

HOW TO WRITE REQUIREMENTS



Requirements Before Development

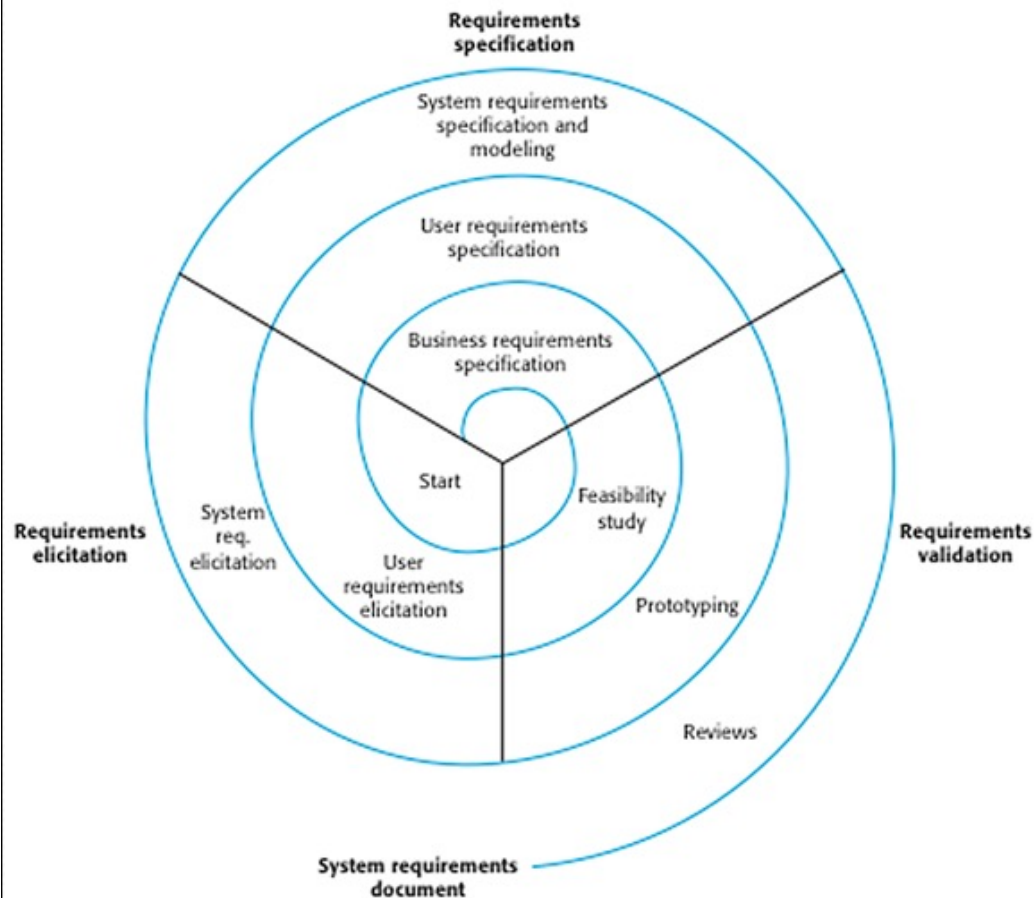
HOW TO WRITE REQUIREMENTS

- What is a User requirement
- Gathering User requirements
- Functional User requirements
 - We will discuss non-functional requirements in the next lecture
- Examples

(USER) REQUIREMENTS

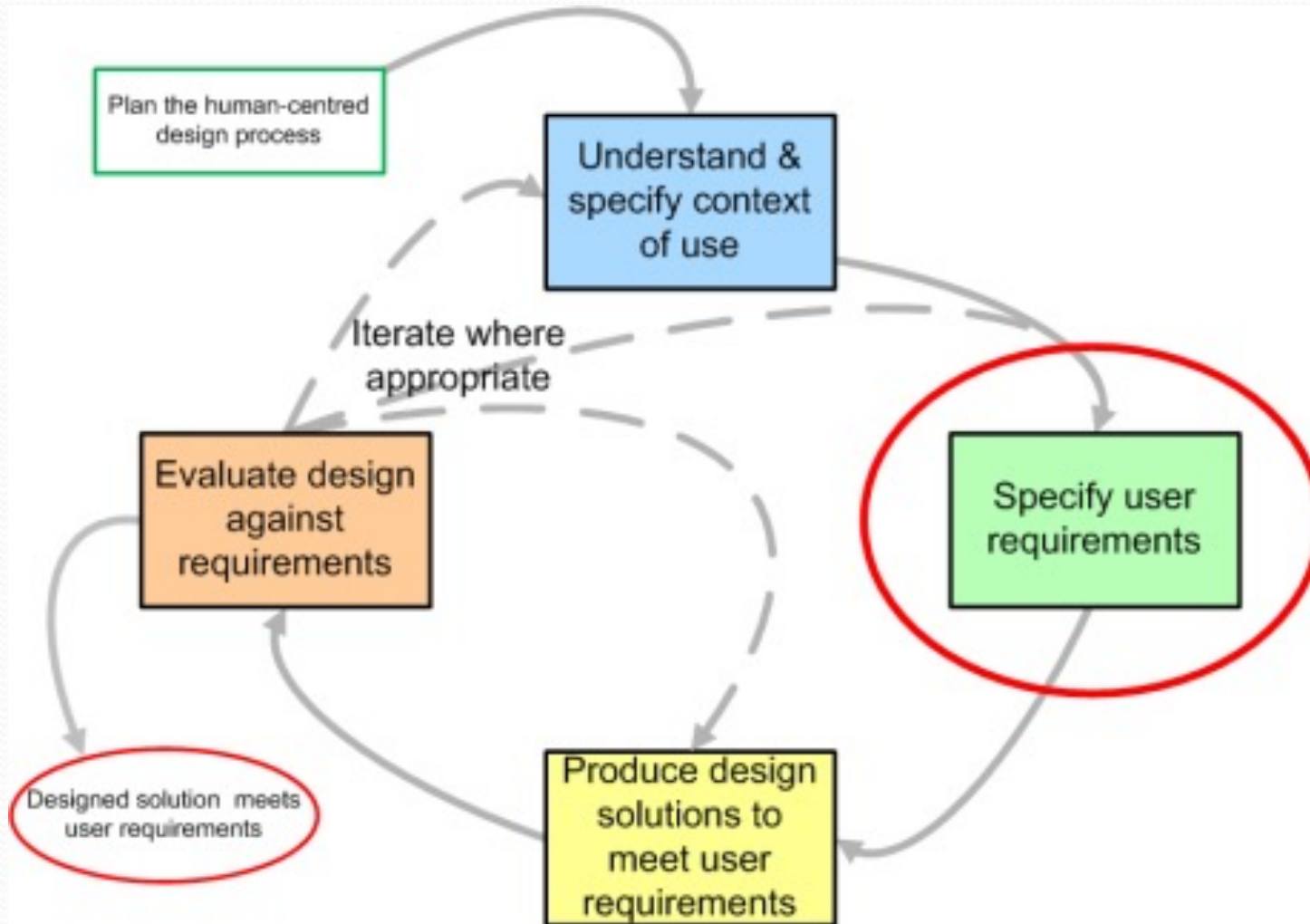
- Analysing what the customer and other users need the system **to do**
- In terms that the customer **understands**
- **What** not How
- Acceptance **tests** (Criteria) written

REQUIREMENT ENGINEERING

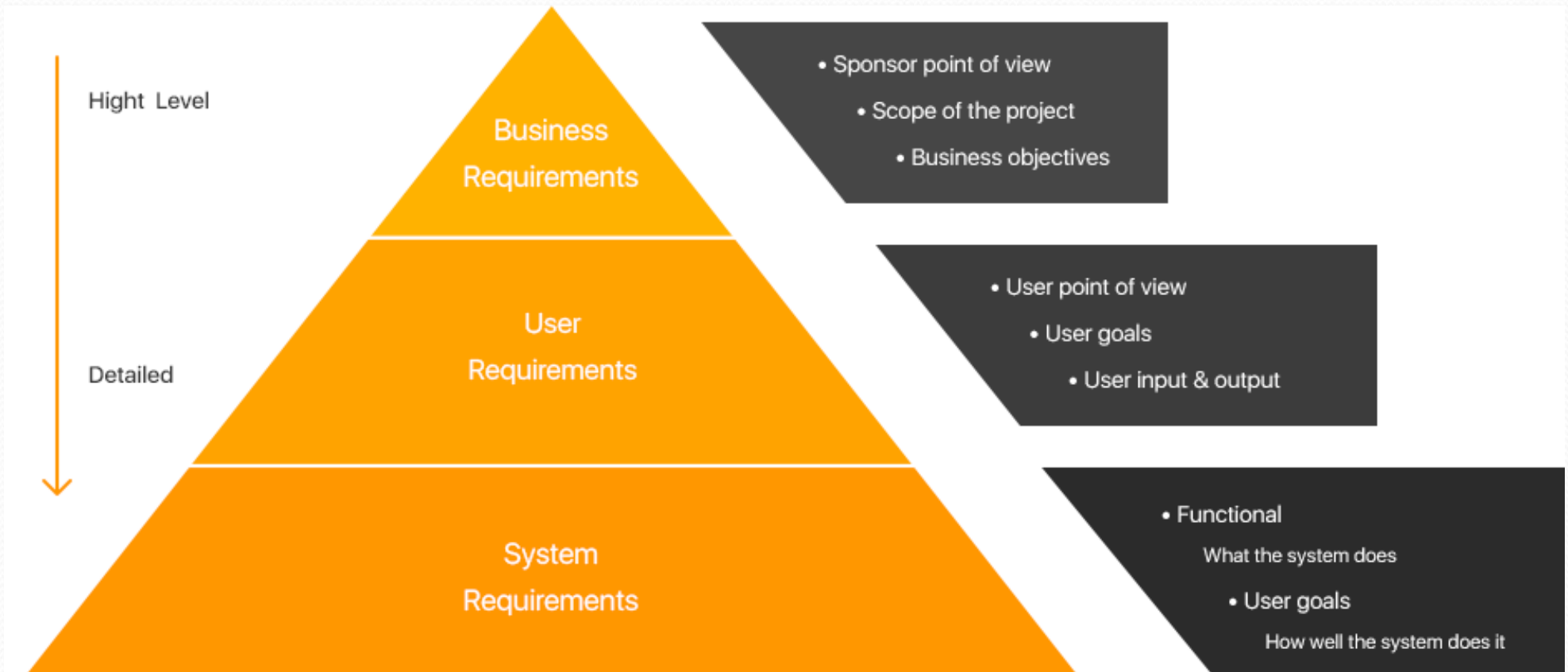


- Requirements determined
 - **very high level requirements** at the start.
 - These will be developed, becoming more detailed, as the systems is developed.
 - 1 feature / requirement at a time. (Agile development)
 - **In detail once high level requirements are agreed**
 - **> Become system requirements**

REQUIREMENTS ENGINEERING

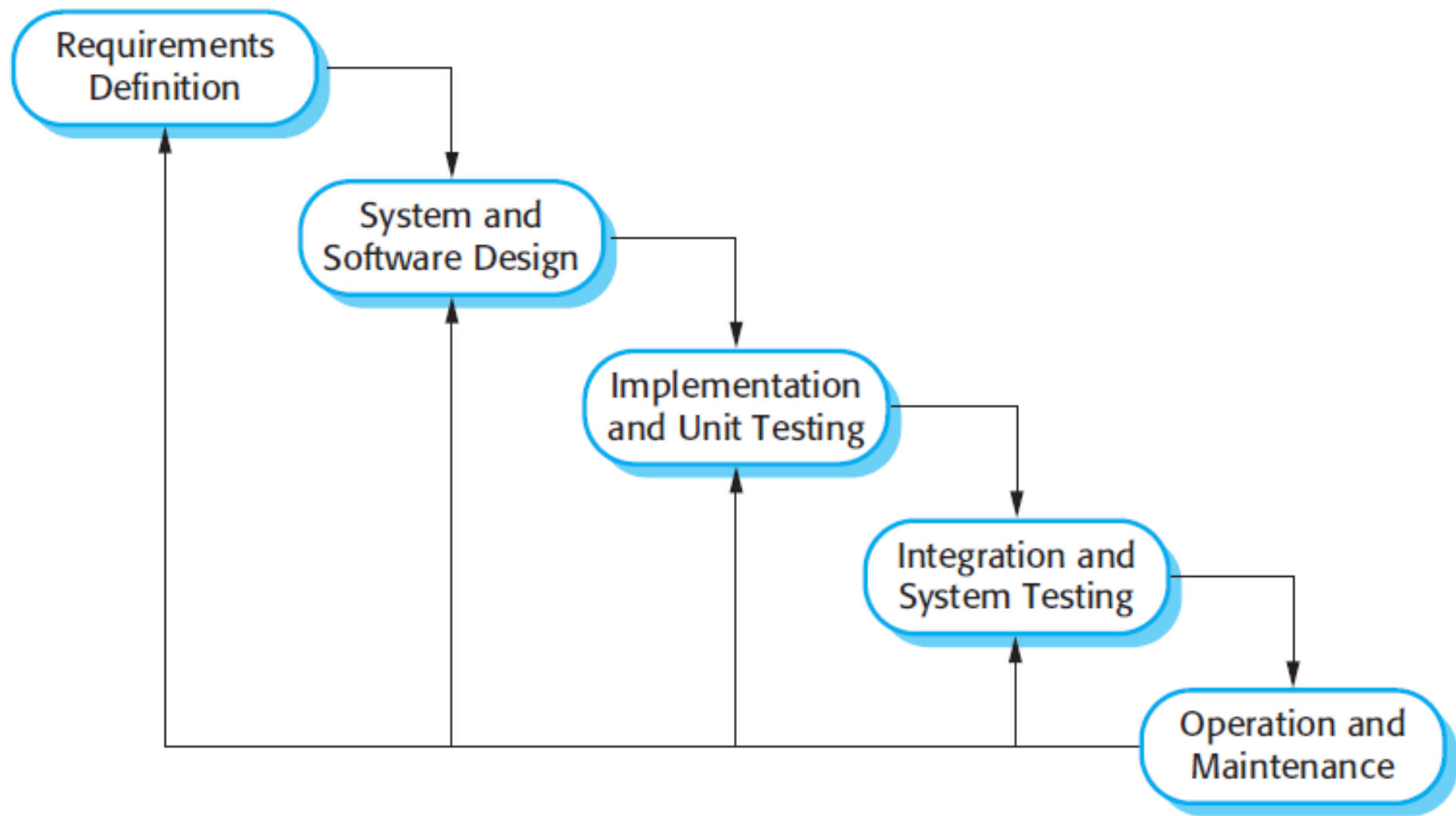


REQUIREMENTS ENGINEERING






















<http://steelkiwi.com/blog/requirements-why-it-important/>

WHY IS IT NECESSARY?



<https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc>

WHO IS BEHIND?

	DEVELOPERS	DESIGNERS	PROJECT MANAGERS	QA	SYSADMINS
SEEN BY DEVELOPERS					
SEEN BY DESIGNERS					
SEEN BY PROJECT MANAGERS					
SEEN BY QA					
SEEN BY SYSADMINS					

https://pbs.twimg.com/media/CS_cSHaXAAj27q.jpg

REQUIREMENTS ENGINEERING IS DIFFICULT:

- Stakeholders may have difficulty **articulating** what they want
- They may have implicit, **domain** knowledge and express requirements in their terms
- There may be **conflicting** requirements
- The importance of requirements may **change**, different stakeholders will have different priorities
- Some may be incompatible, others impractical.
- Constraints may mean that a selection has to be made from the list of requirements.
- **New** requirements may arise



REQUIREMENTS ENGINEERING IS DIFFICULT:

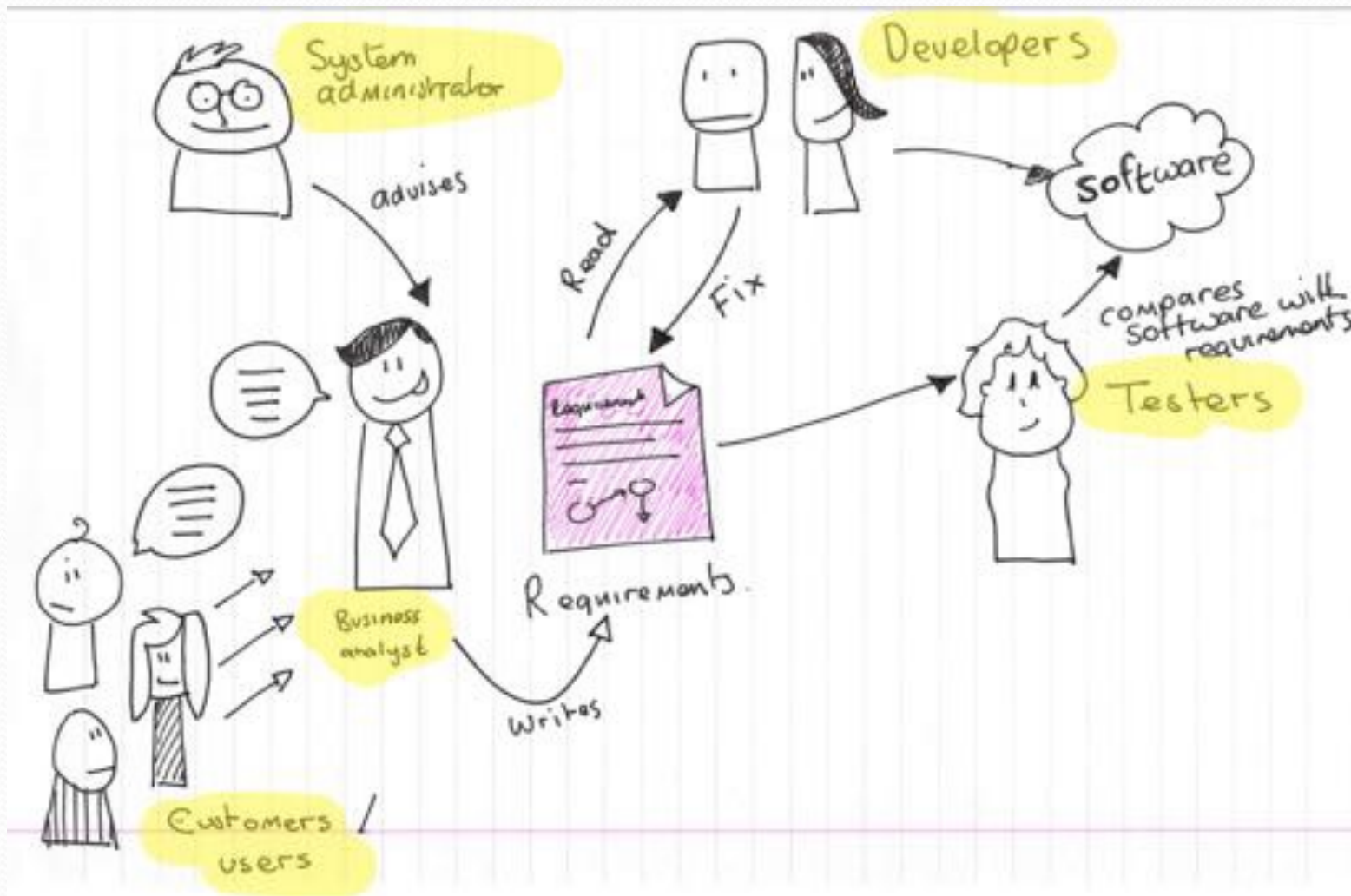


Photo cred: Richard Craggs

We need to care about requirements

REQUIREMENTS ENGINEERING IS DIFFICULT:



How the customer explained it



How the Project Leader understood it



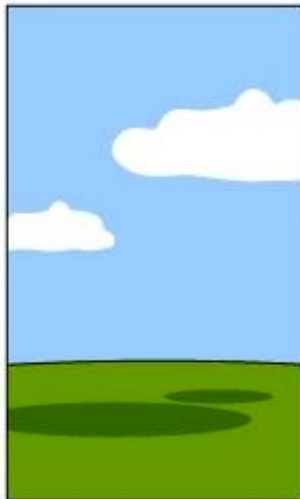
How the Analyst designed it



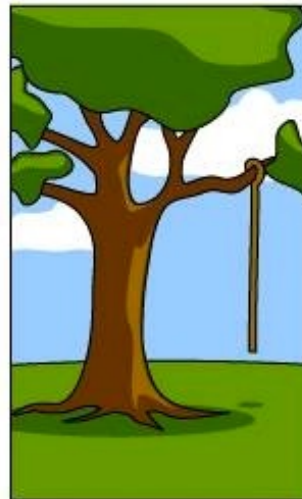
How the Programmer wrote it



How the Business Consultant described it



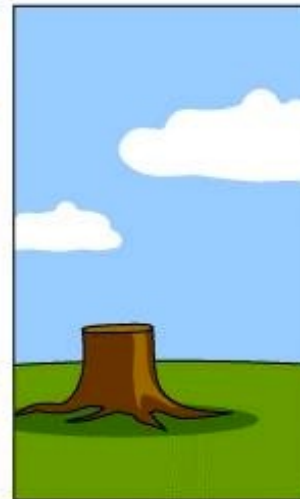
How the project was documented



What operations installed



How the customer was billed



How it was supported



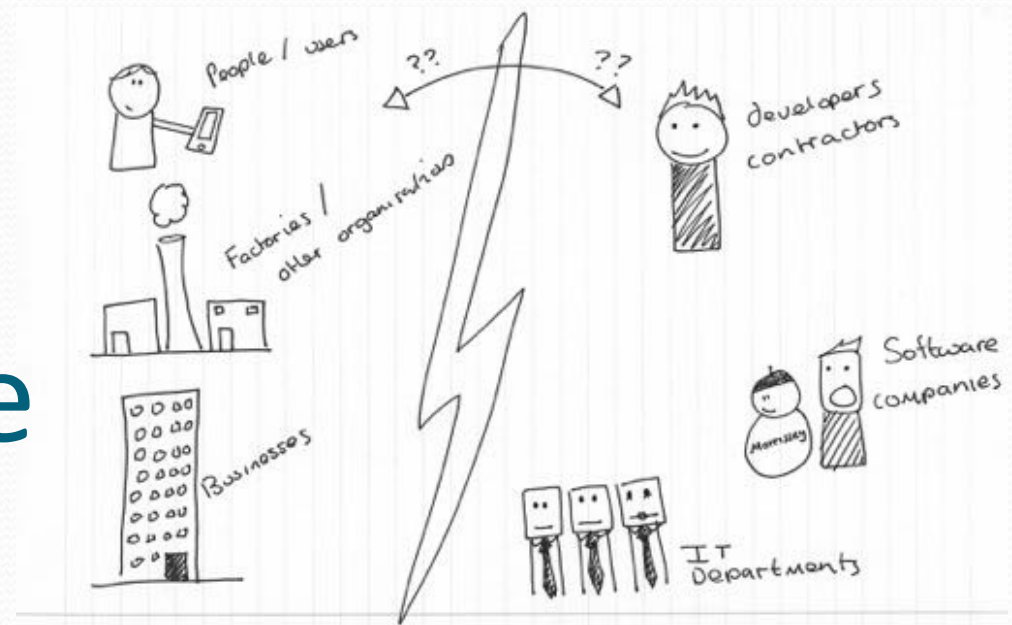
What the customer really needed

REQUIREMENTS ELICITATION:

Working with customers, end users and other stake holders (indirect users) to find out about:

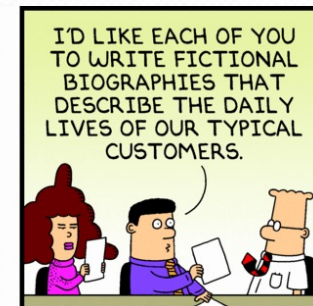
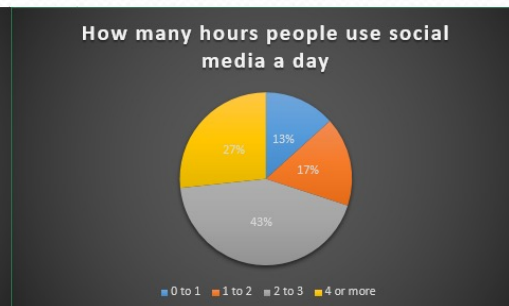
- **the application domain;** stakeholders will use domain language and have implicit knowledge, as the developer you must gain an understanding of the application domain and any existing systems.
- **services the system should provide;**
- **constraints,** (standards, Legal issues, Hardware, Software)

Main Challenge



SOURCES OF REQUIREMENTS :

- **Ask the users;**
 - Interviews (closed & open), Questionnaires, brainstorming
- **Observation;**
 - See what actually happens rather than what people say happens or the procedures that are described in documents, shows requirements based on cooperation and awareness of other peoples activity. Can help to determine implicit requirements.
- **Documents;**
 - Can provide information about the domain. Forms tell you about data in the domain (e.g. a bank paying in slip), manuals of procedures tell you how things are supposed to be done, (but remember may not be kept up to date).

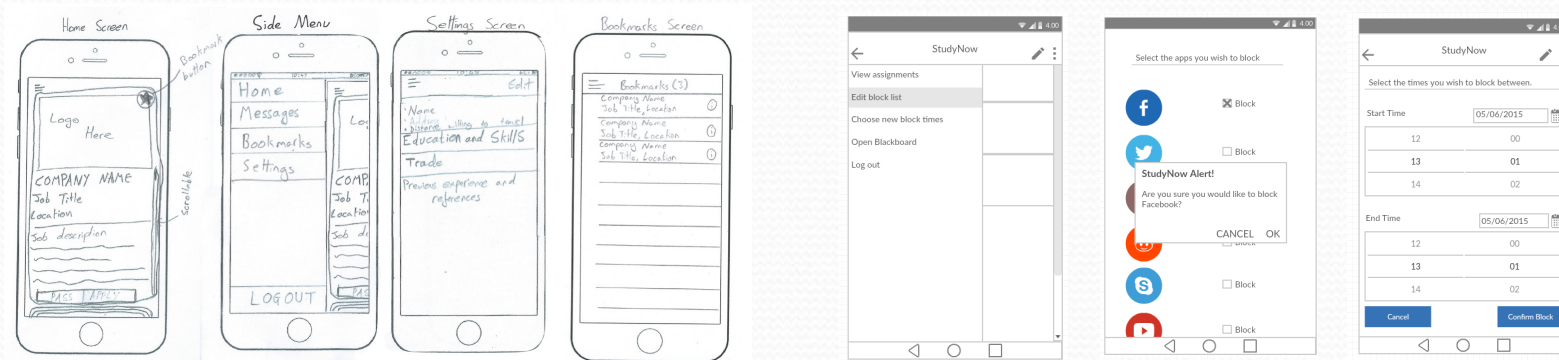


SOURCES OF REQUIREMENTS :

- **Scenarios; -**
 - Integral part of agile methods, see later in module
 - descriptions of example interaction sessions,
 - easier to relate to a 'real-life' example so users can understand and critique a scenario.
 - Can be text based.

NOTE - Use-case is a scenario based technique, it can be used to identify individual interactions with the system and supported with descriptive text or with other **UML models** which provide more detail.

- **Prototypes. –** Can be used to help clarify some requirements.



REQUIREMENTS VALIDATION

- The **requirements** specification needs to be **validated** to ensure that stated requirements are the ones that the different **stakeholders actually want**
- Requirements should be reviewed to see if they are:
 - **Valid** - expresses the real needs of the stakeholders
 - **Unambiguous** - not be open to misinterpretation
 - **Complete** - no relevant aspect has been left out
 - **Consistent** - no contradictions or conflicts with other requirements
 - **Feasible** - possible to satisfy requirements within known constraints
 - **Testable/verifiable** - measurable where possible - avoid vague terms

REQUIREMENTS VALIDATION

- The User requirements specification will be used to develop the User Acceptance testing UAT's of the system
- User Acceptance tests should be defined
 - Developing a verifiable test criteria for each requirement helps reduce ambiguity in validating the final product

SPECIFYING USER REQUIREMENTS

- Two types of Requirements
 - **Functional** -- What the customer needs the system to enable then to do.
 - **Non Functional** – What the customer needs the system to be. (non functional requirements can lead to functional system requirements.)

FUNCTIONAL REQUIREMENTS

- Can be written in Natural language
 - Prone to ambiguity, therefore acceptance criteria
- Models – UML – Use Cases (Diagram and Descriptions)
Structured natural language - Used to document **functions** provided by the software
 - OOA - Description of a use case might include - use case ID, use case name, actors, pre-conditions, description of the main steps in the scenario, related use cases, post-conditions (return to later)

FUNCTIONAL REQUIREMENTS

**A Functional requirement is
a specific thing a
stakeholder wants to be able
to do using the system.**

- Statements about what the stakeholder should be able to do
- Literally the functions 'someone acting on the system' can carry out.

USER REQUIREMENTS

Describe what the **user does** with the system, often referred to as **user needs**

- What activities the user might be able to perform
- The services that the system should provide and the constraints under which it must operate
- Should be understandable by users who **don't have technical knowledge** -> **avoid details on features, software jargon, or formal notations.**
- User Requirements - > User Requirement document
 - Use personas, scenarios, user stories, etc.

Primary input for creating system requirements

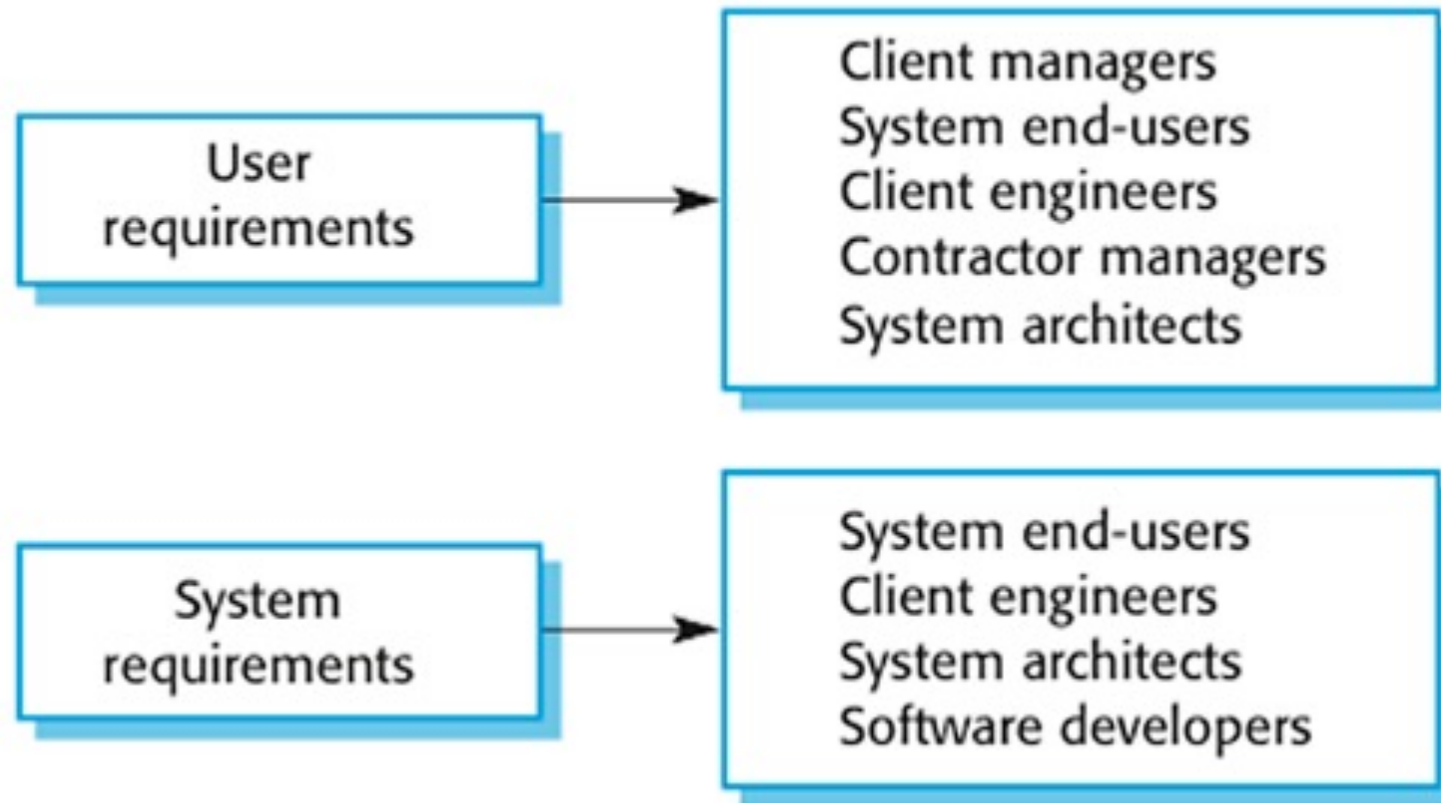
SYSTEM REQUIREMENTS

A more **detailed** description of the system **services** and the **operational constraints** such as how the system will be used, and development constraints such as the programming languages.

These are the traditional “***shall***” statements that describe what the system “***shall do.***”

They add **detail** and **explain** how the **user requirements** should be provided by the **system**. **They shouldn't be concerned with how the system should be implemented or designed.**

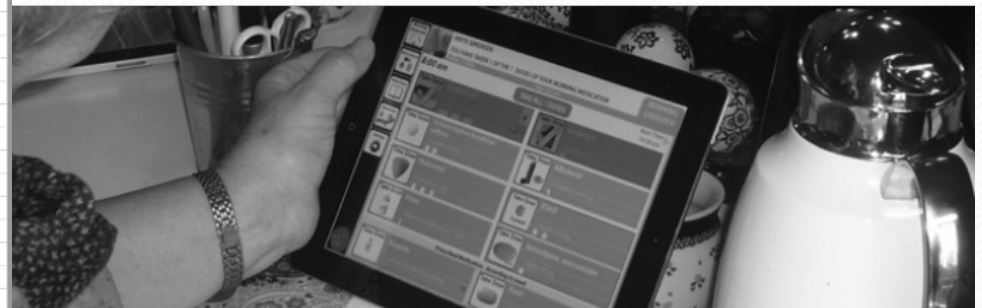
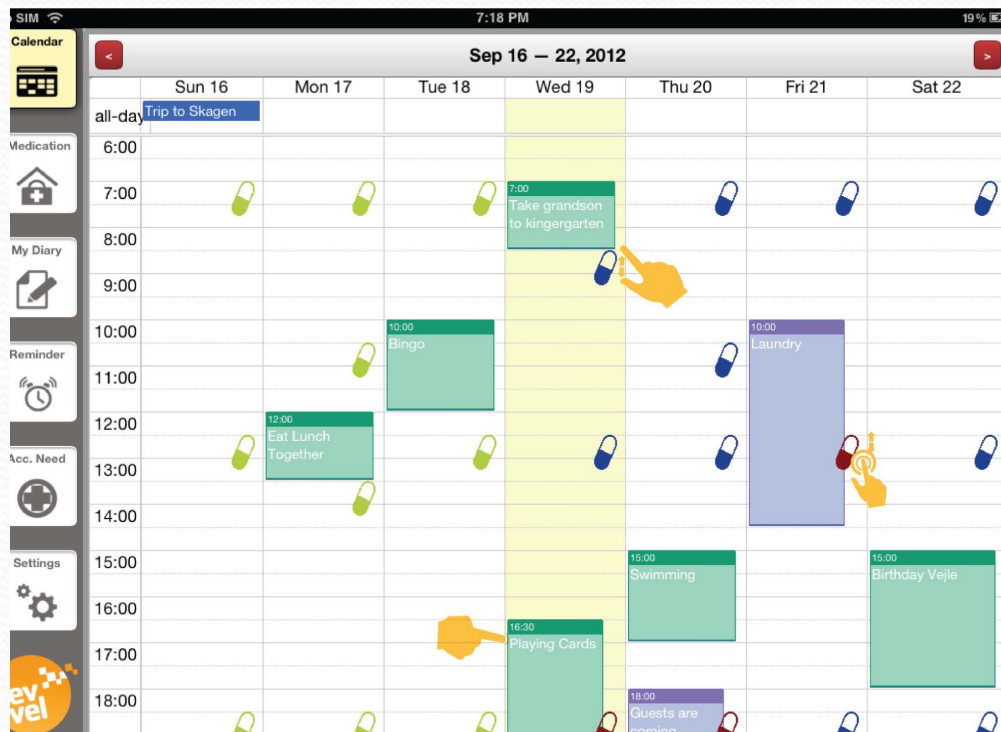
USER VS SYSTEM REQUIREMENTS -> DETAILS



- In "user requirements" the user is a subject, and the program being developed is an object.
- In "system requirements" the program being developed is a subject

FUNCTIONAL REQUIREMENTS

MediFrame should provide customizable reminders to each person's medication regimen and in accordance with their daily activities including the medication refill reminder.



Example

What the customer wants

Todd and Gina want more than a normal doggie door. They have everything from the TV to the music system to the garage door operating off remote controls. They want a way of letting their dog in and out of the house without having to walk to the door and open it themselves.

They have contacted Doug dog doors for a door and now Doug want us to build them the system to operate the dog door of Todd and Gina's dreams



Tired of cleaning up your dog's mistakes?
Ready for someone else to let your dog outside?
Sick of dog doors that stick when you open them?

It's time to call...

Doug's Dog Doors

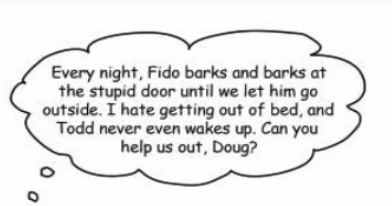
- ★ Professionally installed by our door experts.
- ★ Patented all-steel construction.
- ★ Choose your own custom colors and imprints.
- ★ Custom-cut door for your dog.



FROM WORDS TO WANTS

Todd and Gina:

Todd and Gina want more than a normal doggie door. Todd has everything from his TV to his music system to his garage door operating off remote controls. They want a way of letting their dog in and out of the house without having to walk to the door and open it themselves. They have contacted Doug dog doors for a door and now Doug wants us to build them the system to operate the dog door of Todd and Gina's dreams



Todd and Gina

Your dog door requirements list:

A

B

C will use Mongo DB

D

NO
This is a HOW not a
WHAT

User requirements version 1

Todd and Gina's Dog Door

Not a functional requirement's List

1. The dog door opening must be at least 30cm tall.
2. A button on the remote control opens the dog door if the door is closed, and closes the dog if the door is open.
3. Once the dog door has opened, it should close automatically if the door isn't already closed.

How not what

Conflicting and Ambiguous

USE SCENARIO



Todd and Gina's Dog Door

What the Door does

1. Fido barks to be let out.
2. Todd or Gina hears Fido barking.
3. Todd or Gina use your system.
4. The dog door opens.
5. Fido goes outside.
6. Fido does his business.
 - 6.1 The door shuts automatically.
 - 6.2 Fido barks to be let back inside.
 - 6.3 Todd or Gina hears Fido barking (again)
 - 6.4 Todd or Gina use your system.
 - 6.5 The dog door opens (again).
7. Fido goes back inside.
8. The door shuts automatically and locks.



USE A SCENARIO – VERSION 2



Todd and Gina's Dog Door

What the Door does

1. Fido barks to be let out or in.
2. Todd or Gina hears Fido barking.
3. Todd or Gina use your system.
4. The dog door opens.
5. Fido goes through the door.
6. The door shuts automatically.
7. The door locks



Prioritise

- in order of importance
 - **MoSCoW** –
 - **M**ust have,
 - **S**hould have,
 - **C**ould have,
 - **W**ont have

Help: reveal hidden assumptions, project planning and rescheduling if this becomes necessary.

Example Requirement

- **Dog Door system**
- Requirement 1. (Must)
 - Requirement : Todd and Gina **MUST** be able to open the Dog Door.
 - Acceptance Criteria.
 - On using the system the Dog Door flap opens.
 - The Door should remain open for a time
 - The Door should then close
 - Once closed the Door Must lock.

Example Requirement

- **Dog Door system**
- Requirement 11. (Could)
 - Requirement : Todd / Gina COULD be able to alter the time between the door opening and then closing.
 - Acceptance Criteria.
 - Todd / Gina selects to change the timing.
 - System allows Todd / Gina to enter new time delay – number of minutes between 1 and 60.
 - Todd / Gina asked to confirm new time.
 - System confirms that timing has been altered.

Example Requirement

- **Dog Door system**
- Requirement 15.(Won't)
 - Requirement : The system will not (Won't) respond to the bark of a dog as a signal to open the door.

GOOD REQUIREMENTS ARE



Not written
down too early



Unambiguous
and testable



Clear and
Concise



Consistency

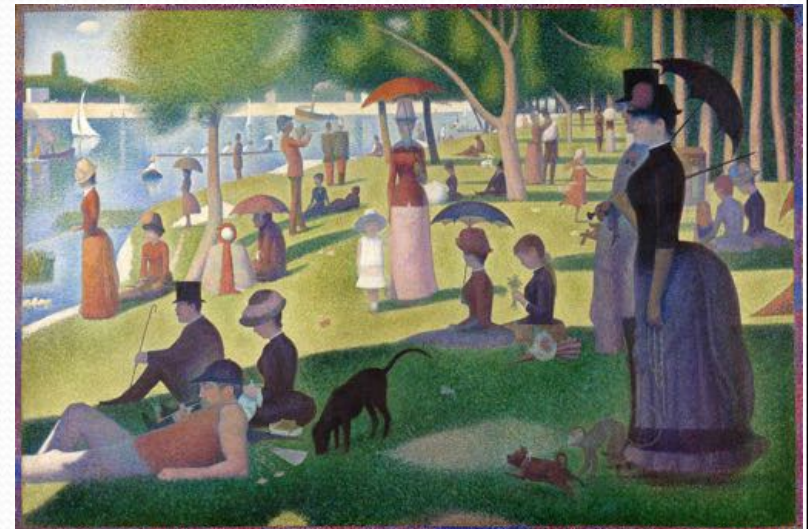


Attainable and
necessary

HOW TO WRITE GOOD REQUIREMENTS

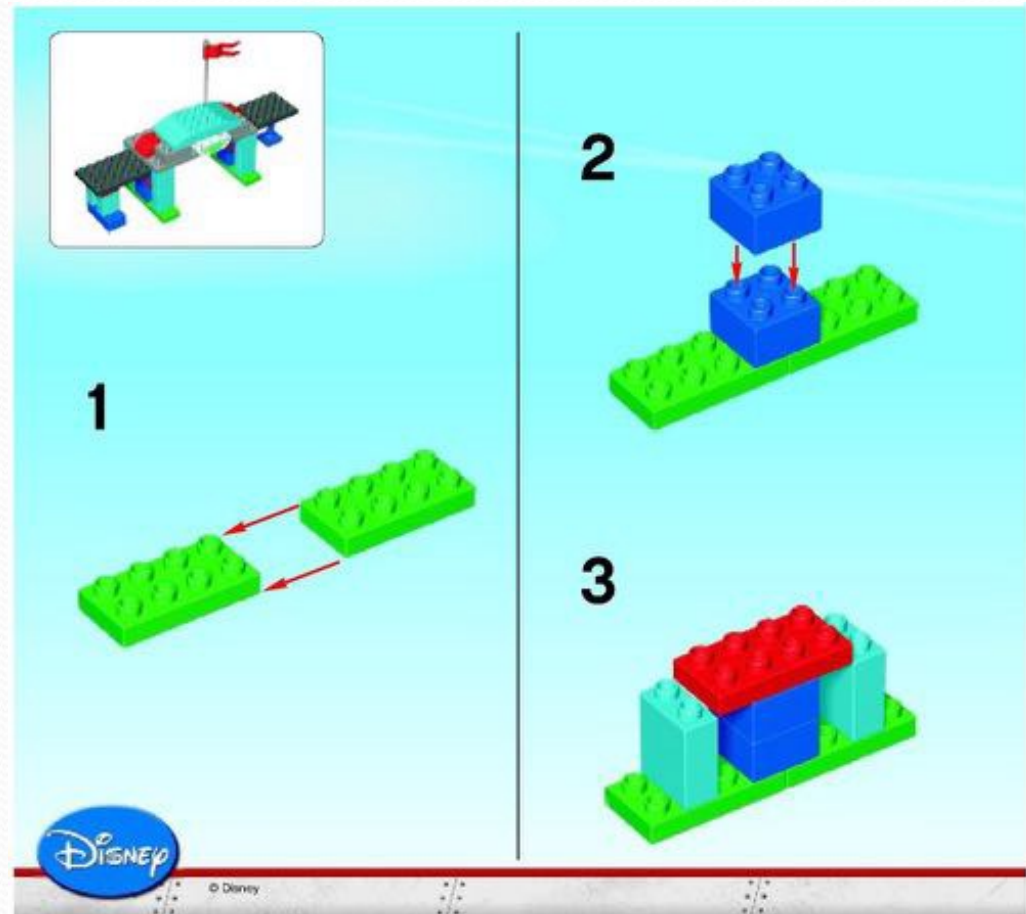
A user requirement is good, if it is:

1. **Verifiable** (it can be tested)
2. **Clear** (easy to read and understand by non technical people)
 - Leaves no one guessing (e.g., for how long? Central point?)
3. **Concise** (no more than 30-50 words in length)
4. **Coherent** (logical and consistent)
5. **Unambiguous** (can't be interpreted in multiple ways)
 - Avoid vague words (e.g., some)
6. **Implementation-free** (must not contain software or technology design decisions)
 - **DO NOT** describe the user interface in words
7. **Traceable** (unique identity or number)
 - Cannot be broken into smaller requirements
 - Can easily be traced



GOOD REQUIREMENTS ARE:

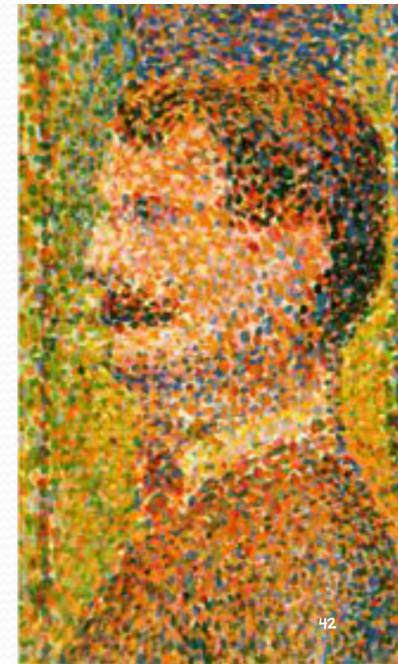
- Each requirement must describe **1 thing** only
 - And probably **1 small detail** about **1 thing**.
- Requirements are:
 - Instructions
 - Testing Criteria



REQUIREMENTS ARE LIKE IMPRESSIONIST PAINTINGS

- It is made up of tiny dots
- Each dot adds a small detail to the picture
- Together the dots describe the scene.

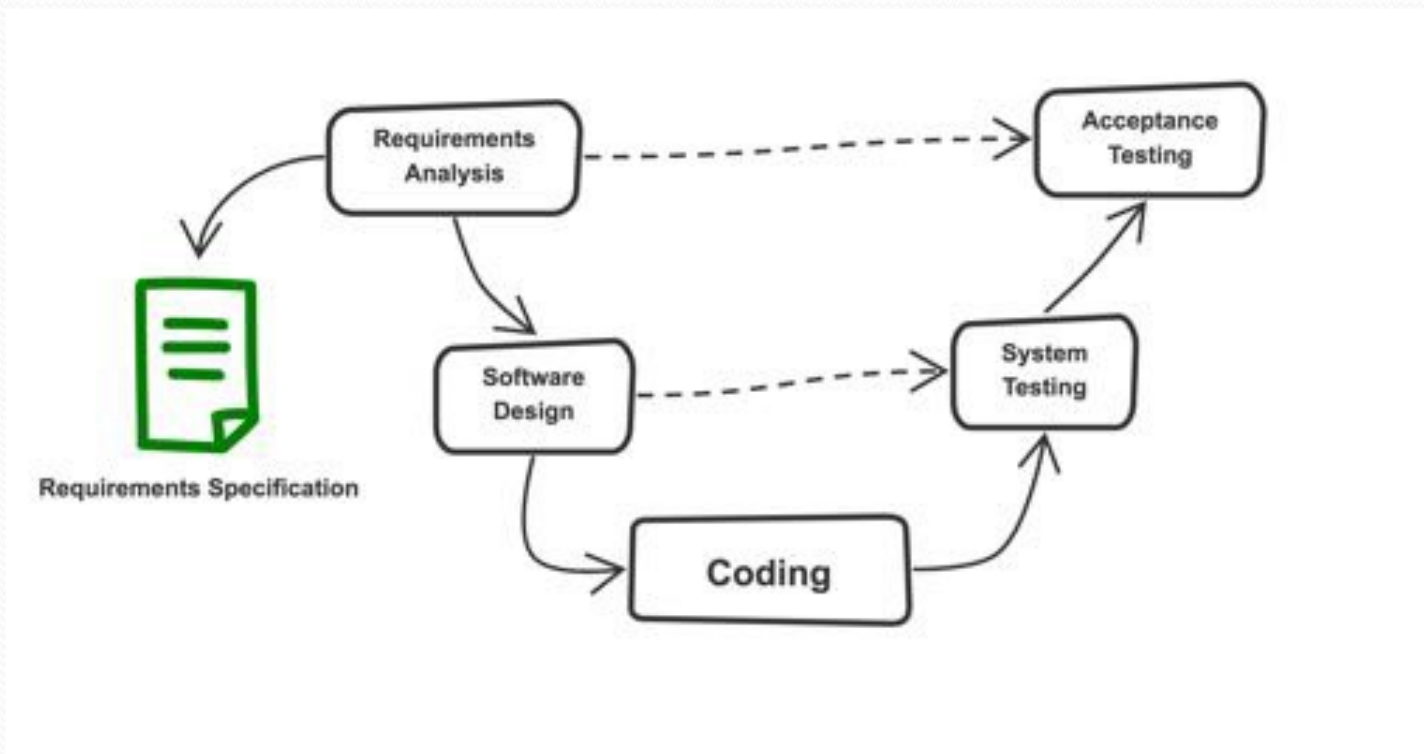
Each **requirement** is a tiny dot. They come together to form the complete picture



REQUIREMENT SPECIFICATION

Documenting the understanding of people's needs

- The “**instructions**” to the development team
- The “**acceptance criteria**” for the final software/system
- Into a document called **Requirement Specification** (one or more)
 - Shared, agreed description for everyone



FUNCTIONAL REQUIREMENTS

Requirement ID	Requirement Statement	Must/Want	Comments
FR001	The software automatically validates customers against the ABC Contact Management System	Must	
FR002	The Sales system should allow users to record customers sales	Must	
FR003	Only Managerial level employees have the right to view revenue data	Must	
FR004	The system shall provide a list of links to organizations related to ours	Want	Town website,
FR005	The system shall provide the contact information for every client in the store	Must	Name, role, email, phone
FR006	Members shall be able to renew their membership via a credit card using the website	Must	Paypal

SUMMARY

Describe what a software does, and the problems it solves or the benefits it gives

Based on:

- Understanding of user's needs -> user and system requirements
- What can technology do to satisfy those needs?
 - Describing all functions and features
(e.g., comprehensible and without ambiguities)
 - Scenarios and prototyping

ACTIVITIES WEEK 6

- Watch/Listen to the YouTube video ON "How to Write High Quality Requirements".
- Review the articles (Learning Materials Week 6)
 - Requirements Engineering: A Roadmap
- Prepare a 0.5 A4 page document providing a critical review of each article
 - What did you learn from reading the article?
 - What questions do you have from reading them? Do you agree with each article? Why?

Requirements Engineering: A Roadmap

Bashar Nuseibeh
Department of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ, U.K.
Email: ban@doc.ic.ac.uk

Steve Easterbrook
Department of Computer Science
University of Toronto
6 King's College Road
Toronto, Ontario M5S 3H5, Canada
Email: sme@cs.toronto.edu

ABSTRACT

This paper presents an overview of the field of software systems requirements engineering (RE). It describes the main areas of RE practice, and highlights some key open research issues for the future.

1 Introduction

The primary measure of success of a software system is the degree to which it meets the purpose for which it was intended. Broadly speaking, *software systems requirements engineering* (RE) is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation. There are a number of inherent difficulties in this process. Stakeholders (including paying customers, users and developers) may be numerous and distributed. Their goals may vary and conflict, depending on their perspectives of the environment in which they work and the tasks they wish to accomplish. Their goals may not be explicit or may be difficult to articulate, and, inevitably, satisfaction of these goals may be constrained by a variety of factors outside their control.

In this paper we present an overview of current research in RE, presented in terms of the main activities that constitute the field. While these activities are described independently and in a particular order, in practice, they are actually interleaved, iterative, and may span the entire software systems development life cycle. Section 2 outlines the

1 provide the foundations for effective RE,
2 3 briefly describes the context and
3 ded in order to begin the RE process.
ugh 8 describe the core RE activities:

digital or hard copies of all or part of this work for
om use is granted without fee provided that copies
tributed for profit or commercial advantage and that
fice and the full citation on the first page. To copy
ish, to post on servers or to redistribute to lists,
ific permission and/or a fee.
e Engineering Limerick Ireland
X00 1-58113-253-0/00/6...\$5.00

- eliciting requirements,
- modelling and analysing requirements,
- communicating requirements,
- agreeing requirements, and
- evolving requirements.

Section 9 discusses how these different activities may be integrated into a single development process. We conclude with a summary of the state of the art in RE, and offer our view of the key challenges for future RE research.

2 Foundations

Before discussing RE activities in more detail, it is worth examining the role of RE in software and systems engineering, and the many disciplines upon which it draws. Zave [83] provides one of the clearest definitions of RE:

"Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families."

This definition is attractive for a number of reasons. First, it highlights the importance of "real-world goals" that motivate the development of a software system. These represent the 'why' as well as the 'what' of a system. Second, it refers to "precise specifications". These provide the basis for *analysing* requirements, *validating* that they are indeed what stakeholders want, *defining* what designers have to build, and *verifying* that they have done so correctly upon delivery. Finally, the definition refers to specifications' "evolution over time and across software families", emphasising the reality of a changing world and the need to reuse partial specifications, as engineers often do in other branches of engineering.

It has been argued that requirements *engineering* is a misnomer. Typical textbook definitions of engineering refer to the creation of cost-effective solutions to practical problems by applying scientific knowledge [74]. Therefore, the use of the term *engineering* in RE serves as a reminder that RE is an important part of an engineering process, being the part concerned with anchoring development activities to a real-world problem, so that the

☐ Week 6 - Discussing the Requirements Engineering Roadmap paper

Discussion thread for the article in week 6.

To discuss, focus on answering the following questions:

- What did you learn from reading the article?
- What questions do you have from reading them?
- Did you enjoy the article? Why?