User Interface Design & Evaluation

Requirements

Overview

- User experience
- The user-centred design process
- Requirements

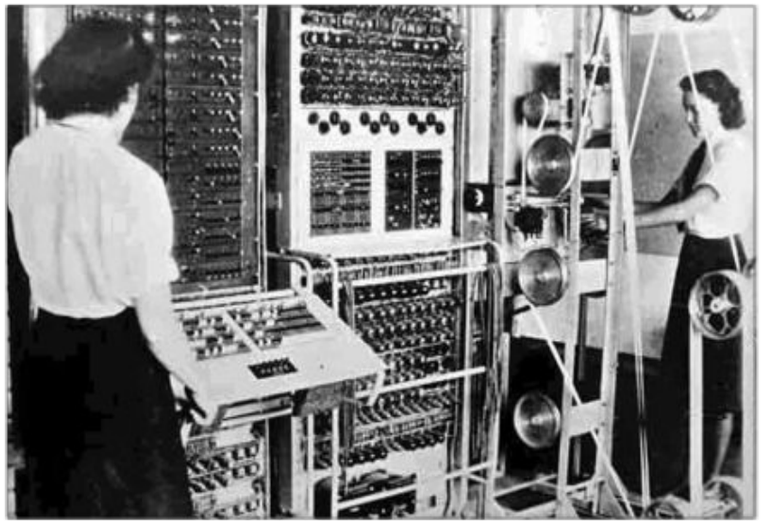

You are here ☺


You are here ☺

Usability

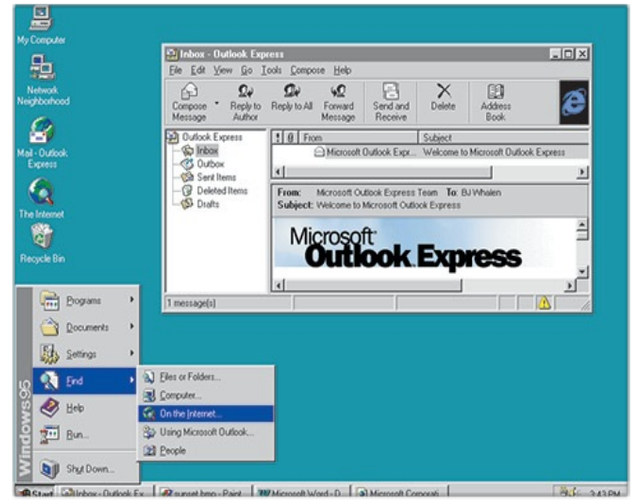
- Effectiveness
- Efficiency
- Safety
- Utility
- Learnability
- Memorability

Beyond usability: User experience



Colossus

Apple IIe



Windows 95

Windows 8

iPad Air

USER EXPERIENCE GOALS

Desirable:

Satisfying
Enjoyable
Engaging
Pleasurable
Exciting
Entertaining
Helpful
Motivating
Challenging
Enhancing sociability

Supporting creativity
Cognitively stimulating
Fun
Provocative
Surprising
Rewarding
Emotionally fulfilling

Undesirable:

Boring
Frustrating
Making one feel guilty
Annoying
Childish
Unpleasant
Patronizing
Making one feel stupid
Cutesy
Gimmicky

USER-CENTRED DESIGN PROCESS

Requirements: Conceptual model

Design: Interaction design

Prototype: Lo-fi and hi-fi methods

Evaluation: Visual design UI
Heuristics …
Experiments?

Users' needs
Users and personas
Uses and scenarios

REQUIREMENTS

Specifying what the system should achieve

Requirements gathering and analysis is almost always the first step in software engineering

Important to get this right at the beginning

Much easier, cheaper and quicker to fix problems earlier rather than later

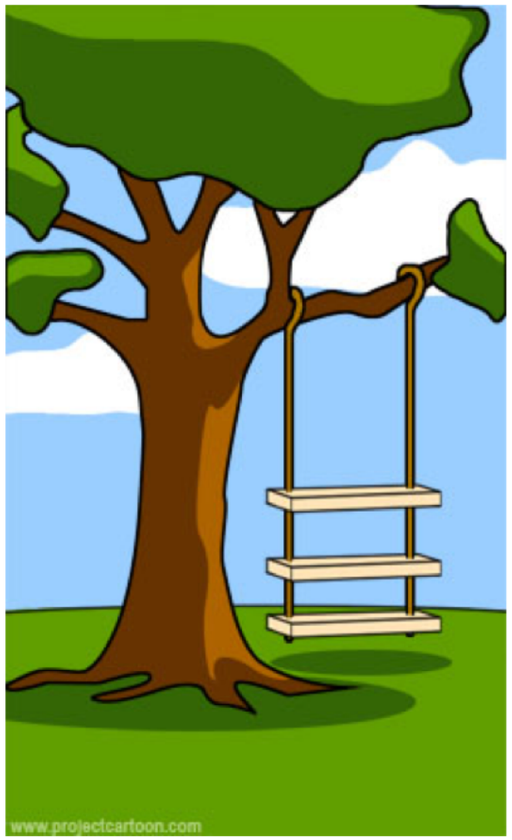
COMMON ISSUES

Customers don't really know what they want

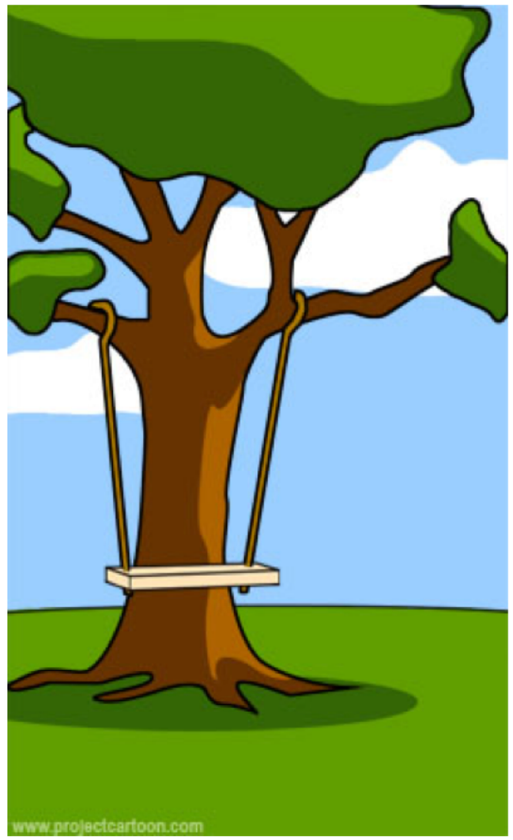
Requirements change and adapt over the course of a project

Miscommunication/misunderstandings between stakeholders

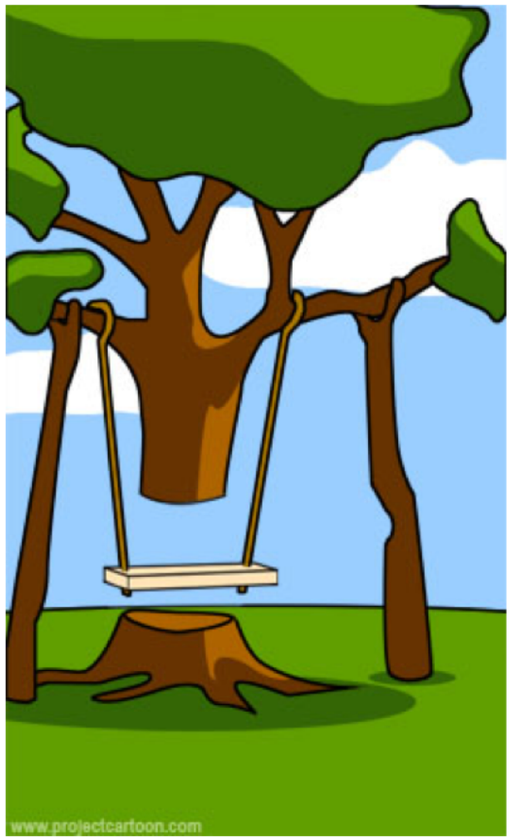
Misunderstandings



How the customer explained it



How the project leader understood it



How the analyst designed it

What the customer wanted

The Purpose of Requirements

- Can act as a form of contract between the provider and their stakeholders.

- Describe what people want from their systems as well as what the systems have to do for them.

- Fulfills a need of some kind that is currently missing.
  - In the UK MoD this is called a ‘capability’

What makes a good requirement?

What quality attributes (QAs) should requirements possess?

Requirements should be:

- Traceable
- Precise and Unambiguous
- Correct
- Complete
- Verifiable
- Atomic
- Consistent
- Design independent
- Feasible
- Flexible

QA: Traceability

Backwards ("traced"):

- Every requirement should be linked to its source
  - The person or document demanding the requirement
  - This link needs to be properly documented!

- Non-traceable requirements might be
  - when assessing inappropriately recorded
  - Incorrect or misleading
  - Ignored the impact of change requests

QA: Traceability

Forwards (“traceable”):

Every requirement should be identifiable

So that it can be referenced later

QA: Preciseness and Unambiguousness

People use Natural Language:

Natural language can be understood by all stakeholders
Natural language can be satisfactory for some aspects

Needed to help explain things

Problems of natural language:

Too imprecise for describing core engineering artifacts
Can be badly structured
Can be ambiguous

QA: PRECISION

- All requirements should be stated clearly, to the point

- NASA Requirement for Command and Control module:

  While acting as the bus controller, the C&C MDM CSCI shall set the e,c,w, indicator identified in Table 3.2.16-II for the corresponding RT to “failed” and set the failure status to “failed” for all RTs on the bus upon detection of transaction errors of selected messages to RTs whose 1553 FDIR is not inhibited in two consecutive processing frames within 100ms of detection of the second transaction error; if a backup BC is available, the BC has been switched on in the last 20s, the SPD card reset capability is inhibited, or the SPD card has been reset in the last 10 major (10s) frames, and either…

QA: PRECISION

Precise but unnatural language (badly structured)!

QA: Unambiguousness

- All requirements should have only one interpretation
- In practice this implies setting up a glossary as well
  - All terms used must be explained
    - But a glossary, by itself, does not prevent ambiguity…
- Example of ambiguity: Signs at the foot of an escalator

Examples of natural language ambiguities

Some newspaper headlines:

"Rioter Sentenced to Six Months in Violin Case"

Examples of natural language ambiguities

"Miners Refuse to Work After Death"

QA: Verifiability

Requirements must be testable/verifiable

- Are they stated in such a way that test engineers can devise tests which can show if the system meets the requirements?

May be directly testable:

- **Simple user interface – test criterion: “<15 options available"**
- Count the options!

May be indirectly testable:

- Simple user interface – test criterion: “a typical user rates it as simple"
- Set up a test session and get user ratings

Fit criteria make sure that requirements are testable!

QA: Atomicity

No combined requirements

- Does the description of a requirement describe a single requirement or can it be broken down into several different requirements?

Problems of combined requirements:

- May have different priorities
- May be implemented differently
- May not be tested thoroughly

One indication of combined requirements:

- If several separate fit criteria are needed, then perhaps there are several requirements

QA: Consistency

Divergence from business goals

Is the requirement inconsistent with the business goals?
Contradiction and conflict
Explicit – incompatible goals
Implicit – e.g., "minimise cost", "minimise time", and “maintain full functionality”
Emergent – e.g., it is difficult to design ease-of-use
Different stakeholders are often source of conflict
Example: Reliability vs. Safety
One channel design (vs. two channel design) is more reliable but less safe

Fewer components give more reliability
Lack of redundancy gives single point of failure

QA: DESIGN INDEPENDENCE

Premature design, or over-specification

The requirement includes premature design or implementation information and describes a solution rather than the problem

Classic fallacy:

“High level details belong to requirements definition"

“Low level details belong to design"

Detail can apply to requirements model or design model

DESIGN INDEPENDENCE EXAMPLES

Consider a ship's pump that may need to fill vessel to 3.753 plus/minus .004 meters

Low level detail, but technology independent and therefore a requirement

Use microprocessor rather than analog-control circuitry

High level detail, but technology dependent and thus a design issue

QA: FEASIBILITY

Requirements must be feasible

Wishful thinking,

i.e., a gap between

What the stakeholders would like to achieve, and

What can actually be achieved in practice

No wishful thinking

Is there a feature that cannot possibly be implemented?

Is the requirement realistic given the technology which will be used to implement the system?

Feasibility must be recorded

A requirement's fit criterion should make sure that the requirement is feasible

In exceptional cases, feasibility may be in doubt until system development is complete

TYPES OF REQUIREMENT

Traditional division

Functional
What the system should do – operations it must support
e.g. Open account, display drive size, schedule delivery

Non-functional
Constraints on the system and the development process
e.g. performance, security, system delivery deadlines, code standards, anything ending in -ability

Single Statement of Need (SSON)

A clear, concise statement about what is the overall goal of the system and how it will accomplish those goals.

Usually determined by the client/sponsor early on.
Although it is often poorly done.

Describes what capability the system being developed will provide.
Provides a broad understanding of what the final outcome of the endeavour is going to be.

Can act as a long term measuring stick regarding how well the requirements process has been completed.
– If you as a requirements engineer cannot look back at the SSON and decide whether your requirements describe a system that meets this need you have failed your stakeholders.

Example of an SSON: Blizzard

"[A goal of this] 'starter' dungeon is that you and your party can have a solid, memorable experience with plenty of story payoffs and some nice rewards to boot -- all in a single run."
– Blizzard Development Team, Wrath of the Lich King

(http://www.worldofwarcraft.com/wrath/feature s/dungeons/utgarde.xml)

Gives a clear idea of what they want to accomplish, but how are they going to do it?

<DND>Lapp
Sejta
Xodan
Maeil
Jhazrun
Kruf
Jubeto
Kahva
Dilivil
Despinarx
Verdisha
Tauntie
Ussoß
Synti
Zhinn
Wakozi
Tsukky
Xenophics
Lazeil
Rakez
Liha
Diamondbear
Iiris
Mancheese
Ilonie

Example of an SSON: Blizzard

How will the dungeon achieve the goal?

"What should make the experience of playing through Utgarde Keep even more engaging is a sense of purpose for the players, and even greater connection to the story than before ... one of the most important [things] is to establish easily identifiable goals. When you feel like you're there for a clear reason, that sense of purpose makes a dungeon far more engaging."

Clear, concise information about how they will make the game more engaging. String it together, and you have a nice SSON for dungeon design in Wrath of the Lich King.

User Requirements

Statements regarding the tasks of users that should be satisfied by the target system.
Driven from the high level goals of the users in organizations

Users as stakeholders should be engaged as early as possible in the requirements process – Immediately after a need is recognized.

Written for non-technical people involved in the requirements process.

System Requirements

A description of how the system will deliver on the needs of the users.

Detailed descriptions of functionality, services and constraints.

Written for technical implementations.
Multiple system requirements will often satisfy a single user requirement.

User to System Requirements

User Requirement

The game must provide a means of securing a player's characters and possessions against malicious misuse.

System Requirements

The game will provide authentication of a user name.

The game will secure password information with one-way encryption.

The game will secure password information retrieval with provision of a Q&A scheme.

Types of Requirements

Type taxonomy for requirements:

Functional requirements

- Things a system must do
- An action that the system has to take if it is to provide useful functionality for its user

Non-functional requirements (NFRs)

- Qualities a system must have
- Constraints on functional requirements
- Often critical to a system's success

Constraint requirements

- Global issues that shape the requirements
- Preferably defined at the beginning of gathering requirements

Functional Requirements

Transformation: Required response to a condition/event
– E.g., “customer inserts card” shall lead to “request PIN” in the description of a cash machine

Invariant: Property that must always hold
– E.g., “angle of attack (angle between airplane wing and airflow) shall never exceed 27 degrees”

Failures: Forbidden/permissible transformations
– Failure modes to be avoided, such as failures leading to safety hazards
– Failure modes to be allowed

Non-Functional Requirements

NFRs constrain individual functional requirements

Some examples: (NFRs are fully discussed next week)

Optimisation

– Maximise/minimise some property, e.g., “minimise power consumption”

Reliability/Availability

– Statistical qualification, e.g., invariant “shall hold 99% of the time”

Timing

– Temporal constraints on transformation, e.g., “request PIN within 2s”

Precision constraint

– Constraint on measurement of property, e.g., “calculate angle of attack to 1 degree”

NFRs: Safety and Security Requirements

Safety requirements

Have to be separately identified and managed
Must be signed off by a **regulatory body** (Health & Safety Executive)
Are stated in a **safety case** (incl. solutions & justification)
Are focused on the need to identifying **hazards**

Security requirements

Need to be identified and managed
Must (probably) be signed off by a **regulatory body** (Financial Services)
Are typically expressed by a **security policy**
– Project constraints, such as constraints re: confidentiality of data – Process constraints, such as demanding a security analysis

Constraint Requirements

Constraint requirements apply to the entire system
Examples of system-wide constraints:

Project constraints
– Financial constraints: “spend less than £5m”

Process constraints
– Standardisation constraints: ‘use Def Stan 00-55’ (standard for the development of safety critical software systems in defence equipment)

Design constraints
– Technology constraints: “shall run on a personal computer”

Fit Criteria for Requirements

A good quality requirement consists of two parts:
– A description capturing a stakeholder's intention
– A fit criterion that quantifies/measures this intention

The quantification of a requirement is its **fit criterion**

Examples (using a functional and a non-functional requirement):
– Description: The product shall record the weather station readings.
– Fit criterion: The recorded weather station readings shall match those sent by the weather station.
– Description: The product must produce the road de-icing schedule in an acceptable time.
– Fit criterion: The road de-icing schedule shall be produced within 15 seconds for 90% of the times that it is available; and within 20 seconds for the remaining 10%.

TYPES OF REQUIREMENTS FOR UCD

Functional
Non-functional
Data
Context of use
Physical, Social, Organisational,
Technical
User characteristics

Functional requirements
– What the system should do

Non-functional requirements
– How well the system should do it
– Cannot be expressed as functions of the system – Can be regarded as the constraints on a system

Safety and security requirements
– What the system should not do
"Do this, and nothing else"

NFRs are difficult to classify, quantify, and elicit

Classifications for NFRs

1. Process-tools,standards,...
2. Product-usability,reliability,safety,... 3. External-legal,environmental,...

1. “Look and Feel” - the product’s appearance
2. Usability–effectiveness,efficiency,satisfaction,learnability,
3. memorability
4. Performance-safety,speed,capacity,accuracy,...
5. Operational-operatingenvironment
6. Maintainabilityandportability
7. Security
8. CulturalandPolitical
9. Legal

“Aspects”

- Functional aspects - what it does – Stakeholder views
  - Usability aspect - how people will use it
  - International aspect - where people will use it
  - Performance/reliability aspect - how well it does it
  - Environmental aspect – what is around it
  - Legal aspect - is it allowed to do it
  - Safety and security aspects - what it does not do [see next lecture]
  - Development aspect - how it gets built, how much that costs, ...
  - Deployment/installation aspect - moving from development to use
  - Maintenance aspect - what happens after deployment
  - ...

Usability Aspect

Effectiveness

– How well can the users complete their tasks with the system?

• Efficiency

– How efficiently (by some metric/measurement) are tasks performed?

• Learnability

– How easy is the system to learn?

• Memorability

– How well can tasks be repeated after training?

• Satisfiability

– “Look and feel”

– Use of graphics, colour, sound – The product’s appearance

International aspect

language
– Fonts, accents, text direction, menu position – Text length
– Translation
• All your base are belong to us - Zero Wing (Sega Megadrive game)
•Dates and times
  • – Format (04/05/1999 - 4th May 1999 (UK), April 5th 1999 (US)
  • – Time zones (mobility problems)
•International laws
•– UK Data Protection Act vs. US Right of Access

Performance/Reliability Aspect

- Speed
- Capacity
- – Throughput, latency, bandwidth – Storage, persistence, distribution – Efficiency of resource usage
- Accuracy, precision
- Reliability
- – Robustness - failover
- – Recovery, disaster response
- Availability
- – Resilience, fault tolerance, response time – Operational hours, downtime

Legal Aspect

- Safety and Security
- Data Protection Act
- – Who has the disk drive today?
- – Do you really need all data about everyone on one CD?
- Anti-discrimination laws
- – Accessibility
- People with disabilities, older adults
- – Ethnic access • Language
- Accounting laws
- – VAT, Stock Market rules – Audit trails

– Accountability!

Development Aspect

- Process and tools to be used
  - – Requirements and analysis process
  - – Design process – e.g., Rational Unified Process/UML
  - – Implementation language(s), development software packages
  - – Testing
  - – Source code control system, versioning
  - – Backups, documentation, ...
- Project management
  - – Reporting, accounting
  - – Change management
  - – Risk management: reqts. volatility, external dependencies
  - – Sign-off authorisation
- Development standards

– ISO 9000, ...

Deployment/Installation Aspect

Maintenance Aspect

Scope of NFRs

- System wide
  - Product requirement
    - System performance “never more than 50% CPU load” – Process requirement
    - Use IEEE Std 1012 for software validation & verification (V&V)
  - External requirement
    - Cost of development must not exceed £500,000
- Single function NFRs
  - Associated with a specific functional requirement
    - Slot provided in the Use Case template

Summary

Non-functional requirements are often more crucial to a project's success than functional requirements

Good quality requirements are important but cannot be obtained straightforwardly

Properly documenting requirements and their sources provides the foundation for being able to accommodate future change requests