Hyperiondev

TASK

Define your Product

Visit our website

Introduction

Welcome to The 'Define your Product' Task!

Before you start your final Capstone Project, there is one more area that needs your attention: defining your software product. It makes sense that, before you code a software solution, you should know exactly what the system is meant to do. Up to now, you have been focusing primarily on gaining the coding skills you need in order to be able to create a web application. There is more to it than that though. This task will equip you with some other skills you need to become a professional web developer.

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!

WEB DEVELOPMENT SKILLS

A full stack web developer needs to see the bigger picture. He/she needs to be able to understand, amongst other things, who will be using the system, how the users will be using the system, the goals and requirements of the system and the environment that the system will be running on. As you can see, a full stack web developer must be more than just a skilled coder. This task helps one to understand one of the non-coding skills that a full stack web developer needs - gathering, understanding and communicating the requirements associated with a web application.

REQUIREMENTS AND THE SOFTWARE DEVELOPMENT PROCESS

It is worth noting at this point, that the manner in which you go about gathering and documenting requirements, may be influenced by the software process (see the task: Introduction to Agile Development Process) you and your team choose to use.

No matter what software process you use though, there are two general tasks that need to be considered before you start programming:

- Understand the problem/requirements.
- Plan and design the solution.

The principles regarding what type of information you need to consider when analysing the problem and understanding the requirements of a system are the same for any software process. The process you use to translate those requirements into software and documentation is different though. This task highlights the requirements and information you need to consider when developing a system. It highlights 1) some traditional ways of documenting this information and 2) shows how the requirements collected are used for agile development.

UNDERSTAND THE REQUIREMENTS

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, such a purpose could be controlling a device, finding information or placing an order. Requirements engineering is the process of finding out, analysing, documenting and checking these requirements.

Requirements are generally divided into two categories:

- **User requirements**: These requirements describe what services the system is expected to provide to system users and the constraints under which it must operate.

- **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:

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

- **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 those 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. However, you can 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. The quality and aesthetic nature of content on a web application, to a large extent, influence the quality of the application.

- **Reliability**: These requirements describe the dependability of a system. Often a system exhibits such behaviours as service outages and incorrect processing. Reliability has to do with how a system detects and recovers from those problems. Web applications are expected to be available all day, every day. Web applications must also be designed to cope with unpredictable and varying loads. A website may have relatively few visitors on some occasions but massive amounts of users all using the site at the same time on others (e.g. imagine events like Black Friday or other special promotions designed to increase traffic to a website).

- **Performance**: These requirements describe operational characteristics related to measures of workloads, such as throughput and response time. This is a key requirement that web developers should take into account. If a user has to wait too long for a web page to load, they are likely to just navigate to another web page.

- **Security**: These requirements describe how access to the application will be controlled and how data will be protected during storage and transmission. Since web applications are designed to be accessed by users over a network, there are special security measures to be taken to ensure the security of web applications.

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 method 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. Web applications usually need to be updated almost continuously.

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. 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.

We will see how interviews and scenarios can be used for requirements discovery.

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 put questions to stakeholders about the system that they currently use and the system to be developed. Requirements are derived from the answers to these questions.

There are two types of interviews:

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

- **Open interviews**: In this type of interview, there is no predefined agenda. You explore a range of issues with system stakeholders so as 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 eliciting knowledge about organisational requirements and constraints.

Information from interviews should be used to supplement other information about the system from documentation describing business processes or existing systems, user observations. 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, supplemented by diagrams, screenshots etc.

PLAN THE SOLUTION

Once you have a clear understanding of the system requirements, you can start planning your system. With an XP process, customers are very involved in this process. The customer is, in fact, considered a vital part of the development team and also discusses scenarios with the other members of the team.

All team members then develop a "story card." Story cards are used to plan and describe the desired output and functionality of the system. The development team works together to prioritise the stories. The development team also decides how to group stories together and which stories are going to be included in the next software release. This process is useful in ensuring requirements validation.

A note from our coding mentor
Jared

Watch this short video to see how user stories are used and what makes a good user story. In addition, this video provides some practical examples of story cards.

Requirements Validation

Requirements validation is the process of checking that requirements actually define the system that the customer really 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 requirements problem by making changes to a system is usually much greater 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 that the document should not contain any contradictory constraints or different descriptions of the same system function.

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

- **Realism checks**: Using knowledge of existing technology, the requirements should be checked to ensure that they can actually be implemented.

- **Verifiability**: To reduce the potential for a dispute with the customer, system requirements should always be written down 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.

An important requirements validation technique is test-case generation. Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements 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. With XP, acceptance tests are developed as story cards are developed.

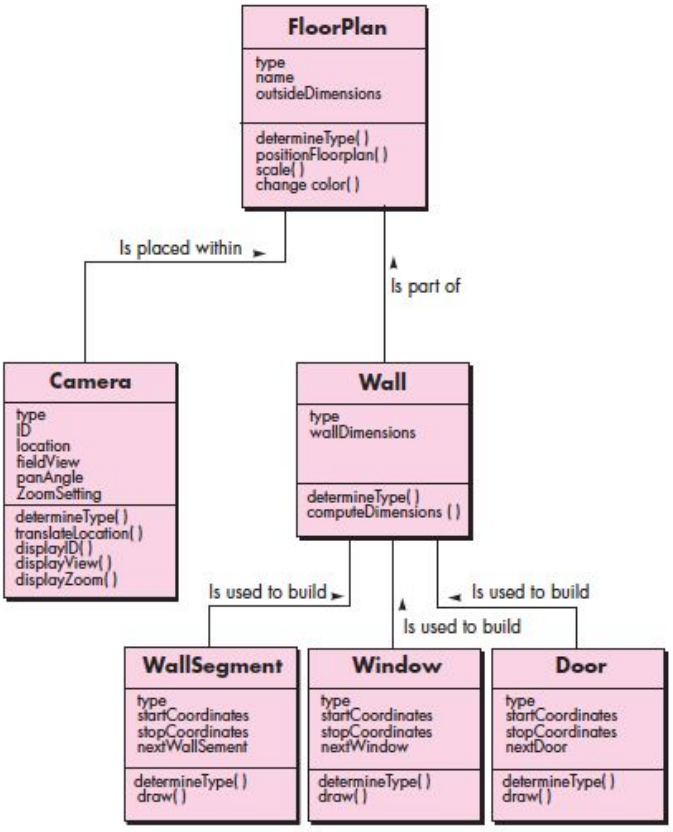
It is very difficult to show that a set of requirements meets a user's needs. Users need to picture the system in operation and imagine how that system would fit into their work. As a result, you rarely find all requirements problems during the requirements validation process. It is inevitable that there will be further requirements changes to correct omissions and misunderstandings after the requirements document has been agreed upon. That is one of the reasons why with XP (and other agile development processes), systems are built in small increments and then used to test and verify that requirements are clearly understood and implemented.

DESIGN THE SOLUTION

The software process that you choose to use will once again affect the extent of the activities that you carry out here. You may choose to follow an approach where your anticipated software solution is extensively designed and the designs are documented in detail. Agile development, on the other hand, advocates a keep it simple (KIS) approach to design since the system is developed in increments instead of designing the whole system and all its functionality at once. In each iteration of the design phase, the design should give guidance for the implementation of a story. No matter what software process you choose to adhere to, the design phase will likely include the activities discussed in this section.

CRC modeling

Class-Responsibility-Collaborator modeling is a tool that is often used in the agile development of web applications. An example of a CRC diagram is shown below.



CRC modeling is used for the design of object-oriented programs. Basically, CRC models give information about:

- All **classes** that will make up a system. Remember what you have learned about OOP? A class is a blueprint for an object. Wall, Window, Door etc in the diagram above are all examples of classes.

- **Responsibilities**: Remember that all classes consist of attributes and methods. These attributes and methods are known as responsibilities. Responsibilities are, therefore, everything that a class knows or does.

- **Collaborators**: Collaborators are classes that need to provide information for another class to be able to carry out a responsibility. Collaborators show the interaction between classes.

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.

In extreme programming (XP), it is recommended that operational prototypes be developed for difficult design problems. The design prototype that is implemented and evaluated is known as a spike solution. Even when using other software processes, prototyping is often used.

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.

Prototyping is also an essential part of the user interface design process. Textual descriptions and diagrams are not good enough for expressing the 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.

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.

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 into 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 be used to refer 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, place more of a burden on the user mentally by forcing them to guess what is within the drop-down menu or what the element might be, for example.

Questions that your web user interface should answer

Pressman highlights that a key consideration for the design of a UI for a web application is being able to answer the following questions:

- **Where am I?** The UI should let the user know which web application has been accessed and where in the content hierarchy the user currently is.

- **What can I do now?** The user should be able to clearly understand what they can do currently. The user should know what functions are available, what links are live and what content is relevant.

- **Where have I been; where am I going?** The web application should be easy to navigate. The user should be provided with a "map" that shows where they have been in the web application and how they can navigate the rest of the application.

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. UI design should promote user efficiency. The user should also always know where they are and how to go back to the previous page to retrace their steps.

- **Strategically use color and texture**: You can direct attention toward, or redirect attention away from items, by using colour, 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 users. 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

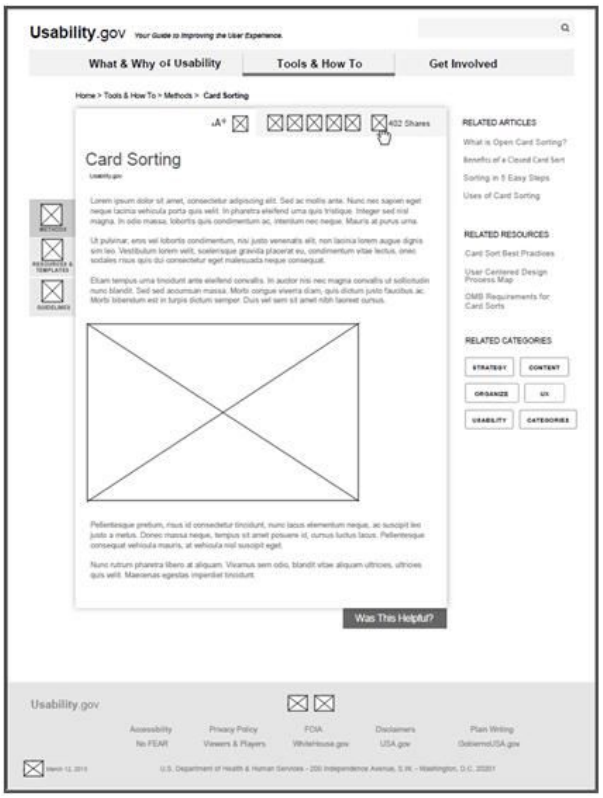
system is processing so that the user doesn't repeatedly press a button because they think your system isn't responding.

- **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 tools that are used extensively for the design of UIs for web applications. 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 behaviours. Wireframes typically do not include any styling, colour 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. Keep it simple 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.

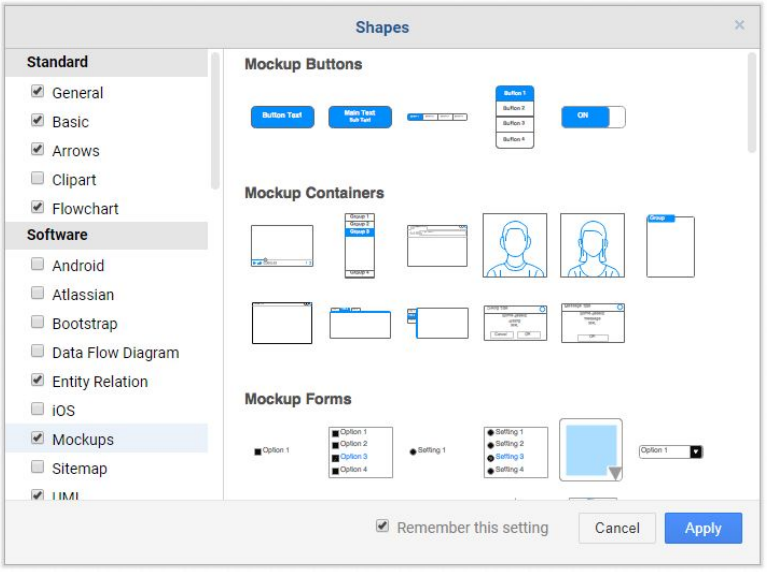
- Logo
- Search field
- Breadcrumb
- Headers, including page title
- Navigation systems, including global navigation and local navigation
- Body content
- Share buttons
- Contact information
- Footer

A note from our coding mentor
Jared

There are various tools that can be used to create wireframes. One such tool is found here. To use draw.io to create wireframes, make sure that you add the shapes used to create "Mockups". To do this:

- *Open https://www.draw.io/*
- *When a blank, untitled diagram is opened, click on the "More Shapes" button at the bottom left-hand corner of the screen.*

- *From the dialog that appears select "Mockups" (in the "Software" category)*

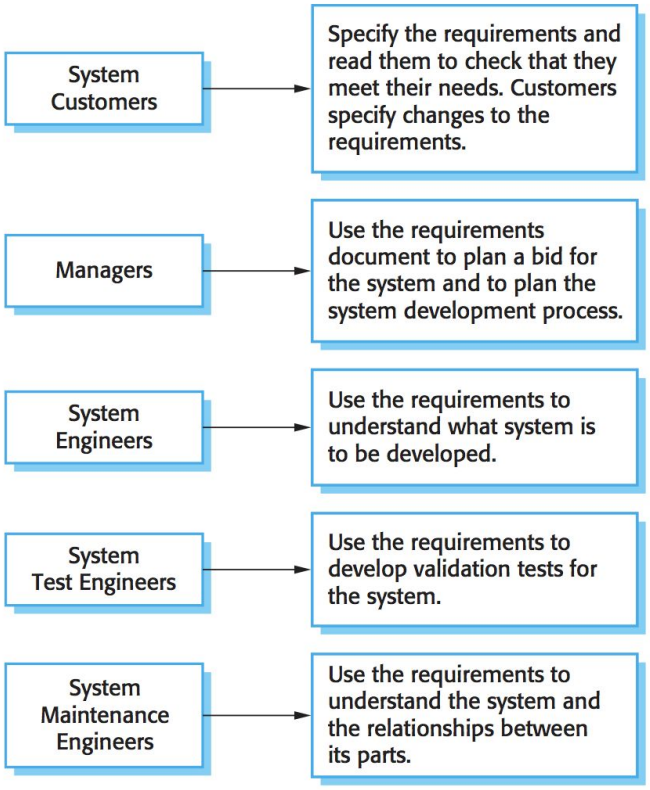


THE SOFTWARE REQUIREMENTS DOCUMENTATION

Traditional software processes often require the development of a lot of documentation during the planning and designing phases. This is not true of agile development. As mentioned previously, agile development software processes advocate a keep it simple (KIS) approach to design. With agile development, documentation is usually kept to a minimum during the design and planning phases of the software process. Besides the documents discussed in this task (user stories, CRC models and wireframes), there is not usually much other documentation generated as the system is planned and designed. As code (including prototypes) is implemented though, there should also be clear internal documentation (comments that explain the code).

More traditional software processes often encourage the development of a 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. The diagram below shows possible users of the document and how they use it:



Users of a Requirements Document (Sommerville, 2010)

Since 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 standard is a generic standard that can be adapted to specific uses.

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components, the system, and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Structure of a Requirements Document (Sommerville, 2010)

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.

Thinking in terms of a software product

It is worth noting at this stage, that a role that is becoming more valued and sought after in many companies is the role of a product manager. Although product management is beyond the scope of this bootcamp, it is of value to learn to think of some web applications as products.

A software product is a product that is mainly made up of software that can be "offered to a market for attention, acquisition, use or consumption that might satisfy a want or need" (Kotler et al 2010). The market for a software product is a group of people or organisations with similar needs that will pay for the product. Since there are so many different software products available today, software product designers and software product managers have ever-increasing value to organisations by making sure that their software product remains competitive and can be marketed in such a way as to make a profit. According to Fricker, "Software product management is the discipline which governs a software product from its inception to its close-down to generate as large value as possible for the business" (2012).

Compulsory Task 1

Follow these steps:

- Imagine that you are required to create a Task Management web app. Your web app should be able to be used to allow a user to:
    - Sign in
    - Add tasks
    - Assign tasks to themselves or other users
    - Delete tasks
    - Edit existing tasks
    - Mark tasks as complete
    - Filter tasks by:
        - Completed tasks
        - Tasks that must still be completed
        - Deadline
        - Tasks that are overdue

- For this task, you are required to design and plan your Task Management web app. In order to do this:
    - Analyse some existing web applications that do something similar to what you want to accomplish. e.g. Asana
    - Submit:
        - A document that lists the functional and nonfunctional requirements of your web application.
        - A wireframe showing the UI of your website.
        - At least 5 user stories.
        - A document that briefly describes how you are going to make your website (software product) stand out from those of your competitors.

Once you have completed the task in line with the instructions above, click the button below to request your mentor to review your work and provide feedback. If

you have any questions while attempting the task, leave your mentor a note on the comments.txt file in your Student Dropbox folder.

Completed the task(s)?

Ask your mentor review your work!

Review work

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.