TASK

System Requirements and Design

Visit our website

Introduction

Welcome to the System Requirements and Design Task!

This task explores the best practice guidelines for defining your product - including gathering and documenting system requirements. It also introduces UI/UX design guidelines and tools such as wireframing, prototyping and use cases.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

Requirements

The process of designing software includes identifying the requirements of a software system. Requirements describe what a system should do. They describe all the activities the system must perform and the constraints that the system must meet. These requirements reflect the needs of customers for a system that serves a certain purpose. For example, a purpose could be controlling a device, finding information or placing an order. Requirements engineering is the process of finding, analysing, documenting and checking these requirements.

Requirements are generally divided into two categories:

1. **User requirements:** These requirements are statements (in a natural language) and diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.

2. **System requirements:** These requirements are more detailed descriptions of the software system's functions, services, and operational constraints.

Software system requirements are classified as functional requirements or non-functional requirements:

1. **Functional requirements:** The activities that the system must perform.

2. **Nonfunctional requirements:** The characteristics of the system other than those activities it must perform or support.

Functional Requirements

Functional requirements describe what a system should do. These requirements depend on the type of software being developed, the expected users of the software, and the approach taken by the organisation when writing requirements. For example, if you are developing a payroll system, the required business uses might include functions such as "generate electronic fund transfers," "calculate commission amounts," "calculate payroll taxes," and "maintain employee dependent information". The new system must handle all these functions.

Functional requirements are based on the procedures and rules that the organisation uses to run its business. They are sometimes well documented and

easy to identify and describe, however, this is not always the case. Some business rules can be more obtuse or difficult to find.

Non-Functional Requirements

Non-functional requirements are requirements that are not directly concerned with the specific activities a system must perform or support. They are often more critical than individual functional requirements, however. System users can usually find ways to work around a system function that doesn't really meet their needs.

It is not always easy to distinguish between functional and nonfunctional requirements. You can, however, use a framework for identifying and classifying requirements. The most widely used such framework is called FURPS+. FURPS is an acronym that stands for functionality, usability, reliability, performance, and security. Functionality refers to functional requirements while the remaining categories describe non-functional requirements:

- **Usability:** These requirements describe operational characteristics related to users, such as the user interface, related work procedures, online help, and documentation.

- **Reliability:** These requirements describe the dependability of a system. In other words, reliability is to do with how a system detects and recovers from such behaviors as service outages and incorrect processing.

- **Performance:** These requirements describe operational characteristics related to measures of workload, such as throughput and response time.

- **Security:** These requirements describe how access to the application will be controlled and how data will be protected during storage and transmission.

FURPS+ is an extension of FURPS that adds additional categories all of which are summarised by the plus sign. Below is a short description of each additional category:

- **Design constraints:** These describe restrictions to which the hardware and software must adhere.

- **Implementation requirements:** These describe constraints such as required programming languages and tools, documentation methods and

level of detail, and a specific communication protocol for distributed components.

- **Interface requirements:** These describe interactions among systems.

- **Physical requirements:** These describe the characteristics of the hardware such as size, weight, power consumption, and operating conditions.

- **Supportability requirements:** These describe how a system is installed, configured, monitored, and updated.

The Software Requirements Document

The software requirements document is an official statement of what the system developers should implement. It should include both the user requirements and a detailed specification of the system requirements. Sometimes, the user and system requirements are integrated into a single description and in other cases, the user requirements are defined in an introduction to the system requirements specification. The detailed system requirements may be presented in a separate document if there are a large number of requirements.

The requirements document has a wide set of users. Below is a list of the possible users of this document and how they use it:

System customers: Customers identify the requirements and check them to decide if they meet their needs. They also identify changes that need to be made to the requirements.

Managers: Managers use the requirements document to create a proposal and bid for the system. They also use it to plan the development process.

System engineers: System engineers use requirements to get a better understanding of the system that is to be developed.

System test engineers: Requirements are used by system test engineers to develop validation tests for the system.

System maintenance engineers: Requirements are used by system maintenance engineers to get a better understanding of the system and the relationships

between its components.

Because there is such a wide range of possible users, the requirements document has to communicate the requirements to customers, define the requirements in precise detail for developers and testers, and include information about possible system evolution.

The level of detail that you should include in a requirements document depends on the type of system that is being developed and the development process used. For example, critical systems need to have detailed requirements because safety and security have to be analysed in detail. The table below shows the structure of a requirements document. This is based on an IEEE standard for requirements documents (IEEE, 1998). This is a is a generic standard that can be adapted to specific uses.

Chapter	Description
Preface	Defines who you expect to read the document as well as describes its version history. The description of the version history should include the reason for creating the new version and a summary of the changes made in each version.
Introduction	Describes why the system is needed. Also describes the system's functions and explains how it will work with other systems. Describes how the system fits into the business overall or the strategic objectives of the organisation that commissioned the software.
Glossary	Defines the technical terms used in the document.
User requirements definition	The services provided for the user are defined here. The non-functional requirements are also described. Product and process standards that must be followed must be specified.
System architecture	Presents a high-level overview of the system's anticipated architecture by showing the distribution of functions across system modules. Highlights any architectural components that are reused.
System requirements	Describes the functional and nonfunctional requirements in more detail. Interfaces to other systems can also be

specification	defined.
System models	Might include graphical system models that show the relationship between system components, the system, and its environment. Some examples of models are; object models, data-flow models and semantic data models.
System evolution	Describes the fundamental assumptions which the system is based on. It also describes any anticipated changes due to hardware evolution, changing user needs, etc.
Appendices	Provide detailed, specific information that is related to the application that is being developed. For example, hardware and database descriptions.
Index	Several indexes might be included. This might include a normal alphabetic index, an index of diagrams, an index of functions, etc.

The information that is included in a requirements document depends on the type of software being developed and the approach to development that is to be used.

Requirements Specification

Requirements specification is the process of writing down the user and system requirements in a requirements document. Ideally, the user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent. However, this is rarely the case.

The user requirements for a system should be described in a way that is understandable to system users who don't have detailed technical knowledge. They should ideally specify only the external behavior of the system. The requirements document should not include details of the system architecture or design. If you are writing user requirements, you should not use software jargon, structured, or formal notations. User requirements should be written in natural language, with simple tables, forms, and diagrams.

System requirements are expanded versions of the user requirements. They are used by software engineers as the starting point for the system design. They add detail and explain how the user requirements should be provided by the system. They should be a complete and detailed specification of the whole system. Like

user requirements, system requirements can also be written in natural language but other notations based on forms, graphical system models, or mathematical system models can also be used. The table below describes the possible notations that could be used for writing system requirements:

Notation	Description
Natural language	Each requirement is written as a numbered natural language sentence. One sentence should express one requirement.
Structured natural language	Each requirement is written on a standard template or form in natural language. Each field in the template should provide information about an aspect of the requirement.
Design description language	A language similar to a programming language but with more abstract features to specify the requirements by defining an operational model of the system is used.
Graphical notation	Graphical models with text annotations are used to define the functional requirements of a system. E.g. UML use case and sequence diagrams.
Mathematical specification	These notations are based on mathematical concepts. Most customers don't understand this formal type of specification, however, it is unambiguous and can reduce the ambiguity in a requirements document.

The Requirements Engineering Process

The requirements engineering process includes four high-level activities:

1. **Feasibility study:** Assesses if the system is useful to the business.
2. **Elicitation and analysis:** Discovers requirements.
3. **Specification:** Converts requirements into some standard form.
4. **Validation:** Checks that the requirements actually define the system that the customer wants.

The diagram below shows how these activities are organised as an iterative process around a spiral, with the output being a system requirements document.

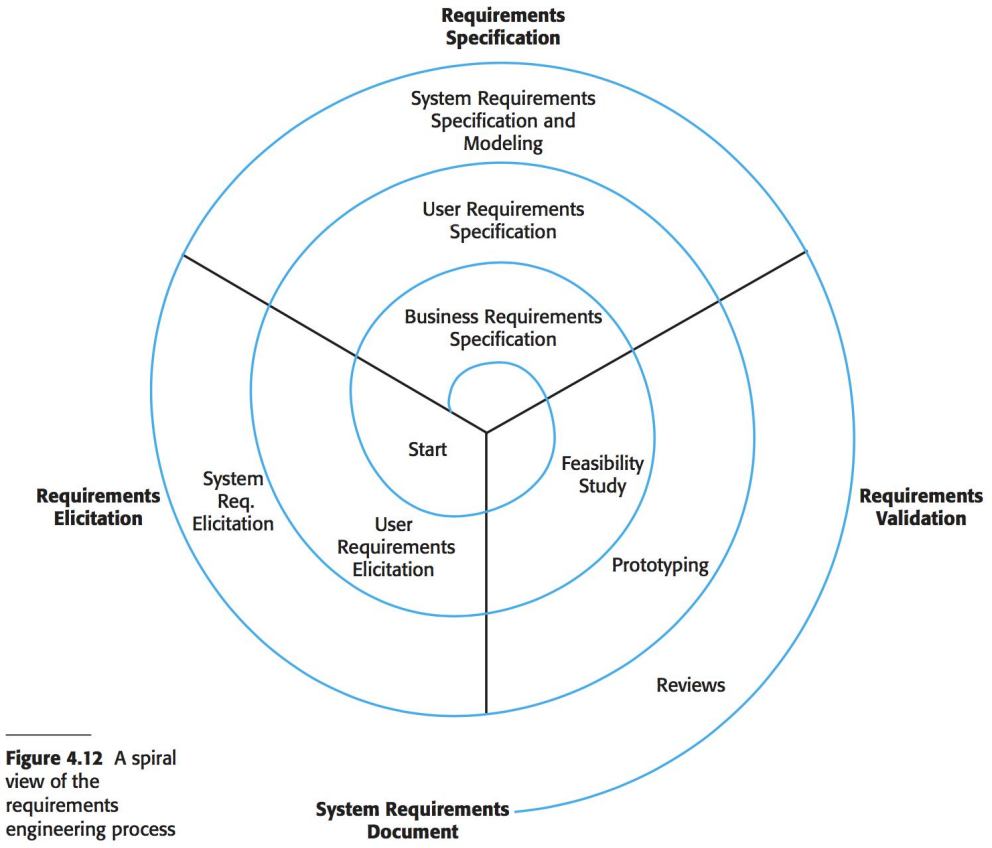


Figure 4.12 A spiral view of the requirements engineering process

The Requirements Engineering Process (Sommerville, 2010)

Requirements Elicitation and Analysis

After the feasibility study, which determines if the system is useful to the business and should actually be developed, the next stage of the requirements engineering process is requirements elicitation and analysis. In this activity, software engineers work with customers and system end-users to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints, etc.

Requirements elicitation and analysis often involves a variety of different kinds of people in an organisation. All the people who have an interest in the successful implementation of the system are known as stakeholders. Stakeholders include end-users who will interact with the system and anyone else in an organisation

who will be affected by it.

The process activities for requirements elicitation and analysis are:

1. **Requirements discovery:** This is the process of interacting with stakeholders of the system to discover their requirements.

2. **Requirements classification and organisation:** This activity involves taking an unstructured collection of requirements and organising them into coherent clusters.

3. **Requirements prioritisation and negotiation:** When multiple stakeholders are involved, requirements will often conflict with one another. This activity is concerned with determining the importance of requirements and finding and resolving requirement conflicts through negotiation. Stakeholders usually have to meet to resolve differences and compromise on requirements.

4. **Requirements specification:** The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced.

Requirements elicitation and analysis is an iterative process with continual feedback from each activity to other activities. Your understanding of the requirements improves with each round of the cycle. The cycle ends when the requirements document is complete.

Requirements Discovery

Requirements discovery is the process of collecting information about the required system and gathering the user and system requirements from this information. Documentation, system stakeholders, and specifications of similar systems are all sources of information that is used during the requirements discovery phase.

Interviews

You can interact with stakeholders through interviews. These formal or informal interviews with system stakeholders are part of most requirements engineering

processes. During these interviews, you ask stakeholders questions about the system that they currently use and the system to be developed. Requirements are then derived from the answers to these questions.

There are two types of interviews:

1. **Closed interview:** This is where the stakeholder answers a predefined set of questions.

2. **Open interviews:** In this type of interview, there is no predefined agenda. You explore a range of issues with system stakeholders to develop a better understanding of their needs.

Interviews with stakeholders are normally a mixture of both types. Completely open-ended discussions rarely work well. You usually have to ask some questions to get started and to keep the interview focused on the system to be developed.

Interviews are good for getting an overall understanding of what stakeholders do, how they might interact with the new system and the difficulties that they face with current systems. However, interviews are not so helpful in understanding the requirements of the application domain and they are also not as effective in gaining knowledge about organisational requirements and constraints.

There are two characteristics that all effective interviewers have:

1. They are open-minded, avoid preconceived ideas about the requirements, and are willing to listen to stakeholders. If the stakeholder comes up with surprising requirements, then the interviewer is willing to change their mind about the system.

2. They prompt the person being interviewed to get discussions going. They can do this by using a springboard question, a requirements proposal, or by working together on a prototype system.

Information from interviews should be used to supplement other information about the system from documentation describing business processes or existing systems. It should be used in conjunction with other requirements elicitation techniques as interviewing on its own can miss essential information.

Scenarios

People find it easier to relate to real-life examples rather than abstract descriptions. They can understand and criticise a scenario of how they might interact with a software system. Engineers can then use the information gained from this discussion to generate the actual system requirements.

Each scenario usually covers one or a small number of possible interactions with a system. Different forms of scenarios should be developed that provide different types of information at different levels of detail about the system. A scenario starts with an outline of the interaction. During the elicitation process, details are then added to create a complete description of that interaction. A scenario may generally include:

- A description of what the system and users expect when the scenario starts.
- A description of the normal flow of events in the scenario.
- A description of what can go wrong and how this is handled.
- Information about other activities that might be going on at the same time.
- A description of the system state when the scenario finishes.

Scenario-based elicitation involves working with stakeholders to identify scenarios and to capture details to be included in these scenarios. Scenarios may be written as text, or supplemented by diagrams, screenshots etc.

Use Cases

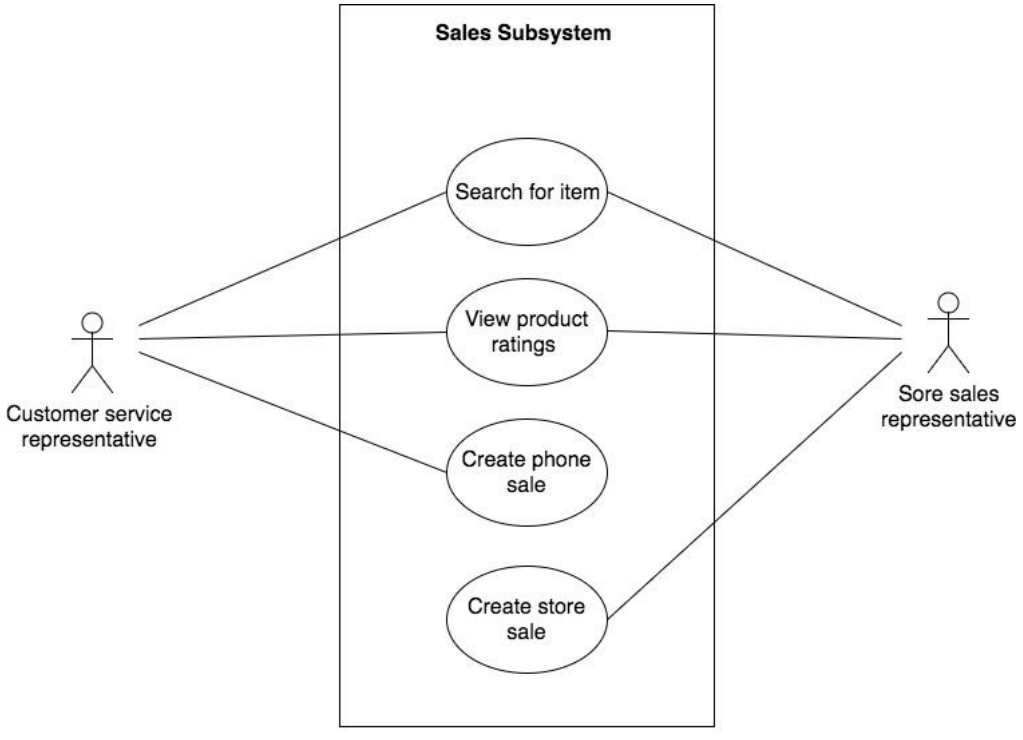
A use case is an activity that the system performs, usually in response to a request by a user. It identifies the actors (people or groups, external to the system, that interacts with the system by supplying or receiving data) involved in an interaction and names the type of interaction. The set of use cases represents all of the possible interactions described in the system requirements.

Each use case should be documented with a brief description which can then be used to create other UML models. The table below shows some examples of brief use case descriptions:

Use case	Brief use case description
Create customer account	The actor enters new customer account data, and the system assigns the customer an account

	number, creates a customer record, and creates an account record.
Look up customer	The actor enters the customer's account number, and the system retrieves and displays customer and account data.
Process account adjustment	The actor enters an order number, and the system retrieves customer and order data. The actor enters adjustment amount, and the system creates a transaction record for the adjustment.

Use cases are documented using a use case diagram. Actors in the process, who may be human or other systems, are represented as stick figures and the use case itself is represented by an oval with the name of the use case inside. Lines link the actors with the interaction or use case. The following diagram is an example of a use case diagram:



There is no clear distinction between scenarios and use cases. Some people consider that each use case is a single scenario while others encapsulate a set of scenarios in a single use case. Scenarios and use cases are effective techniques for eliciting requirements from stakeholders who interact directly with the system. Each type of interaction can be represented as a use case. They are not as effective

for determining constraints or high-level business and nonfunctional requirements or for discovering domain requirements, however, as they focus on interactions with the system.

Requirements Validation

Requirements validation is the process of checking that requirements actually define the system that the customer wants. It overlaps with analysis as it is concerned with finding problems with the requirements. Requirements validation is important because errors in a requirements document can lead to extensive costs when these problems are discovered during development or after the system is in service. The cost of fixing a requirement by making changes to a system is usually much higher than repairing design or coding errors.

During the requirements validation process, different types of checks should be carried out on the requirements in the requirements document. These checks include:

- **Validity checks:** A user may think that a system is needed to perform certain functions. However, further thought and analysis may identify additional or different functions that are required.

- **Consistency checks:** Requirements in the document should not conflict with one another. This means, there should not be contradictory constraints or different descriptions of the same system function.

- **Completeness checks:** The requirements document should include requirements that define all functions and constraints intended by the system user.

- **Realism checks:** The requirements should be checked to ensure that they can actually be implemented using existing technology.

- **Verifiability:** To reduce the potential for a dispute with the customer, system requirements should always be written so that they are verifiable. This means that you should be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement.

There are a number of requirements validation techniques that can be used individually or in conjunction with one another:

- **Requirements reviews:** The requirements are analysed systematically by a team of reviewers who check for errors and inconsistencies.

- **Prototyping:** An executable model of the system in question is demonstrated to end-users and customers. End-users and customers can experiment with this model to see if it meets their needs.

- **Test-case generation:** Requirements should be testable. If the tests for the requirements are devised as part of the validation process, it can reveal requirement problems. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered.

It is very difficult to show that a set of requirements meets a user's needs. Users need to picture the system in operation and imagine how that system would fit into their work. As a result, you rarely find all of the requirement problems during the requirements validation process. It is inevitable that there will be further requirement changes to correct omissions and misunderstandings after the requirements document has been agreed upon.

Prototyping

A prototype is an initial version of a software system that is used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions. It is essential to develop a prototype rapidly and iteratively so that costs are controlled and system stakeholders can use and experiment with the prototype early in the software process.

A software prototype can be used in a software development process to help anticipate changes that may be required:

- A prototype can help with the elicitation and validation of system requirements in the requirements engineering process.

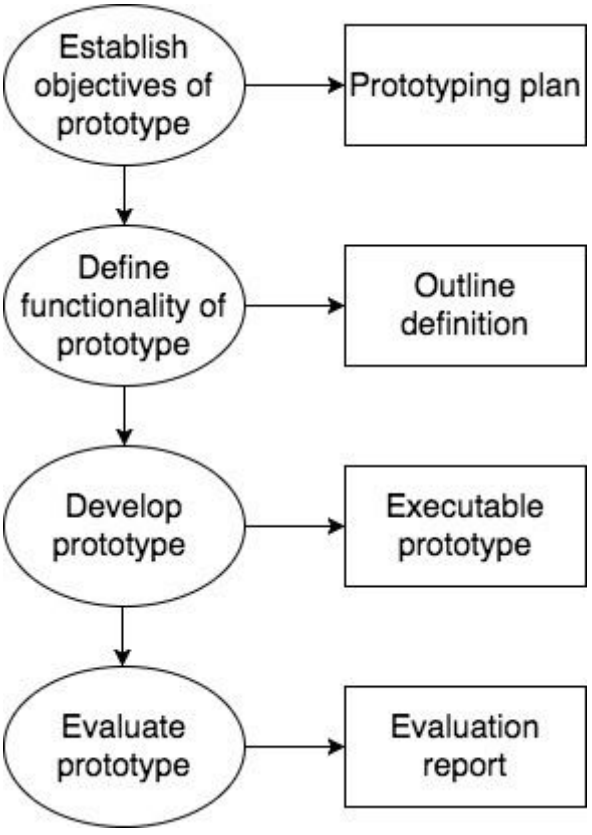
- A prototype can be used to explore particular software solutions and to support user interface design in the system design process.

System prototypes allow users to see how well the system supports their work. They are used to find new ideas for requirements and areas of strength and

weakness in the software. Prototypes may also reveal errors and omissions in the requirements that have been proposed as it is developed.

A prototype can be used while the system is being designed to carry out design experiments to check the feasibility of a design. Prototyping is also an essential part of the user interface design process. Textual descriptions and diagrams are not good enough for expressing user interface requirements because of the dynamic nature of user interfaces. Rapid prototyping with end-user involvement is therefore, the only sensible way to develop graphical user interfaces for software systems.

Below is the process model for prototype development:



It is extremely important that the objectives of prototyping should be made explicit from the start of the process. These objectives may be used to develop a system to prototype the user interface, to develop a system to validate functional system requirements, or to develop a system to demonstrate the feasibility of the application to managers, however, the same prototype cannot meet all objectives. If you do not state the objective, it is possible for the users to misunderstand the function of the prototype and they may therefore, not get the benefits that they

expected from the prototype development.

You then need to decide what to put into and what to leave out of the prototype system. You may have to leave some functionality out of the prototype in order to reduce prototyping costs and accelerate the delivery schedule.

Prototype evaluation is the final stage in the process. Users need time to become comfortable with a new system and to settle into a normal pattern of usage. Provision must be made for user training during this stage and the prototype objectives should be used to derive a plan for evaluation. Requirements errors and omissions can then be discovered once the users are using the system normally.

One of the problems with prototyping is that the prototype may not be used in the same way as the final system. The tester of the prototype may not be a typical system user and the training time during prototype evaluation may not be sufficient. The testers may adjust their way of working if the prototype is slow and avoid those system features that have slow response times. They may then use the final system in a different way when provided with a better response.

Prototypes do not have to be executable to be useful. For example, paper-based mock-ups of the system user interface can be effective in helping users refine an interface design and work through usage scenarios. These are very cheap to develop and can be constructed in a few days.

User Interface Design

Software design is the stage in the software engineering process at which an executable software system is developed. A key step in systems design is to classify the inputs and outputs for each event as either a system interface or a user interface. User interfaces are inputs and outputs that directly involve a system user.

Many people think the user interface is developed and added to the system near the end of the development process, however, this is not the case. The user interface is extremely important since it is the component that the end user comes in contact with while using the system. From a user perspective, the user interface is the entire system and the programs, databases, and hardware behind the interface are irrelevant.

User Interface design is the design of user interfaces for software or machines with a focus on ease of use and pleasurability for the user. User interface design generally refers to the design of graphical user interfaces, however, it can also to other interfaces, such as natural and voice user interfaces. It focuses on anticipating the actions a user might need to perform and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions.

Interface elements include:

- Input Controls: checkboxes, radio buttons, dropdown lists, list boxes, buttons, toggles, text fields, date field
- Navigational Components: breadcrumb, slider, search field, pagination, slider, tags, icons
- Informational Components: tooltips, icons, progress bar, notifications, message boxes, modal windows
- Containers: accordion

There are times when multiple elements might be appropriate for displaying content. It is important to consider the trade-offs when this happens. Sometimes elements that can help save you space put more burden on the user mentally by forcing them to guess what is within the drop-down or what the element might be for example.

Best Practices for Designing an Interface

- **Keep the interface simple:** Interfaces that are simple are the best interfaces. They avoid unnecessary elements and are clear in the language they use on labels and in messaging.
- **Create consistency and use common UI elements:** Users feel more comfortable and are able to do things more quickly when you use common elements in your UI. You should also aim to create patterns in language, layout and design throughout the site to help facilitate efficiency. Once a user learns how to do something, they should be able to transfer that skill to

other parts of the site.

- **Be purposeful in page layout:** You should consider the spatial relationships between items on the page and structure the page based on importance. You can help draw attention to the most important pieces of information and can aid scanning and readability by careful placement of items.

- **Strategically use color and texture:** You can direct attention toward, or redirect attention away from items, by using color, light, contrast, and texture to your advantage.

- **Use typography to create hierarchy and clarity:** How you use typeface should be carefully considered. Use different sizes, fonts, and arrangement of the text to help increase scannability, legibility and readability.

- **Make sure that the system communicates what's happening:** You should always inform your users of location, actions, changes in state, or errors. The use of various UI elements to communicate status can reduce frustration for your user. For example, a simple progress bar can give the user an indication of how long an action will take. It can also show that the system is processing so that the user doesn't repeatedly press a button because they think your system isn't responding.

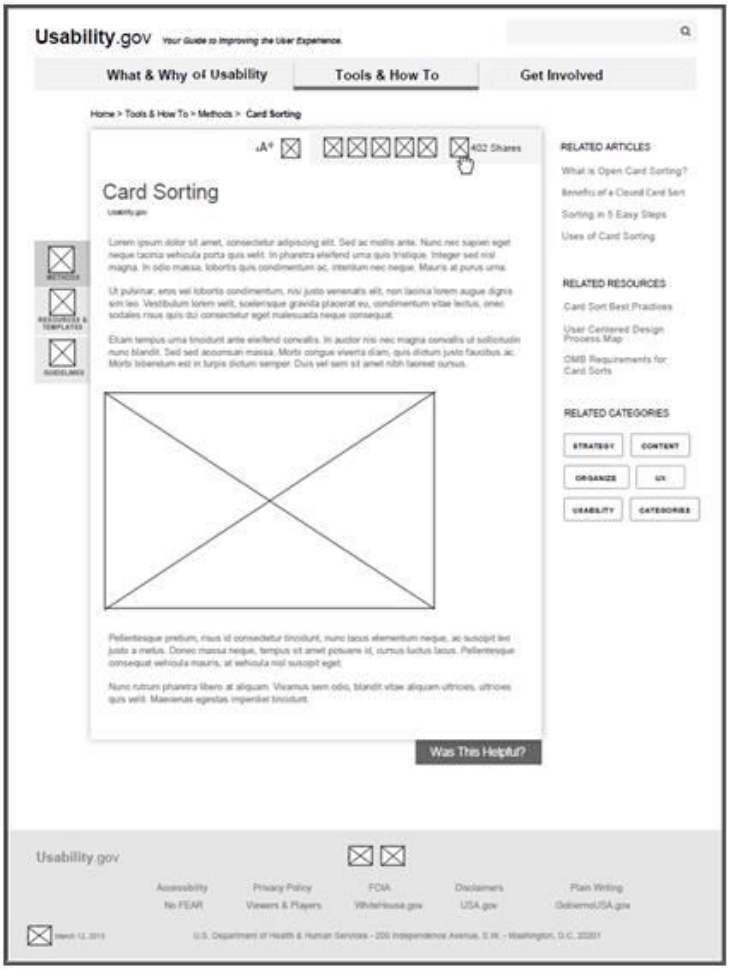
- **Think about the defaults:** You can create defaults that reduce the burden on the user by carefully thinking about and anticipating the goals of the people using your system. This is particularly important when it comes to form design where you might have an opportunity to have some fields already filled out.

Wireframing

Wireframes are simply a "blueprint" of a web application. It is a two-dimensional illustration of a webpage's interface that focuses specifically on space allocation and prioritisation of content, functionalities available, and intended behaviors. Wireframes typically do not include any styling, color, or graphics for these reasons.

Wireframes are guides to where the major navigation and content elements of your site are going to appear on the page. The aim is to keep it as simple as possible as the goal of the illustrations is not to depict visual design. You should not use colours. If you need to use colours to distinguish items, use various grey tones

instead. You should also not use images since they tend to distract from the task at hand. You can use a rectangular box sized to dimension with an “x” through it to indicate the placement and size of an image. Finally, you should only use one generic font. Typography should not be a part of the wireframing discussion, however, you can still resize the font to indicate various headers and changes in the hierarchy of the text on the page. Below is an example of a wireframe:



It is important to remember that wireframes don’t do well with showing interactive features of the interface since they are two-dimensional.

The following is a list of elements that are often included as standard elements on wireframes. However, wireframes tend to differ from site to site.

- Logo
- Search field
- Breadcrumb
- Headers, including page title
- Navigation systems, including global navigation and local navigation

- Body content
- Share buttons
- Contact information
- Footer

Compulsory Task

Answer the following questions:

- Consider the following statement of requirements for part of a ticket-issuing system:

  An automated ticket-issuing system sells rail tickets. Users select their destination and input a credit card and a personal identification number. The rail ticket is issued and their credit card account charged. When the user presses the start button, a menu display of potential destinations is activated, along with a message to the user to select a destination. Once a destination has been selected, users are requested to input their credit card. Its validity is checked and the user is then requested to input a personal identifier. When the credit transaction has been validated, the ticket is issued.

  Based on this:

  - Discuss any ambiguities or omissions in the statement of requirements for the part of a ticket-issuing system.

  - Write a set of non-functional requirements for the ticket-issuing system above. You are free to make assumptions regarding the system based on ambiguities or omissions you identified previously.

- Use your knowledge of how an ATM is used to develop a set of use cases that could serve as a basis for understanding the requirements for an ATM system. Construct a use case diagram to document your use cases.

- What would you do if you got conflicting answers about the same procedure from two different people you interviewed?

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.