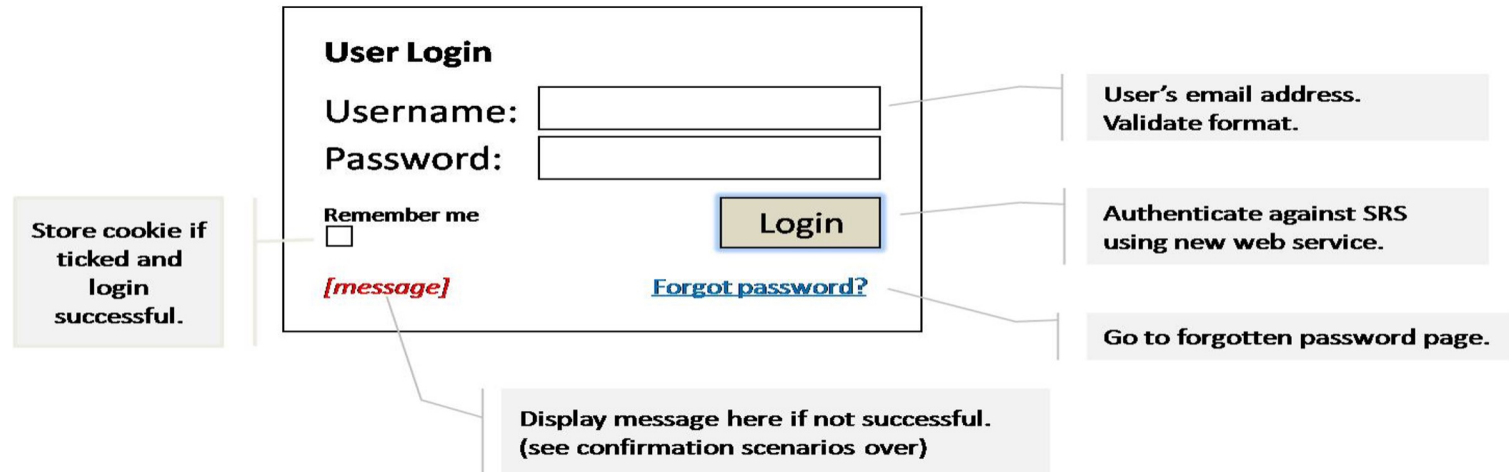
- Upgrade Log4J library to version 2.6 [security]
- Integrate npmlog Node.js logging library [enhancement]
- Fix “Search” to look for partial strings [enhancement]
- Split the “Purchase” component into “Catalog” and “Order” [design improvement]
- Transition to microservice-based architecture [design improvement]

# Formal Requirement Specification Document (a must in regulatory environments)

Chapter	Description
Preface	This should define the expected readership of the document and describe its <u>version history</u> , including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the <u>need for the system</u> . It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
→ User requirements definition	Here, you describe the services provided for the user. <u>The nonfunctional system requirements</u> should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
→ System architecture	This chapter should present a <u>high-level overview of the anticipated system architecture</u> , showing the distribution of functions across system modules.

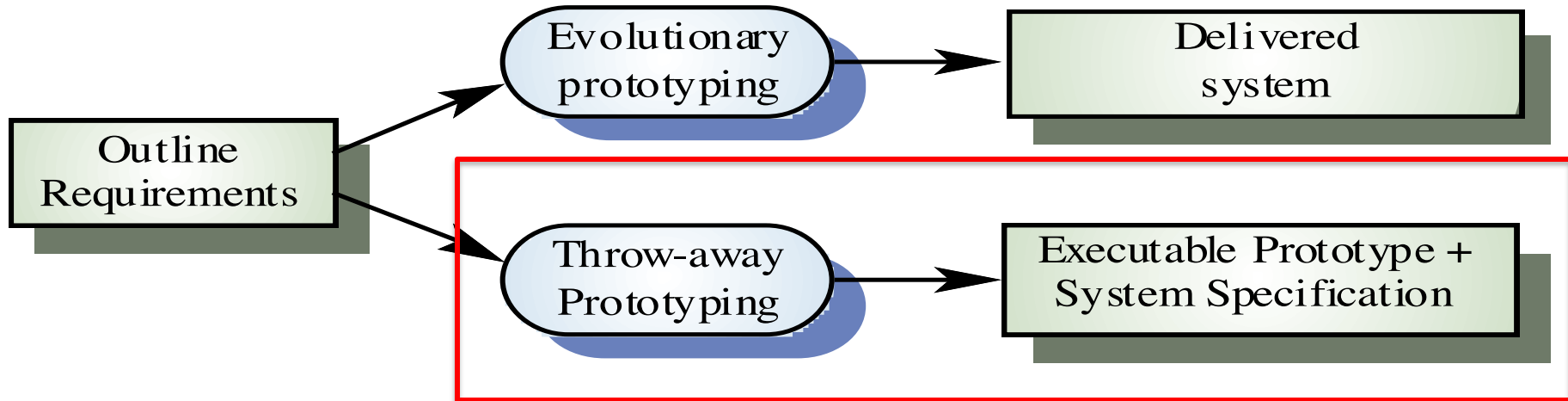
# Mockups / Screen Sketches

- When it is hard to effectively describe in text the look and feel of a user interface...



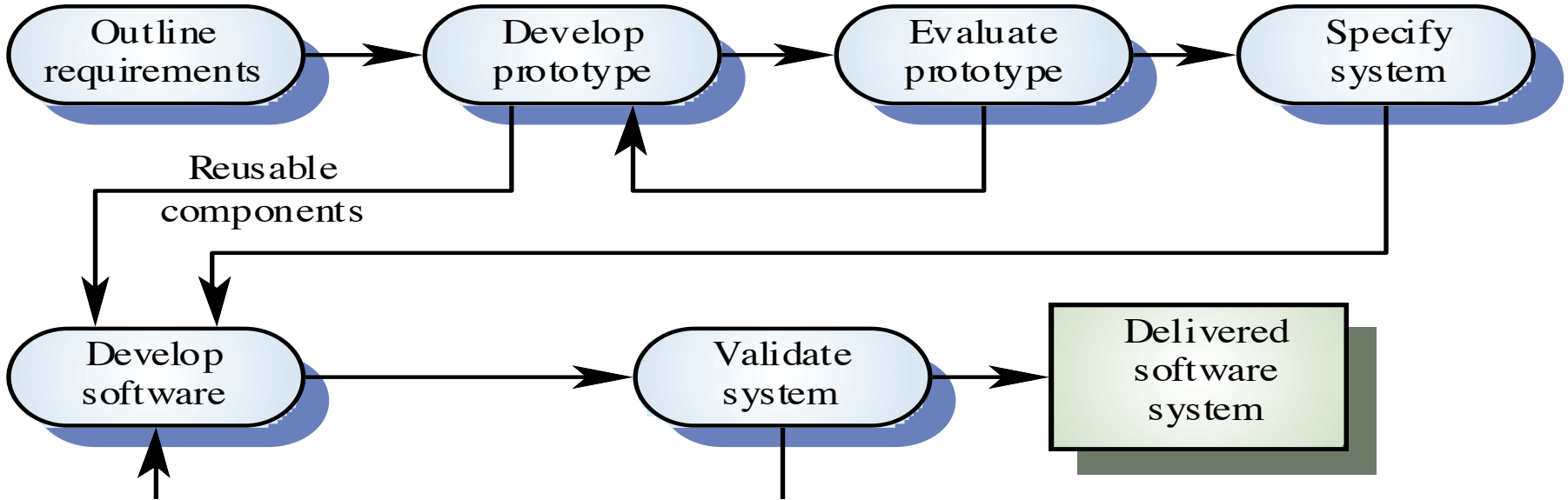
# Throw-Away Prototypes

- When it is hard to effectively describe in text and images what the system does...



# Throw-Away Prototypes

- Used for defining the specification
- (Mostly) throw-away as the system might be poorly structured and difficult to maintain



# ***Throw-Away Prototypes as Specifications***

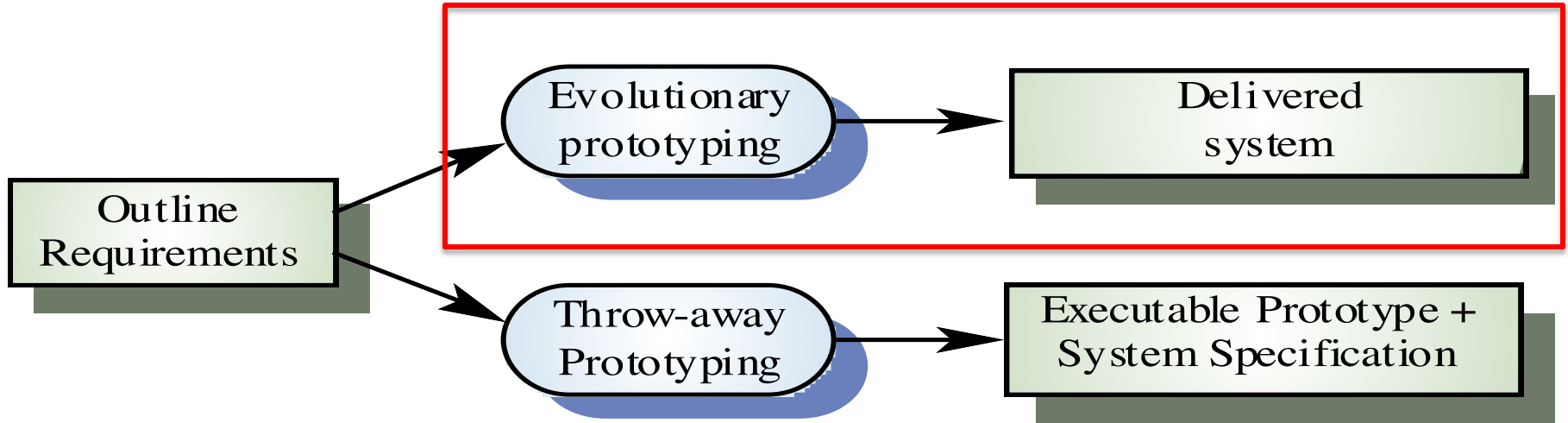
## Pros:

- Clear and easy to understand
- Appealing to the users

## Cons:

- Time-consuming
- Non-functional requirements cannot be adequately expressed
- Some functional requirements (e.g., safety-critical functions) may be difficult to prototype
- An implementation has no legal standing as a contract

# Prototypes



*More practical in this course*



# ***Non-Functional Requirements***

1. The user should not need more than 5 clicks to perform any action [usability]
2. The message should be received within 5 seconds after it was sent [performance]
  - Pull request will drain the battery
3. The search should terminate within 7 seconds [performance]
4. The battery life should not drop by more than 1% after continuous use of the app for 15 minutes [energy efficiency]
  - Better: measure energy with the BatteryHistorian tool

*Remember: specific and measurable*

# ***Defining Requirements in This Course***

## **Functional**

- Actors, functional requirement names, use case diagram
- Lightweight formal use case specification of each requirement:
  - name, short description, primary actors, success scenarios, failing scenarios
- Mockups, if helps/needed

## **Non-Functional (at least 2)**

- Specification for each requirement:
  - Textual description
  - Justification (why needed)
  - Validation approach (how to confirm)

***Example and Exercise on Wednesday!***

# *Summary so Far*

- Why we need to identify, capture, and manage *functional and non-functional* requirements
  - To avoid ambiguity, estimate workload, drive design and testing
- What a good requirement is
  - Specific and precise, testable and measurable
- How to collect requirements
  - Talk to users, analyze their needs, “simulate” users with personas
- How to document requirements
  - User stories, formal use cases, structured text, mockups, prototypes

***See you on Wednesday!***