

08-04-23

Software Engineering

CSC 648/848

Fall 2023

Use Cases, Requirements and Specs

Prof. Dragutin Petkovic
Anthony Souza

Modern SE development – make it user centric

Start with user needs (personae, use cases)
then design and develop systems to satisfy
those needs

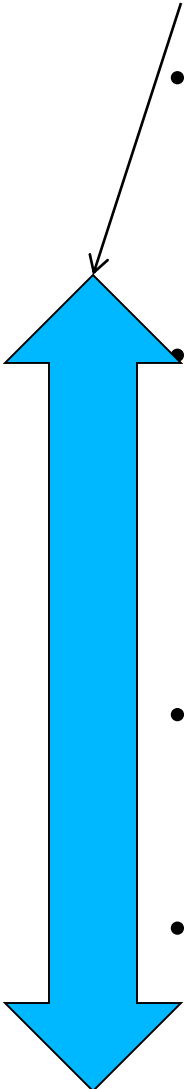
(NOT the other way around)

Iterate with users in the loop

SW Dev. Pipeline: Use Cases,

Important

Iterations!!!! Requirements, Specifications

- 
- **Personae and Use Cases:** describe who are the main actors (*“personas”*) and then a story (*“use cases”*) of *WHAT services/functions users will need*, in plain English. Focus on **WHAT** not **HOW**. Involve marketing, sales, competitive study, focus groups
 - **Requirements:** developed from use cases, usually by marketing, engineering, sales (*focus on WHAT not how*):
 - **Functional** (what functions the SW has, what does it do)
 - **Non-functional** – what properties the SW has (security, privacy, performance, deploy in Cloud? etc.)
 - **Specifications** – more concrete engineering guidance of **HOW** to develop the SW (often combined with requirements into SW requirement specifications - **SRS**)
 - Followed by **architectural design, implementation, testing in an iterative fashion**

Start with Personas – a way to represent user types for your use cases and UI design – and testing

From Interaction Design Organization, downloaded 05/28/18

- <https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them>

Personas are fictional characters, which you create based upon your research in order to represent the different user types that might use your service, product, site, or brand in a similar way. Creating personas will help you to understand your users' needs, experiences, behaviours and goals. Creating personas can help you step out of yourself. It can help you to recognise that different people have different needs and expectations, and it can also help you to identify with the user you're designing for. Personas make the design task at hand less complex, they guide your [ideation](#) processes, and they can help you to achieve the goal of creating a good user experience for your target user group.

Personae and Use cases

- Used as a starting point for the design of full systems, and not just for UI components
- First thing to do before discussing more detailed requirements and functions
- Starting from functions (and not doing personae and use cases) may cause you to forget important functions

Personas

Important

Personas have:

- Attitude
- Skills
- Limitations
- Pain points
- Goals

Personas go over scenarios (use cases) using your application in the appropriate context (home, on the road...)

Create one page persona description (with image) to be handy during the design – some examples below

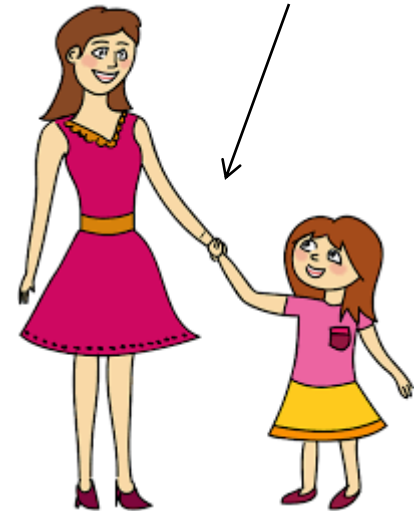
- https://www.google.com/search?q=ui+design+persona&hl=en&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjmu_Wv6KjbAhUfIDQIHTHTD2YQ_AUICigB&biw=1440&bih=782

Example persona: Mary– mother and part time worker

Add some image to
make it more personal

About Mary:

- Very busy
- Loves to take kids to park
- Environmentally conscious: likes to take kids to environmentally safe parks; will report environmental issues
- Not too patient with WWW apps
- Has basic WWW skills



Goals and scenario:

- Decided to go to a park. Wants to quickly check (from home) if park is safe re: environmental issues
- If she finds a problem in the park, she likes to report it but does not have much time to do it.

Important

From Personae to Use Cases

They are major driver for requirements and specs
and architecture

(the other driver is business/client needs)

Design systems to satisfy use cases (e.g. user needs)
not the other way around

Think of the user first, and also of client/business

Personae vs. use cases

Persona: *general* characteristics of the users related to your applications

Use cases: *What services and functions* personae will use your envisioned system

Example Use Case – e-TAT – teamwork assessment tool – free text

John is the **instructor** of SE class. He has already adopted the recommended **learning materials and methods**. However, he also needs to assess and monitor student teamwork during the class. John first sets up the recommended **class tools** (for collaboration and SW development) and forms the student teams using the learning and teaching material instructions, resources and tools. He downloads, installs and familiarizes himself with **e-TAT** using recommended tutorials and documentation. Using the **admin module** of e-TAT John initializes the **student teams**. Based on his class objectives, he modifies and customizes the **questionnaires** and **final presentation** data within the options provided by e-TAT.

(**BOLD** items should be defined in data glossary and have well defined and special meaning)

Example Use Case – Paper Form Entry – structured version

- Users skills: high school. Minimal training for the task (1 hour)
- Work setup: desk. Cluttered with a lot of paper, and not enough space
- Form (show picture)
- Work steps: Take paper from the pile, places at the middle of the table and copies entries into the form.
- Difficulties and errors: Paper is often lost. Drops from the table. Papers get mixed up.
- Performance: 10 minutes per form.
- Reliability: 5 errors per form on average. 10% forms are lost
- User satisfaction: Very frustrated. Tires easily. High staff turnover

Use cases

Important

- One of the most powerful methods. *Must do!*
 - Define **actors** in the application – **persona** – a model of a typical user e.g. define user skill level, pain points....
 - Describe **user tasks** and errors
 - Know frequency and importance of tasks
 - Describe user **environment and context**
 - Describe or use data items from data glossary – be consistent
- Methods for gathering information:
 - Ask users (single or team)
 - Observe users in their actual environment
 - Look at user documentation or any other information
 - Surveys, focus groups
 - Ask your own marketing, sales, support, client reps
 - Read WWW, product reviews
 - Ask your friends

Important

Use Cases >> Task analysis

- Task name, description and current duration as well as frequency of usage
 - User skill level, experience and role
 - Work steps including usage of other systems (printers, faxes...)
 - System features, GUI screens currently used
 - Difficulties and errors in using current system
 - Current systems performance and reliability
 - Current user satisfaction and “pain points”
 - Follow-up questions
 - Sample inputs, screens etc.
-
- Good to compare current and future (with your system) use cases in terms of number of steps, time to complete etc.

Use cases levels of detail

- *High level* – focus on WHAT the user does at high level (no details of HOW)
- *Lower-level* – more details, spell out functions, steps etc. (less used)
- High-level use cases are critical
 - Drive them by user needs NOT the constraints of the envisioned system. Compromise only if you must, be a user advocate.
 - Focus on WHAT user needs not HOW to design or implement

Usage of use cases

- Communicates vision of the system easily understood to non-technical people (CEO, marketing, sales, investors, business)
- Drives the design and requirements
- Use to test all aspects of the system, especially the UI
- Use to develop QA plans

Examples of bad high level use case, then a corrected one

Important

- Mary wants to search for environmental issues. She goes to our site, types “Golden Gate” in yellow search button which is upper right corner of the site, hits GO button then goes to the list of search results presented at the bottom of the screen in two columns

Too many UI design details (underlined) – it is too early for this!

Better one (leaves UI design details for later)

- Mary wants to search for environmental issues. She goes to our site and searches by park name e.g. “Golden Gate” then reviews the list of search results

Use cases even at high level drive UI design – e.g. lazy registration

- Mary wants to post a content on our site. She chooses the appropriate function from our www site, fills out and form and upon submission is prompted to register or log in

This way you ensure lazy registration implementation metaphor where user is prompted to register/login only when already busing the function – this is the right way

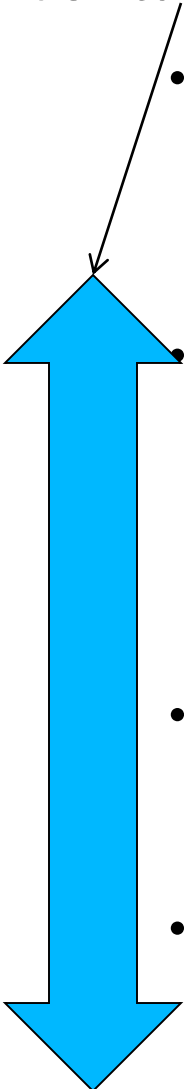
- <https://www.smashingmagazine.com/2009/06/10-ui-design-patterns-you-should-be-paying-attention-to/>

**Then: Personae > Use Cases >
Requirements**

Use Cases, Requirements,

Iterations!!!! Specifications

Important

- 
- **Personae and Use Cases:** describe who are the main actors (“*personas*”) and then a story (“*use cases*”) of *WHAT services/functions users will need*, in plain English. Focus on WHT not HOW. Involve marketing, sales, competitive study, focus groups
 - **Requirements:** developed from use cases, usually by marketing, engineering, sales (*focus on WHAT not how*):
 - **Functional** (what functions the SW has, what does it do)
 - **Non-functional** – what properties the SW has (security, privacy, performance, deploy in Cloud? etc.)
 - **Specifications** – more concrete engineering guidance of HOW to develop the SW (often combined with requirements into SW requirement specifications - **SRS**)
 - Followed by **architectural design, implementation, testing in an iterative fashion**

Software Requirements

- Descriptions of a SW system – starting point for specifications, design and implementation
- One of the main and early component in any SE process – focus is on WHAT not HOW
- Main vehicle of communication between:
 - Idea initiator and the rest of the world
 - Clients and SW providers
 - Customer and product development (SW, marketing, sales)
 - Product development and management
 - Product development and marketing
 - Between developers (especially at different locations)
 - Developers and QA

Good overview

http://en.wikipedia.org/wiki/Requirements_analysis

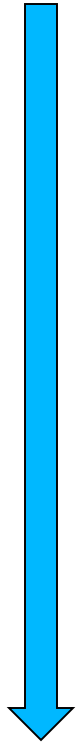
Important

What are Requirements

- Requirements: Abstract statements or descriptions of SW services, functions or constraints
- For high-level requirements focus is on WHAT not HOW
- Level of abstraction goes from high (usually called requirement definitions) to more detailed (usually called requirement specifications)

Format and types of Requirements

Formalism



- Implicit, not written, “in your head only” - usually SW done for your own needs (e.g. your homework; self-education; fun)
- Verbal understanding and communication (extreme programming, small team, early prototype, team project at school)
- A few pages, free format (startups, small company, school)
- Structured documents (commercial, format often fixed and specific to a company)
- Very formal and elaborate (commercial, medical, avionics, government)

Essential to ensure all have the same understanding of what needs to be done

And, they change in time!!!!

Req. Categories: Functional and Non-functional Requirements

- **Functional requirements**
 - WHAT DOES IT DO? Services and functions system provides. How will system react and behave depending on inputs. Specific functions provided to the user. Key data items
- **Non-Functional requirements**
 - HOW DOES IT DELIVER? Various constraints on timing, cost, ease of use, standards, security, performance, capacity, HW/SW environment, privacy etc.
- In some cases: **Domain requirements** (in case of certain app domains)
 - Comes from application domain and reflect specifics of that domain
 - Key in multidisciplinary projects e.g. bioinformatics

Functional and Non-functional Requirements **Important**

Functional requirements:

- ***Functions:*** Describe functions and services that system provides
- Also: ***Data glossary/description:*** describe key data items and entities in the system

Non-functional requirements:

- ***System requirements:*** describe the system requirements (architecture, system services, networks, platforms etc.)
- ***Usability requirements:*** describe specific usability and UI issues, users who will use the system, delivery clients
- ***Performance requirements:*** describe system performance (speed, accuracy, latency, delay, bandwidth etc.)
- ***Storage, security, environmental requirements***
- ***Marketing, legal requirements*** (logos, branding, licensing)
- ***Content*** (size, formats...)
- ***Privacy*** (what data is collected, how is it used...)

Functional Requirements

1) *Functions*: Describe functions and services provided by the system

Use SHALL

Not: will, may, should...

- Examples:

- Users shall be able to search using image categories
- Users shall be able to purchase music files using credit card
- Music shall be accessible by author and title

Note: may contains some usability requirements (in choices of functions available), but usability can be addressed also in non-functional requirements too)

Functional requirements

Important

- 2) *Data description*: Describe key entities and data items and give them unique and meaningful names
- Main actors/users in the system (admin, customer...) and their roles
 - Key data elements (book item, shopping cart, registration, search metadata). Important to describe *digital content* (see next slide)
 - For each have the name, sub-items, means of input/initialization, usage, storage etc.
- Use these names consistently for documentation, UI, reference, naming of classes and variables, database schema

Proper data description is the key for communication and understanding between/among developers and clients

Examples of High-level Requirements

1. System shall provide full functionality without using mouse
2. System shall provide user access to summary as well as raw data
3. User shall be able to edit and correct the data after it is submitted
4. System shall provide access to all data items based on customer name and account number
5. Customer transactions shall be executed via secure internet connection

NOTE: enumerated for tracking, positive logic, one thing per req., use SHALL

**Do not specify what system
DOES NOT do – that is an
infinite set!**

If not in requirement it is not to be worked on,
period!

Functional requirements are prioritized

Important

- Details will be covered later but the key is to prioritize functions
 - Helps project planning
 - Helps UI design – high priority functions deserve more prominent UI exposure
 - (We will cover priorities later)

Data glossary/description

Important

Goals: Define key entities, data, terms

- Extremely useful (e.g. defines name space, rules etc.) to bring the team together
- **Many problems in design, debugging and maintenance occur because people use different names for same things**
- Defines class/variable/method names and DB design and naming
- Sometimes defines behavior/permissions too (what certain class of users are allowed to do)
- This is part of early design and gets checked by more than just engineering (could involve marketing, sales, legal)

Impetorative to use consistent naming

- Define “name space” in data glossary, then keep the same names in requirements, SW classes, DB tables etc.
- Use only one, defined name, for each entity – be consistent
- If you defined “unregistered user” do not later use “guest user” to mean the same thing
- Will be checked and graded

Data glossary/description – examples – high level

Define name

Define behavior

General user: can access only public photos. Does not need to login/register

Approved user: can access photos explicitly approved by the *author*. Needs to login/register

Author: person who owns photo copyrights and posts them on the site

Admin: can access all data and content and modify the database. Needs to login/register

Public photos: accessible to anybody; approved by author

Logical level – say WHAT it is and do not worry about

Data format and implementaion

One example (Mobility project Stanford)

(1) – combines use case and data definitions

MobilityDB *studies* can be of two types:

Public: in this case all study data can be seen and read-only accessed (info, search and download) by any user, registered or not.

Private: only user title is seen to non-registered users. To view study data and to search and download users must be registered AND invited (then accept) to the study. Users can be invited with read-only or read-write privileges

One example (Mobility project Stanford)

MobilityDB *users* can be (2)

Anonymous or non-registered users – they can view, search and download (but not modify or upload) all public user studies and see only title of private studies

Registered users: they register to MobiltyDB but to get access to a study (read-only or read-write) they have to be invited to particular study by study owner, then accept the invitation

Study owners: registered MobilizeDB user who can issue user invitations for his/her studies and set: a) study type (public or private); and b) user privileges as read-only or read-write

User invited to the study: after accepting invitation, registered user is accepted to the study, in *read-only* or *read-write* mode as per invitation setting

Administrator: has all privileges and can access and modify permissions for all users and all studies

Data glossary for the class team project

- Define names and types of users and what they are allowed to do (registered, unregistered, admin) → drives permissions
 - (do not define users by roles e.g. buyer or seller, but by what they can do...)
- Define key data entities (sales item; real-estate item) and all its components → derives classes/objects and DB schema
- At start stay at logical level, not detailed formatting (e.g. define what each element contains like description, image, process, address etc.).
- **Once agreed, use them consistently in the rest of the project: documentation, class/object names, database table names, instructions on the UI screens etc.**

Data Glossary - example

- Define **user types/names** and roles (unregistered, registered, admin)
- Define key **data entities** (sales items, rental property, media element) and their subcomponents (title, decryption, cateegory, price, address...)
- Define any other non-usual and non-trivial item
- Agree on naming of each data item and then be **consistent** in code (e.g. class, method names), documentation, database tables
- Write this in well organized fashion

Non-Functional Requirements ^{Important}

- **Refer to system properties and constraints such as:**
 - Reliability
 - Response time
 - HW and networking requirements
 - Usability requirements
 - Marketing, legal, licensing
 - Media content (formats, size...)
 - Privacy: what is the data collected, how is the data used
 - Compatibility (e.g. which browsers...)
- **Can also refer to:**
 - Product (product behavior like speed, reliability)
 - Organization (e.g. process, standards used)
 - External factors (e.g. branding, legal disclaimers displayed)

Non-Functional Requirements (2)

- Examples:
 - System shall response visually within 5 seconds
 - File size in no time shall exceed 2 Mbytes
 - Users with high-school diploma, after 1 hour training, shall complete the task in 5 minutes with no more than 2 errors.
 - Tools from XYZ shall be used for requirement management
 - Each requirement shall have identifiable portion of code associated with it referenced by module name and code line number
 - Each WWW page shall have official company logo in upper left corner
 - The data shall be used ONLY for for

Non-functional requirements (3)^{Important}

- Determined often by nontechnical constraints:
 - Business needs
 - Legal, marketing, copyright
- Not prioritized – they are all required
- Can not be changed solely by eng. team - change requires all stakeholders
- Drive architecture design
- **For class team project given in high level pacification – use as is, do not add or change**

Non-functional specs drive architectural design

- Efficiency/speed → networks, CPU, servers, caching, special accelerators
- Load – ability to serve large number of users → multiple servers and replication, caching, large servers
- Security → layered architecture, databases
- Localization → all text ins separate files

Domain Requirements

- Reflect specific features and algorithms from the application domain
- Examples:
 - Speed shall be computed as...
 - Illumination at every pixel shall be computed as...
- Might come from totally different domain (I.e. biology) so it has to be carefully explained
- Can refer to already existing SW that needs to be modified/enhanced

Communication gap between different domains is often critical – must involve domain experts in req. solicitation and definition

Our experience: Bioinformatics

Based on several years of our experience working with biologists and biologists' developed SW:

- Takes long time to actually understand WHAT they want and HOW they want it
- Takes months to understand what the existing SW does (recall: it has likely been written by untrained programmers)
- Bugs/issues that can occur:
 - SW seems to work from CS standpoint, but is not biologically correct (e.g. computes wrong thing)
 - Accuracy not adequate, otherwise works OK
 - SW is “biologically” correct, but has bugs (crashes, leakage of resources)
 - Hard to read and maintain code
 - Combination of the above

Another classification of requirement categories

Low detail

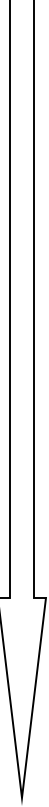
- *High-level requirements or vision or user requirements:* unstructured, natural language with some diagrams
- *System requirements or more detailed requirements:* more structured description of system functions and services. Can serve as a contract between client and developer. Natural language, structured language, simple mockups, data flow diagrams, charts
- *Software specifications:* more detailed SW description to serve as basis for design. Used by developers. Natural language, block diagrams, formulae, program design language (PDL), flow charts, data flow diagrams, mockups, structured language, database schemas etc.

(Exact formats and naming conventions often dictated by company policies)

High level of detail

Requirements: who reads them

Less technical detail (less technical readers)

- 
- *User or High-level Requirements*: clients, users, client engineers, contract managers, systems architects, legal, marketing, executives
 - *System requirements*: users, client engineers, system architects, SW developers
 - *Software Design Specs*: System architects, SW developers, maybe client SW developers, users (for UI)

More technical detail (more technical readers)

High-Level requirements: how should they be written?

- High-level: focus on WHAT not HOW. Write from “outside-in” perspective
 - **Not good:** unction X shall be provided by two buttons in upper left corner, and system should be fast. **Important**
- Leave design decisions (HOW) for later
- Use SHALL for must do, SHOULD for optional
- One concept per requirement
- Cover all necessary issues
- Avoid jargon
- Number them for tracking – use Arabic numbers (1, 2, 3...)
- Group by priorities then by actors/subsystems
- Talk in positive logic
- Details added as we go to lower-level requirements
- Speak in positive logic, do not say what is NOT to be done

WHAT vs. HOW

- **WHAT:** what do you want, what the users want, what is the ultimate goal... → used for high level requirements, specs
- **HOW:** how to do this, how to design this → for lower level, more detailed specs, design, implementation
- Ideally, first focus on WHAT then on HOW
- WHY?
- What happens if you do HOW too early?

Requirement measures

- As much as possible, and where it applies, put objectively measurable “tests” to verify that the requirement is fulfilled
 - “System should be easy to use” is OK as high level requirement from CEO, but without more details does not help design and QA.
- Need to add more details at next level, e.g.:
 - 80% of novice users to complete the process in 5 minutes with no more than 2 errors after 1 hour of training

Still WHAT and not HOW

Requirements measures (2)

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Typical Content of a General Requirement

Important

- Title
- Reference number (for tracking)
- Text (a few lines or a few paragraphs)
- Graphics, diagrams (optional), mandatory for UI
- Initiator, owner
- Priority e.g. 1, 2, 3 (can be assigned early or later)

Many companies have their own templates (government, military, IEEE, ACM)

Follow the required process (or work with management to improve it)

Typical Content of a General Requirement - Text

- Description of service, function or constraints to be achieved
- Use case (or reference to it) **Important**
- Error conditions, special cases
- Measures to verify
- For data glossary: name, description, usage, behavior, storage

Product Requirement Documents

- Requirements are collected in a document. Early requirements are often called Marketing or Product Requirement Document (MRD or PRD). Such documents are official (approved, tracked, dates etc.).
- Use consistent format throughout the project
- Companies often have their own internal formats
- Use language in a consistent way. “Shall” means mandatory, “should” means optional.
- Avoid usage of jargon
- Data dictionary/Glossary at the beginning helps
- Keep change history: date, contributor’s name, summary of change 1-3 lines
- https://en.wikipedia.org/wiki/Product_requirements_document

Requirements document (PRD/MRD) management

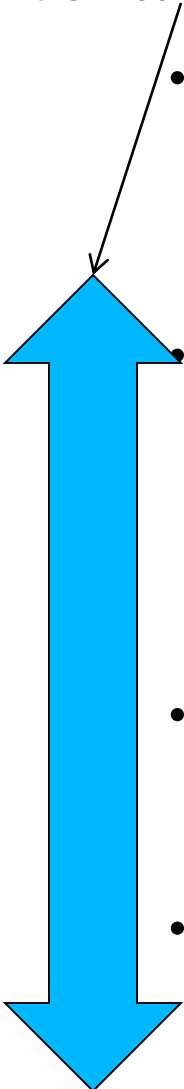
- Can be made arbitrarily complex, to the point that it becomes goal in itself! Be aware!
- Make is simple to enter, modify and track
- Use established tools and standards (MS Word, Excel, HTML)
- Be careful if you want to use special tools. Problems are: cost, training, wide availability, vendor lock in
- Important to keep the versions/dates

Again: sometimes this will be imposed on you by the company, client etc.

Use Cases, Requirements,

Iterations!!!! Specifications

Important

- 
- **Personae and Use Cases:** describe who are the main actors (“*personas*”) and then a story (“*use cases*”) of *WHAT services/functions users will need*, in plain English. Focus on WHT not HOW. Involve marketing, sales, competitive study, focus groups
 - **Requirements:** developed from use cases, usually by marketing, engineering, sales (*focus on WHAT not how*):
 - **Functional** (what functions the SW has, what does it do)
 - **Non-functional** – what properties the SW has (security, privacy, performance, deploy in Cloud? etc.)
 - **Specifications** – more concrete engineering guidance of HOW to develop the SW (often combined with requirements into SW requirement specifications - **SRS**)
 - Followed by **architectural design, implementation, testing in an iterative fashion**

Requirements vs. Specifications

- Distinction is blurred
- One way of thinking: requirements focus on WHAT and specifications add more of HOW information
- Often merged into Software Requirements Specification documents (SRS)
- SW specs: more concrete design and programming guidance (program design language, block diagrams, choice of platforms, compilers, etc,)

Requirements solicitation – gathering requirements

Important

- Requirement Solicitation/elicitation: Process of gathering requirements
- Issues
 - Scope – boundaries of the system to be built
 - Understanding – customers do not understand it either
 - Changes – requirements change in time (better undemanding, changing business/competitive environments, costs, schedules)
 - Cost effective methods of gathering – interviews, meetings, focus groups, marketing research....

http://en.wikipedia.org/wiki/Requirements_elicitation

Issues and problems with requirements for SW Important

- Full requirement set is almost **never known** at early stage
- Requirements **most often change in the process** due to real reasons (people learned more, business changed, users requested it after evaluation) and some other reasons (marketing wants more, competition is there, miscommunication) → recall why it is hard to manage SW projects....
- We will address this in the class (e.g. this is why Waterfall method does not work)
- This is one of the key problems in building and managing large SW systems

Issues and problems with requirements (2)

- Ambiguity: can be interpreted (and will be interpreted) differently by different people or groups.
- Incompleteness: do not specify all what needs to be specified. (Authors assumed that others know too much)
- Conflicting
- Hard to verify
- Hard to use formats and document management
- **It is very hard to create perfect requirements. Don't try to be perfect, be good enough for product to succeed!**
- **Requirements are only means to an end (I.e. successful product)**

Waterfall and other models that required all req. to be known ahead of the time: some history

- David Parnas, Paul Clements, 1986: **Important**
 - A system's users seldom know exactly what they want and cannot articulate all they want
 - Even if we could state all the requirements, there are many details that we can only discover once we are well into implementation
 - Even if we knew all these details, as humans, we can master only so much complexity
 - Even if we could master all this complexity, external forces lead to changes in requirements, some of which may invalidate earlier decisions

Team project requirements solicitation in the class

- During lecture time (this is one reason to attend first section fo the class....
- In team meetings with professor during teamwork session
- Via feedback to Milestone design documents
- You can also discus it with other teams
- Instructors do not have “complete golden set” of requirements ahead of time, they will be developed as we go, each team can have a variation
- Requirement's can change as we go, and be added or dropped!
- **This is how Agile and User Centered SW process works!!!!**

Can ChatGPT help in requirement design?

- Yes, a bit.....check CgatrGPT presentation
- Some resource as an example:
 - <https://www.linkedin.com/pulse/using-chatgpt-4-write-requirements-rick-molony>

BUT NOTE:

- ChatGPT will likely provide generic requirements, but most successful apps are customized to a market or customers → need to check and edit whatever ChatGPT provides (unlikely ChatGPT will know how to customize it for SFSU...not enough data for it to “learn”)
- For team project: recall you MUST provide SFSU unique functionality, ChatGPT will NOT likely help there

Competitive analysis

Important

In design of any system one needs to know

- Goals and objectives
- Competitive landscape – who is doing something similar
- Relationship of your planned product with current market leaders

Developed as *competitive analysis* (jointly with marketing)

All CEOs, managers and investors ask this up front

Competitive analysis— know the background; who did what; and how your envision product compares

- Feature and performance comparison with competitors
- Try competitor products yourself and compare (be honest)
- Survey product review literature and analyze gaps
- Know industry trends and leverage them
- Use GUI metaphors established in relevant industry area
- Define priorities from competitive standpoint (I.e. reviewers/customers always point that competitors have this feature and we do not...)
- Work closely with marketing and sales and customer reps
- Presentation suggestion: A table and a bit of text (summary, key points)

Typical Competitive features table

Important

Feature	Competitor A	Competitor B	Our future product
Text Search	++	+	+
Boolean Search	+	-	+
Browse	+	+	++
Shopping cart	+	+	+

+ feature exists; ++ superior; - does not exist

copyright D. Petkovic

Highlight your product

64

Do not forget...

- In competitive analyses must list your unique functions (this in fact is the very reason you are claiming to be competitive) and relate them to competition
- For team project your SFSU unique functions must be listed and compared

ChatGPT could help in competitive analysis?

- **I say not much! (At least for now)**
- Problem:
 - It tends to invent things that do not exist so check its output
 - Trained till end of 2021 so would not know the latest products
- I would NOT use it for this, not good at factual research where concrete references have to be used (use classic Google)!

Priorities

Important

- It is critical to have priorities attached to each requirement so you can make choices when schedules and costs and resources get constrained
- Agree on priorities with marketing, executives and customers
- It never happens that all features are equally important
 - *Priority 1*: Critical: do not launch product without it.
 - *Priority 2*: Important: it would be really good to have it. Don't delay ship date if not available
 - *Priority 3*: Opportunistic: nice to have. Candidate for next release.
- **When it comes to cutting functions/SCOPE (always happens late in the schedule) use priorities to make a decision**
- Only used for functions to be provided, not for data glossary or non-functional requirements

Priorities vary from domain to domain

- *Medical*: patient and staff safety; accuracy; privacy; ease of use
- *R&D, Academic*: accuracy; performance; ease of use
- *Banking*: transaction accuracy; security; reliability; availability
- *Games*: graphics, story, usability

Check this early and use to guide design and project management/resource optimization

Priorities and specs

- **Functional specs** Important
 - Critical to prioritize
 - Priorities used to guide the design and decisions what to keep/drop/add
 - Helps plan the development (focus on priority 1 first)
 - Grouping by priorities (especially Priority 1) helps better understand what the product features are
- **Nonfunctional specs**
 - Usually fixed and given by funding client/management/legal/marketing
 - **Generally not prioritized – all must be done**
 - Can not be changed solely by engineering team

Requirements for UI and User-Intensive Systems **Important**

- Again: Personae > Uses cases first PLUS:
 - Strongly driven by use cases and types/categories of users - *personas*
 - **Add: Considerations related to usability, inherent characteristics of intended users and UI design principles**
 - It is very hard to communicate and specify such requirements only with text – users need to see it before they can understand or decide → use of graphical mockups and storyboards is essential (with some text please)

Personas - pay attention to UI issues

Important

now

Personas have:

- Attitude
- Skills
- Limitations
- Pain points
- Goals

Personas go over scenarios (use cases) using your application in the appropriate context (home, on the road...)

Create one page persona description (with image) to be handy during the design – some examples below

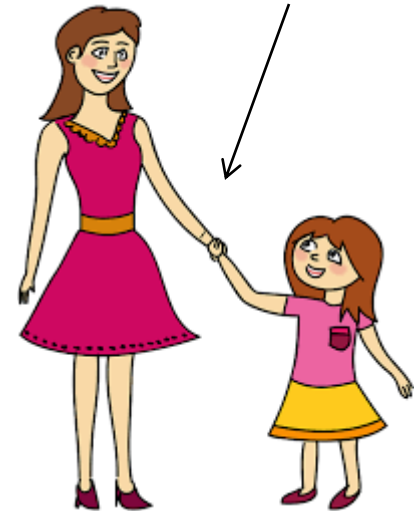
- https://www.google.com/search?q=ui+design+persona&hl=en&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjmu_Wv6KjbAhUfIDQIHTHTD2YQ_AUICigB&biw=1440&bih=782

Example persona: Mary– mother and part time worker

About Mary:

- Very busy
- Loves to take kids to park
- Environmentally conscious: likes to take kids to environmentally safe parks; will report environmental issues
- **Not too patient with WWW apps**
- **Has basic WWW skills**
- **Prefers mobile**
- **Color blind, visually impaired a bit**

Add some image to make it more personal



UI considerations

Goals and scenario:

- Decided to go to a park. Wants to quickly check (from home) if park is safe re: environmental issues

From Use Cases and Personas to High-level Mockups, storyboards

- Simplified graphical representations of envisioned UI
- Focus is on UI basics: functions and behaviors, layout, flow
- Graphics none or very minimal so as to not grab the attention
- High level mockups are best B&W! (Color distracts from UX analysis)
- Text description for all functions and main issues
- Format: wire diagrams, HTML, hand drawings, animation
- Mandatory for UI requirements and specifications

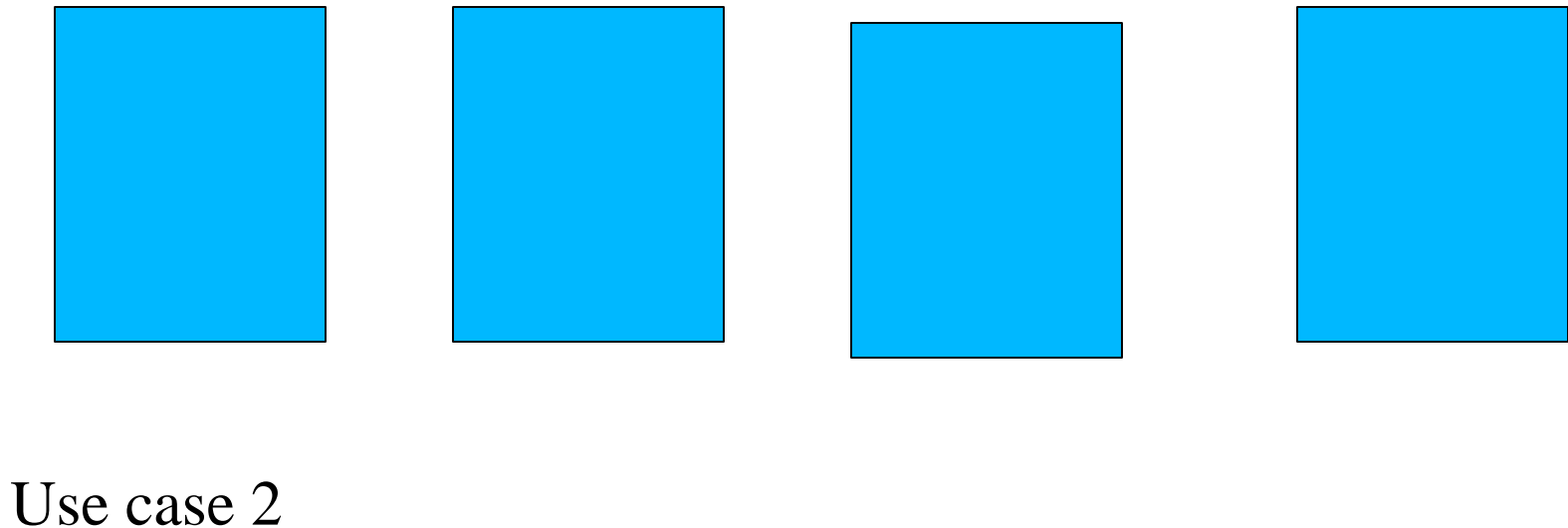
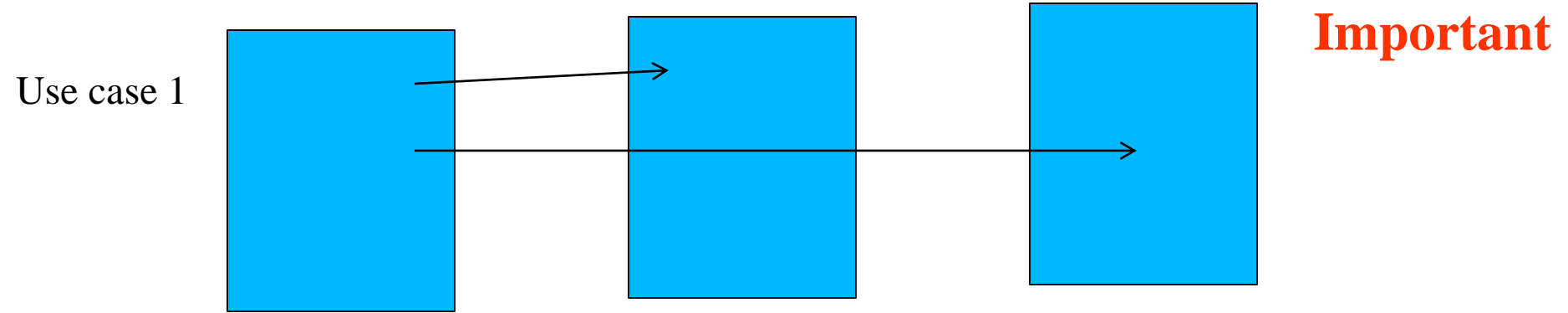
Mockup example



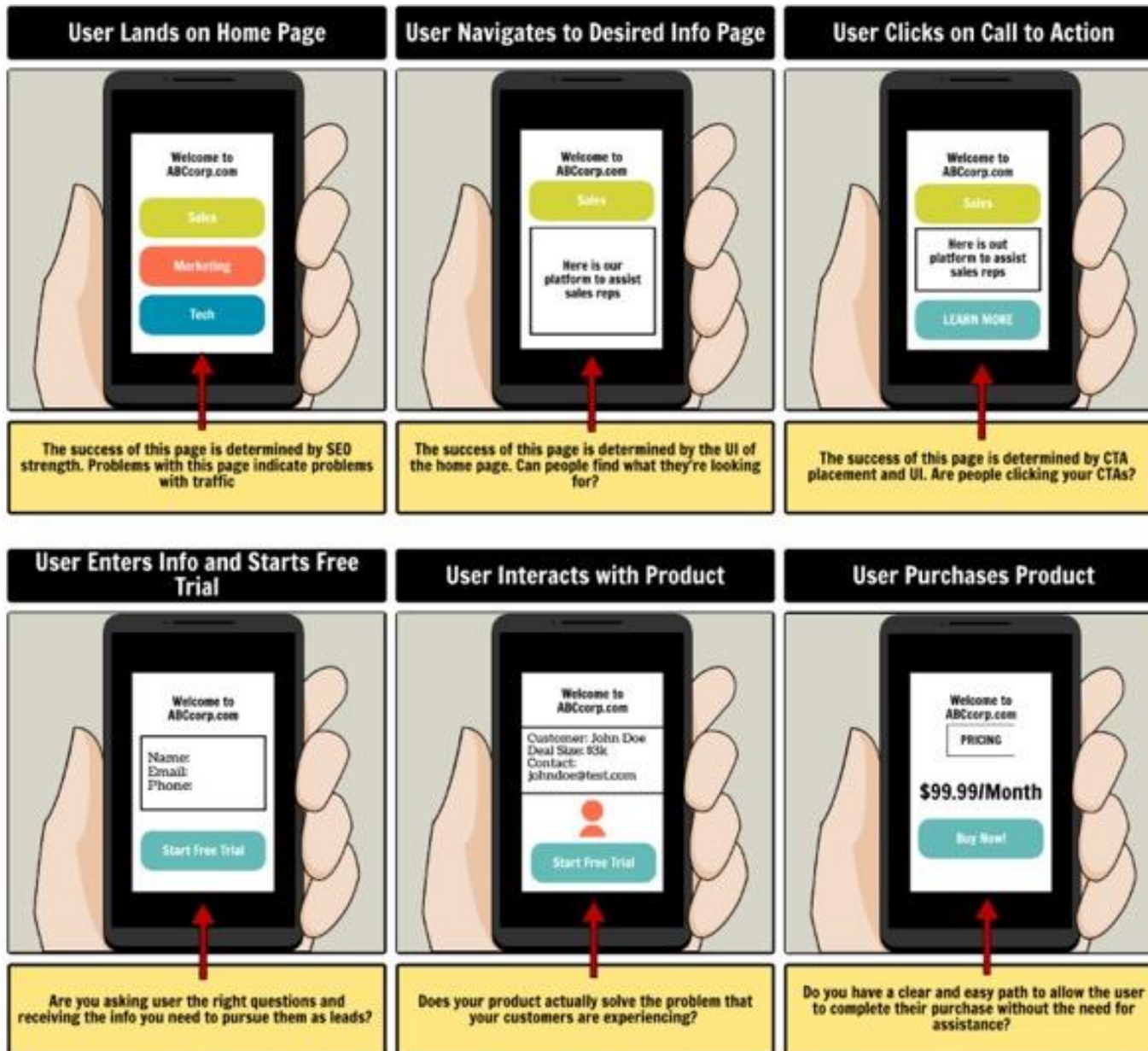
**UI Mockups –
design of VMWare Workstation 3.0 (using “wire diagrams”)**

VMware Toolbar	
VMlist	Status (configuration, setup) or MKS (Mouse, screen, Keyboard area)
Status messages: mouse release, tools install etc.	

Storyboards – sequence of mockups following a use case



Storyboard example



Why are simple wire diagrams *best in early stage?*

Important

- Easy to produce (can be hand drawn too)
- Easy to change (no coding invested)
- Allow easy scribbling, markup
- No graphics or colors help focus on key issues: function and layout
- Allow considerable “scenario and usage” testing

(Many great designs done on a napkin...)

UI Mockups and storyboards – resources (templates, examples)

- <http://webdesignledger.com/inspiration/18-great-examples-of-sketched-ui-wireframes-and-mockups>
- <http://www.onextrapixel.com/2010/09/29/40-brilliant-examples-of-sketched-ui-wireframes-and-mock-ups/>

And there are tools and tools for this

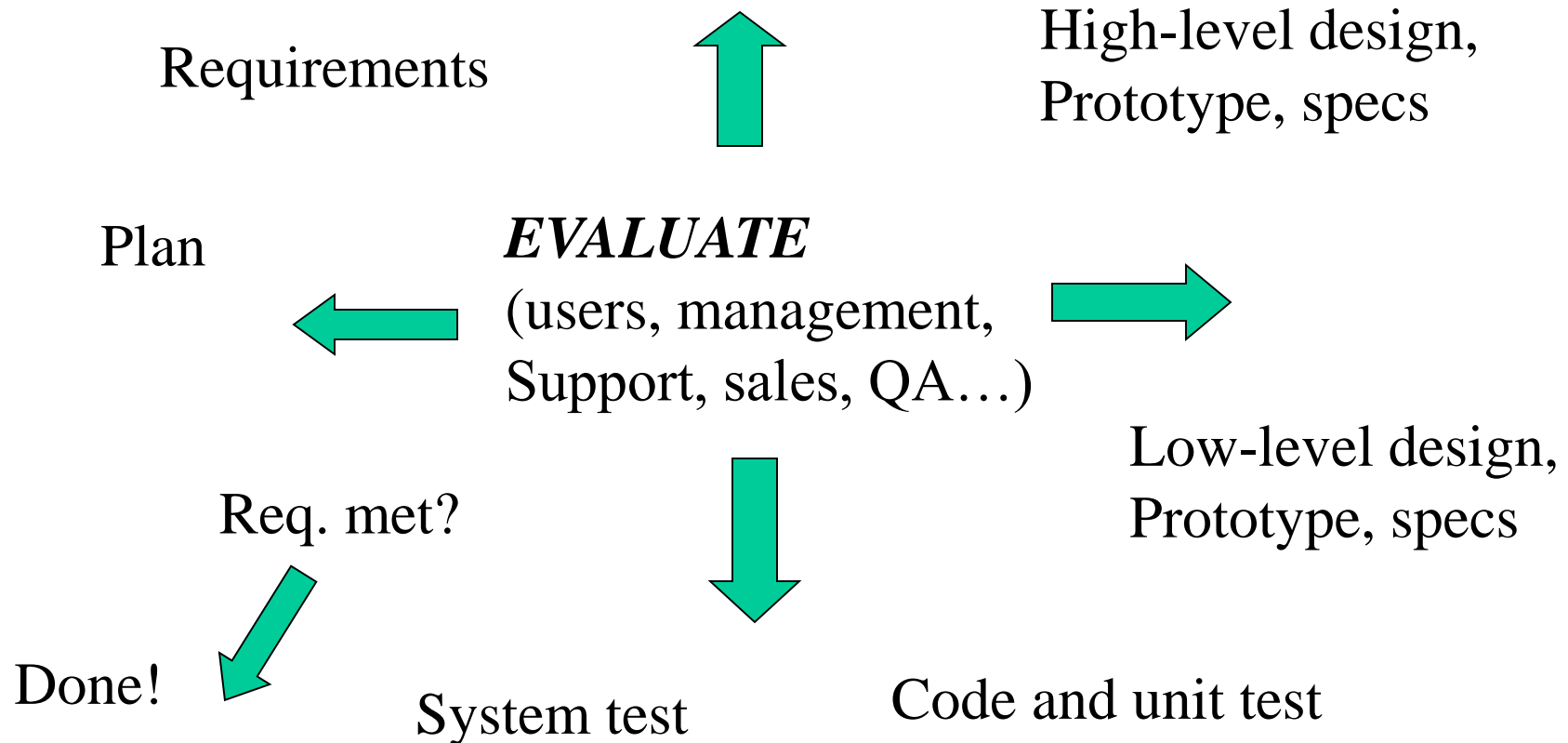
Review – top 22

- <https://blog.prototypr.io/top-20-prototyping-tools-for-ui-and-ux-designers-2017-46d59be0b3a9>
- Figma <https://www.figma.com/> (bought by Adobe)
- Sketch <https://www.sketch.com/>

User-Centered Design and Development – MUST DO process for User intensive systems design

Conceptual design, prototype,
Specs, style guide

Important



Iterative, evaluated all the time by multidisciplinary team

User-Centered Design and Development

Important

- Think of *total user experience* including:
 - Product packaging,
 - Ordering, registration
 - Installation
 - Help and docs
 - GUI itself, mainly *functions, simplicity, aesthetics, responsiveness, reliability* etc.
 - Support
 - Upgrade and legacy issues
 - Uninstall
- ➔ Need to address all this in requirements

High Level Requirements for User Interface – what to cover **Important**

- UI style
- Platforms and standards (browser, client OS, screen size etc.)
- Consistency with other playroom SW
- Key screen contents
- Use of media, graphics, logos, colors, branding
- Screen behavior
- Appearance
- User interaction techniques
- Response time
- Navigation features
- Saving data etc.
- Legacy data issues
- Support, training, help
- Error handling
- Accessibility features
- Localization, Internationalization
-

(Initially do high level.

Validate early.

Iterate.

Focus on WHAT not HOW.

More specific later)

Focus groups - Essential tool in developing requirements in UCD

Important

- http://en.wikipedia.org/wiki/Focus_group
 - “In **usability engineering**, a focus group is a survey method to collect the views of users on software or a website. This marketing method can be applied to computer products to better understand the motivations of users and their perception of the product. Unlike other methods of ergonomics, focus group implies several participants: users or future users of the application. The focus group can only collect subjective data, not objective data on the use of the application as the usability test for example.”
- **Focus groups can review use cases, ideas, mockups in addition to working code**

Can ChatGPT help in early UX req and design

- **Yes, but again:**
 - Check its output
 - Customize it as per your needs (unlikely ChatGPT will know how to do it for SFSU...not enough data for it to “learn”)
- Check “ChatGPT for UX” posted document for some resources - it can help in many UX dev stages

Other things to cover in requirements

Requirements for Media Rich apps

(almost all today)

Important

- Raw data: Files used to render content (MPEG, PDF, JPEG etc.)
- Metadata: data describing content, used for searching
- Supporting data: data used to display items in the search result list. In the form of text, but also may include key frames, samples, previews
- Price
- Copyright rules

Used to design both search as well as database organization

Requires close interaction with data owners, librarians etc.

Content and Information Organization or Information Architecture

Goal: organize data so it is easy to be used by users
(for annotation, search, browse, navigation)

- Exact
 - Alphabetical
 - Chronological
 - Geographical
- Ambiguous:
 - Topical (categories, but what categories)
 - Task oriented (collections of functions and services)
 - Audience specific
 - Metaphor driven (relate to something users know)
 - Hybrid

How to develop requirements regarding content organization

This is information/data design issue not a coding issue

- Interview current owners and users
- Base it on search needs of users as well as nature of the content (common ways of describing content)
- Look at similar other products or applications
- Review legacy (current usage)
- Follow legal rules on content usage
- Application dependent:
 - Movies
 - Stock photo
 - Books
 - Shopping

Media Content Protection

- Practically impossible if full res digital files are available
- Ability to copy digital data infinitely without loss of quality is the key worry of media producers
- Impossible to keep ease of use/access and at the same time track/enforce number of digital copies (requires cumbersome polices and special SW)
- Issue has not been solved technically → legal system is used to punish most important violators (e.g. music)

Data and User privacy Important

- Privacy is legally protected category – MUST be observed
- Important to agree/design early and communicate clearly (e.g. privacy policies). E.G. during user registration:
 - WHAT is collected and WHY
 - HOW it will be used
 - Who will have ACCESS
 - How is it PROTECTED
- Ask user to opt in (check the box) on agreeing to privacy rules during registration/sign in and record this in your DB
- Place link to privacy policy at the bottom of the WWW page

Some medical/health info standards are set by Government and are legally binding, e.g. HIPAA

- https://www.hhs.gov/hipaa/for-professionals/privacy/index.html#:~:text=The%20HIPAA%20Privacy%20Rule%20establishes%20national%20standards%20to,providers%20that%20conduct%20certain%20health%20care%20transactions%20electronically.

Story about requirements – some wisdom and experience

- Requirements are very important, BUT be aware of perils:
 - Insisting on complete requirements is most often an obstacle to SW development. They simply may not be known at that point
 - Same holds for excessive, inflexible and cumbersome documentation and requirement process mechanics
 - “Requirement paralysis” → endless discussions, while schedules are being missed
 - Break the impasse by iterative method and early prototyping, then refine
- There are applications where requirements are critical (mission critical, safety, medical, military, legal)
- Use judgment (as always) as to how much to invest in requirements process

Review Questions

- What are SW requirements, and what is their:
 - Purpose
 - Usage (who reads them)
- What are types/categories of high level requirements – what is the common way of categorizing them. Examples
- What is the level of details in high level requirements? Give examples
- Difference between “shall” and “should” in requirement language
- Explain and give examples for functional, non-functional and domain requirements
- What are some requirement measures for system speed, size, ease of use, reliability, robustness, portability (give examples)
- What are some properties of well-written requirements

Review Questions

- What are some problems with requirements for SW
- Give example of a format of requirements (either from class or from your experience)
- What is the difference between general SW requirements and those related to UI
- What are use cases and how are they used in SW design
- What are typical requirements for WWW systems and applications
- Requirements for user privacy
- Do we prioritize requirements? Which we do, which we do not

Review Questions

- You need to very quickly develop a demo in order to get funding for the project. How will you go about requirements solicitation, verification and management b(how formal, how deep, how formally to manage the requirement documents etc.)
- You are project manager of the team that delivers SW for medical industry. How will you go about requirements solicitation, verification and management b(how formal, how deep, how formally to manage the requirement documents etc.)

Review questions

- How would you organize content of a stock photo company where you buy and sell images of others:
 - Data organization
 - Content protection
 - Content delivery
- How to balance ease of access and search and attractiveness with content protection
- How to allow independent photographers to easily contribute and deliver to you for resale
- If your company produces APIs, think how would you present them, “sell them”, document them and manage changes
- Give examples of good data glossary
- How is the data glossary used in the SW development?