Software Requirements Handout

CSC 322 Software Engineering
ENGR 212 Project Management

Project: Quarto!

Quarto! is a two-player board game by GiGaMic. It is like a 4x4 version of tic-tac-toe that a player wins by getting four in a row. There are 16 different game tokens, each with a unique combination of four binary characteristics. This project creates a top-down-view video game that allows two human players to play against each other on the same computer. Players will share the same mouse/keyboard/monitor, taking turns back and forth.

Requirement Description: Provide a top-down display of the game board for players to interact with. The display shall be the primary interface for players to place tokens on the board. During a turn, one player will choose a token to play and the other player will place it on the board.

Functional Requirements:

- The display shall draw a 4x4 board with all of the slots initially empty, shown in a uniform and subdued style.
- The display shall identify the active player by name.
- The display shall direct the active player either to choose a token or to place a token on the game board, as dictated by the rules of the game.
- When a player is selecting a token for his opponent to play, the display shall highlight the nearest valid token from the unplayed token area, if it is within 10 pixels of the cursor.
- When a player places a token on the board, the display shall highlight the nearest valid empty slot, if it is within 10 pixels of the cursor.
- Occupied slots on the 4x4 board shall show their token instead of an empty slot.
- *There's probably more we could add…*

Non-Functional Requirements:

- The game board shall respond in real-time to all cursor movement.
- The 16 tokens with their four binary characteristics shall be drawn distinctly enough that an eight-year-old child can quickly distinguish between them.
- The display shall be accessible across mobile, tablet, and laptop, and workstation devices and screen sizes, maintaining consistency and functionality.

Acceptance Criteria:

- The display must accurately reflect the current state of the game board by accurately identifying winning conditions, tie conditions, and unfinished conditions 100% of the time.
- At least 90% of all users in a usability test group should be able to uniquely identify common characteristics between the game board tokens.
- At least 60% of eight-year-old users in a usability test group should be able to uniquely identify common characteristics between the game board tokens.
- Performance testing must validate that the display responds within 0.5 seconds when selecting and placing a token.

[Comment: *The traditional verb for all requirements is the word "shall", which you can see in this example. Acceptance criteria often use the word "must" because it is a test that verifies "shall".*]

Definitions for the Top-10 Software Requirements Terms

This list came from ChatGPT 3.5, although Prof. Tallman edited it a little bit

1. Functional Requirements: Specifications that define what the software system should do, outlining its functionalities and capabilities.

2. Non-Functional Requirements: Specifications that define the quality attributes of the software, such as performance, security, usability, and scalability.

3. Use Case: A description of how users interact with the system to accomplish a specific goal, often represented as a sequence of actions.

4. User Story: A concise description of a feature or requirement from an end-user perspective, typically used in Agile methodologies to capture functionalities.

5. Requirement Elicitation: The process of gathering, discovering, and documenting software requirements from stakeholders, users, and other sources.

6. Traceability: The ability to track and document the relationships between different requirements, ensuring they are all accounted for and aligned with the project objectives.

7. Prioritization: The process of ranking requirements based on their importance, criticality, and impact on the project's success.

8. Validation: The process of ensuring that the documented requirements accurately represent the needs and expectations of the stakeholders.

9. Scope Creep: Uncontrolled changes or additions to the project's scope, often arising from poorly managed or undocumented requirements changes.

10. Acceptance Criteria: Conditions that must be met to consider a requirement complete and satisfactory, used as a benchmark for testing and acceptance of delivered functionalities.

The 6 Most Important Software Requirements Practices

Karl Wiegers, author of *Software Requirements Essentials* (full article)

Practice #1: Define business objectives

Organizations undertake a project to solve a problem, exploit a business opportunity, or create a new market. Defining the project's business objectives communicates to all participants and other stakeholders why they are working on the project.

Practice #2: Understand what users need to do with the solution

I strongly advocate taking a usage-centric approach to requirements development and solution design, rather than a feature- or product-centric approach. When you focus on exploring features rather than user goals, it's easy to overlook some necessary functionality. It's also easy to include functionality that seems cool but doesn't help users get their jobs done. Use cases are an effective technique for maintaining this usage-centric mindset.

Practice #3: Prioritize the requirements

I doubt that any project has ever implemented every bit of requested functionality. Even if you could implement it all, you can't do it all at once. Your goal is to deliver the maximum business value to your customers at the lowest cost and in the shortest time. Achieving this goal demands that you prioritize requirements so the team can work on them in the most appropriate sequence.

Practice #4: Elicit and evaluate quality attributes

People naturally focus on a product's functionality when discussing requirements, but those are only part of the solution. Nonfunctional requirements contribute heavily to user satisfaction and suitability for use. When speaking of nonfunctional requirements, people most commonly think of quality attributes.

Practice #5: Review and test the requirements

How do you know if your requirements are accurate? How can you tell if there are clear enough so all the team members know what to do with them? No matter how you choose to represent requirements, it is sometimes ambiguous, incomplete, or simply incorrect. One of the most powerful quality practices available is peer review of requirements. Convene some colleagues to review both textual requirements and diagrams. Different project participants — designers, developers, testers, users — will find different kinds of problems during these reviews.

Practice #6: Manage changes to requirements effectively

No matter how well you understand the problem and how carefully you prepare the requirements, they won't be perfect, complete, or static. The world changes around us as we work. An incremental, agile approach is a good way to cope with this. You plan to build the requirements -- and the solution -- in a series of small chunks, expecting the direction to change and accepting the uncertainty of what you'll have at the end and when you'll have it.

Techniques to deal with ambiguous requirements

Dani Kahil, Independent Consultant and Microsoft MVP (full article)

- ***Use examples***: For each requirement that is not clear to you, ask users to walk you through a real example. While doing so, add more details to your requirement. If you use User Stories, examples will help you confirm or write down the Acceptance Criteria.

- ***Replace or remove ambiguous words***: Ask yourself the question "How can I test that requirement?". If your requirement contains words like "Easy to use", "Robust", "Faster", "Many", etc. how are you supposed to test and mark it as passed or failed? A classic example of an ambiguous requirement is "The system must have a USER-FRIENDLY user interface". What is "USER-FRIENDLY" for John is not the same for Mary.

- ***Use visual tools***: Elicit requirements using appropriate diagrams, proof of concepts or demo. You are probably familiar with the saying "A picture is worth a thousand words".

- ***Compare options***: If there is ambiguity in your requirement, showcase some options to your client to help them clarify their thoughts.

- ***Ask WHY it's important***: The WHY will give you more context and meaning as to why this requirement is so important to your users. I recently listened to a podcast explaining how our brain is constantly looking for meaning subconsciously. The speaker was explaining that if we don't fully understand the meaning of a requirement, our brain will make something up. Which can be completely the opposite of what our user had in mind in the first place.

- ***Storytime Workshops***: Elaborate and refine User Stories further during storytime workshops. Use a combination of the above techniques to make those sessions as productive as possible.