Personal Project

Golf Game Requirements Specification

Team Members: Gurdeep Singh, Jagjit Singh, Maruf Tonmoy

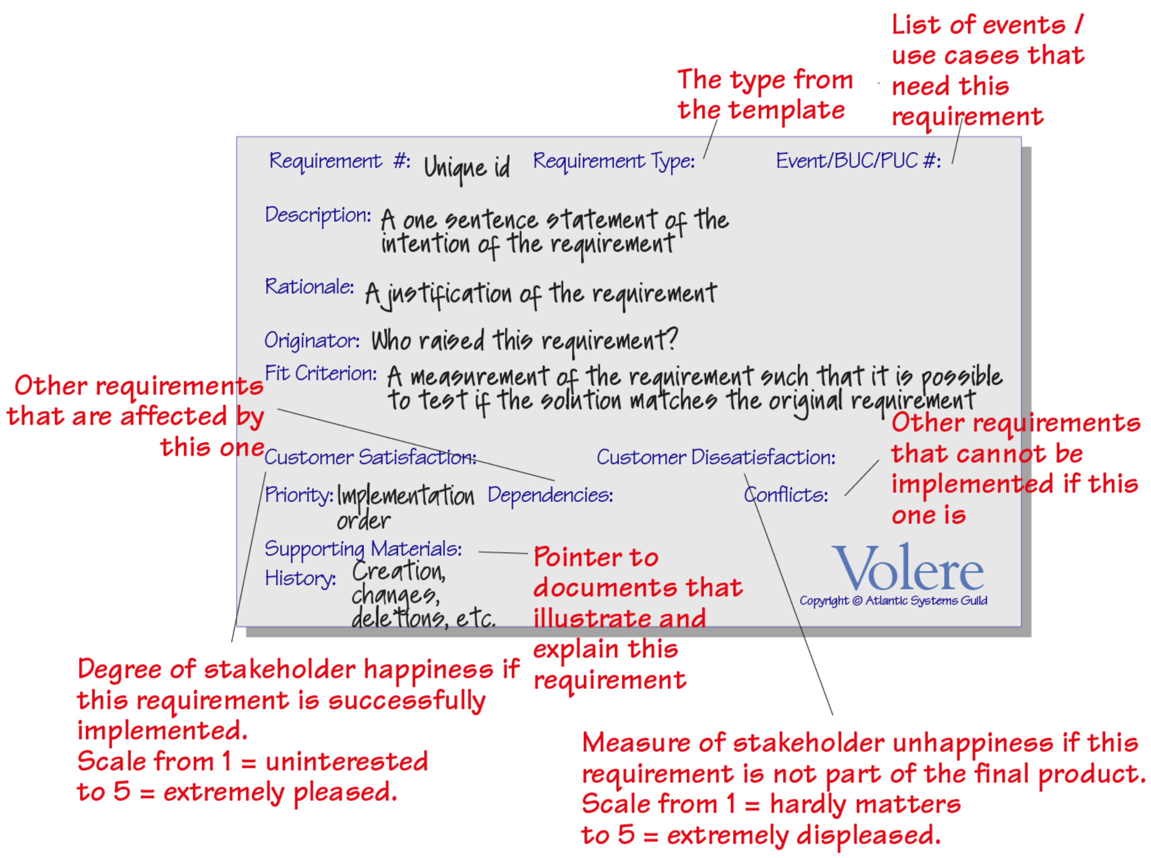
This template is based on the Volere Requirements Specification Template v.20

Atomic Requirements Shell

The requirements shell is a guide to writing each atomic requirement. The components of the shell (also called a "snow card") are identified below. An atomic requirement is made up of this collection of attributes.

An example snow card is shown below, however for the project you need only focus on the following:

1. *Requirement Id.*
2. *Requirement Type. (this will be expressed by the section that the requirement appears in)*
3. *Use Case Id*
4. *Description.*
5. *Rationale*
6. *Fit Criterion*
7. *Priority*



Requirement #: Unique id
Requirement Type:
Event/BUC/PUC #:

Description: A one sentence statement of the intention of the requirement

Rationale: A justification of the requirement

Originator: Who raised this requirement?

Fit Criterion: A measurement of the requirement such that it is possible to test if the solution matches the original requirement

Customer Satisfaction:
Customer Dissatisfaction:

Priority: Implementation order
Dependencies:
Conflicts:

Supporting Materials:

History: Creation, changes, deletions, etc.

Volere
Copyright © Atlantic Systems Guild

The type from the template

List of events / use cases that need this requirement

Other requirements that are affected by this one

Other requirements that cannot be implemented if this one is

Pointer to documents that illustrate and explain this requirement

Degree of stakeholder happiness if this requirement is successfully implemented.
Scale from 1 = uninterested to 5 = extremely pleased.

Measure of stakeholder unhappiness if this requirement is not part of the final product.
Scale from 1 = hardly matters to 5 = extremely displeased.

1. The Purpose of the Project

Goals of the Project

This place To create a card game which can be played on command terminal for customers that is easy to play. Anyone with basic knowledge of computers should be able to play this card game, as it's going to be self explanatory steps to play it.

Requirement To Play The Game

- 52 cards needed to play the game. 2-4 players. Score keeping sheet.

Rules of Golf Card Game

- When a player replaces a card from the deck, the replaced card has to be discarded into a discard pile either face up or down.
- When the draw deck runs out, shuffle the discard pile to make a new deck.

Game Rules

- It is to score the lowest points. Each card has a certain point value if it is unmatched. Matched cards in the same column result in no points.

Game play

- On every Player's first turn, the player flips over two cards from their hand.
- After the players first turn they flip only one card from their hand.
- After the first round, the player then chooses to draw the top card from the discard pile.
- Either player can choose to replace the drawn card with one of the cards in their hand and discard the replaced card or discard the drawn card.
- Play continues this way until a player has all their cards facing up.
- Then the game ends, everyone shows their card and count points.

Describe what you want the game to do. Do not be too wordy in this section—a brief explanation of the project's goals is usually more valuable than a long, rambling treatise. A short, sharp goal will be clearer and improves the chances of reaching a consensus for the goal.

A sentence or a graph or diagram that quantifies how you will measure whether or not the goals have been achieved. Any reasonable goal must be measurable. This is necessary if you are ever to test whether you have succeeded with the project. For example, if the goal of the project is

> **We want to give immediate and complete response to customers who order our goods online.**

you have to ask what advantage meeting that goal brings to the organization. If immediate response will result in more satisfied customers, then the measurement must quantify that satisfaction. For example, you could measure the increase in repeat business (on the basis that a happy customer comes back for more), the increase in customer approval ratings from surveys, the increase in revenue from returning customers, and so on.

2. The Stakeholders

This section describes the stakeholders—the people who have an interest in the product. It is worth your while to spend enough time to accurately determine and describe these people. This will help when you create the user stories for the project.

The Client

The customer who assigned us this project.

The Players

Players who are going to play this game. They should be able to play this game with basic computer knowledge.

Development Team (or Other Stakeholders)

Designers

They will be responsible for making UML/sequence diagrams for the program. This design will be used by the developer to do the implementation of a functional application.

Developers

They will be responsible for developing the program according to the design documentation drafted by the designers to implement the product in C++ to create the functional software.

Quality Assurance/Testers

They will be responsible to test the software and report to the development team about any issues, bugs or improvement that could be made.

Testers will employ the use of unit testing using the Google C++ testing framework (gtest) in order to discover any bugs or unintended behaviour in the product. //i know we are not gonna use tests but writing it just for documentation.

User Priority

Attach a priority to each category of user. This identifies the importance and precedence of the user. Prioritize the users as follows:

1. *Key users: They are critical to the continued success of the product. Give greater importance to requirements generated by this category of user.*
2. *Secondary users: They will use the product, but their opinion of it has no effect on its long-term success. Where there is a conflict between secondary users' requirements and those of key users, the key users take precedence.*
3. *Unimportant users: This category of user is given the lowest priority. It includes infrequent, unauthorized, and unskilled users, as well as people who misuse the product.*

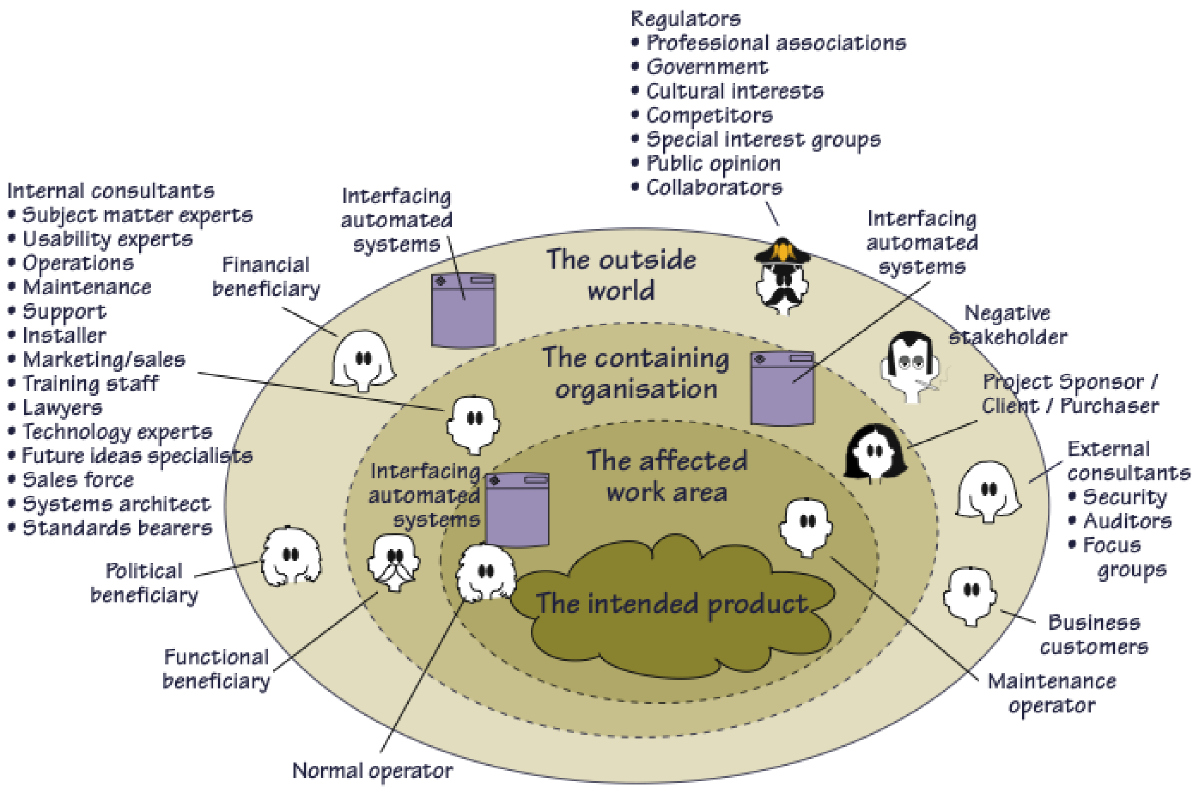
The percentage of the type of user is intended to assess the amount of consideration given to each category of user.

Development Team (or Other Stakeholders)

Consider the needs of the people who are going to develop the software. These can include (but are not necessarily limited to) the following development roles:

1. *Usability experts*
2. *Designers and developers*
3. *Testers*
4. *Software engineers*
5. *Technology experts*
6. *System Maintainers*

You may find the following diagram helpful in thinking about the other stakeholders.



3. Constraints

This section describes constraints on the eventual design of the product. Think of a constraint as: no matter how you solve the problem this must be satisfied.

Constraints are global—they are factors that apply to the entire product. The product must be built within the stated constraints. Often you know about the constraints, or they are mandated before the project gets under way. They are probably determined by management and are worth considering carefully—they restrict what you can do and so shape the product. Constraints, like other types of requirements have a description, rationale, and fit criterion, and generally are written in the same format as functional and non-functional requirements.

Solution Constraints

This specifies constraints on the way that the problem must be solved. Describe the mandated technology or solution. To make the constraint specific include any appropriate version numbers. You should also explain the reason for using the technology.

These constraints guide the development of the final product. Your client, customer, or user may have design preferences, or only certain solutions may be acceptable. If these constraints are not met, your solution is not acceptable.

We want to define the boundaries within which we can solve the problem. Be careful, because anyone who has experience with or exposure to a piece of technology tends to see requirements in terms of that technology. This tendency leads people to impose solution constraints for the wrong reason, making it very easy for false constraints to creep into a specification. The solution constraints should only be those that are absolutely non-negotiable. In other words, however you solve this problem, you must use this particular technology. Any other solution would be unacceptable.

Present the constraints as a subsection (or the like) with the following items:

7. *Description*
8. *Rationale*
9. *Fit Criterion (i.e. how will you know the constraint is met?)*

Examples

Constraints are written using the same form as other atomic requirements (refer to the atomic requirements snow card/shell for the attributes). It is important for each constraint to have a rationale and a fit criterion, as they help to expose false constraints (solutions masquerading as constraints). Also, you will usually find that a constraint affects the entire product rather than one or more product use cases.

The product shall use the current two-way radio system to communicate with the drivers in their trucks.

Rationale: The client will not pay for a new radio system, nor are any other means of communication available to the drivers.

Fit criterion: All signals generated by the product shall be audible and understandable by all drivers via their current two-way radio system.

The product shall operate using Windows XP.

Rationale: The client uses XP and does not wish to change to a later version.

Fit criterion: The product shall be approved as XP compliant by the MS testing group.

3g. Budget Constraints

This section shows the budget for the project, expressed in available resources (i.e. project personnel and time). The requirements must not exceed the budget. This limitation may constrain the number of requirements that can be included in the product.

The intention is to restrict the wildest ambitions and to prevent the team from gathering requirements for a AAA 3D first-person shooter when the budget is only for a text-based command-line game.

4. Naming Conventions and Terminology

All projects have their own unique vocabulary usually containing a variety of acronyms and abbreviations. Failure to understand this project-specific nomenclature correctly inevitably leads to misunderstandings, hours of lost time, miscommunication between team members, and ultimately poor-quality specifications.

4a. Glossary of All Terms, Including Acronyms, Used by Stakeholders Involved in the Project

Face value: A value which is printed on the card. It could range from the number 2 to 10 or it could be one of A, K, J and Q.

Points: These are number values which help in deciding who wins the game. Every card is assigned a value as follows:

Face value	**Points**
2-10	equal to face value
A	1
K	0
J, Q & 10	10

Shuffle: It is to mix up the playing cards in a draw pile so as to randomise the order in which they appear.

Deal: To distribute the cards to all the players before the beginning of a round.

Round: When one whole circle of each player turns ends.

Set: After revealing all the cards of each player ,the game decides a winner through summing all the points of each respective player.

This period and phase is called a Round.

Turn / move : When a player draws a card from a draw pile or shows one card.

Draw:

Deck (full deck):

Discard Pile: This is the deck where we discard cards after the play.
Draw Pile: We draw cards from this pile to play the game.
Total Score: Sum of all the points at the end of thegame to check who is the winner.

Players: The participants who are playing the game including the computer(AI).

Player hand: A set of cards in custody of the playerat the given moment.

Reveal card: It is a function to show one of the invisible cards from the player's hand.

Winner: the player who has a lower total score at the end of the game.

5. Assumptions

A list of the assumptions that the developers are making. These assumptions might be about the intended operational environment, but can be about anything that has an effect on the product. As part of managing expectations, assumptions also contain statements about what the product will ***not*** do.

We often make unconscious assumptions. It is necessary to talk to the members of the project team to discover any unconscious assumptions that they have made. It is important to state these assumptions up front. You might also consider the probability of whether the assumption is correct and, where relevant, a list of alternatives if something that is assumed does not happen.

Examples

1. *Assumptions about what your developers expect to be ready in time for them to use—for example, other parts of your products, the completion of other projects, software tools, or software components.*
2. *Assumptions about the technological environment in which the product will operate. These assumptions should highlight areas of expected compatibility.*
3. *The software components that will be available to the developers.*
4. *Dependencies on computer systems or people external to this project.*
5. *The requirements that will specifically not be carried out by the product.*

7. Functional Requirements

This is a specification for each atomic functional requirement. As for all types of atomic requirements (functional, non-functional, constraint), use the requirements shell as a guide for which attributes should be specified. A full explanation of the atomic requirement and its attributes is included in this template's introductory material.

The use cases will help you to determine the functional requirements.

Include the cost of requirements, which is the effort that you have to spend building them into a product. Once the requirements specification is complete, you can use an estimating method (e.g. Planning Poker) to assess the cost, expressing the result in story points.

8. *Requirement Id.*
9. *Requirement Type. (this will be expressed by the section that the requirement appears in)*
10. *Use Case Id*
11. *Description.*
12. *Rationale*
13. *Fit Criterion*

14. *Priority*

Example

1. A user should be able to start a new game from the prompt with cards dealt to the user or exit the game.

Priority or Implementation Order	1
Use Cases	1,2, last use case TBD
Rationale:	Allow the user to have a choice to start or exit the game.
Fit Criterion	Users can input from the keyboard to either start a game or exit.
Story Points	TBD

2. The player 1 and player 2 should be dealt the right amount of cards at the start of the game.

Priority or Implementation Order	2
Use Cases	2

Rationale:	Cards will be dealt to each player at the start of the game.
Fit Criterion	Player 1 and player 2 should be dealt 6 cards at the start of the game to play it correctly.
Story Points	tbd

3. A functional game of Golf should be carried out to allow by player 1 and player 2 to have a succession of turns until a winner is determined which has the least score at the end of the game.

Priority or Implementation Order	3
Use Cases	TBD
Rationale:	The users can play a complete and functional Golf card game within the command terminal.
Fit Criterion	The implementation of the game allows the user to complete a game by rotating turns of the player until any player reveals all his cards' condition is met. Then points are added and a player with less score is the winner.
Story Points	TBD

4. A user should be able to start a new game from the prompt with cards dealt to the user or exit the game.

Priority or Implementation Order	1
Use Cases	1,2, last use case TBD
Rationale:	Allow the user to have a choice to start or exit the game.
Fit Criterion	Users can input from the keyboard to either start a game or exit.
Story Points	TBD

Priority or Implementation Order	5
Use Cases	7, 9
Rationale:	To be able to schedule untreated roads and highlight potential dangers
Fit Criterion	The recorded treated road shall agree with the driver's road treatment logbook and shall be up to date within 30 minutes of the completion of the road's treatment.
Story Points	3

Non-functional Requirements

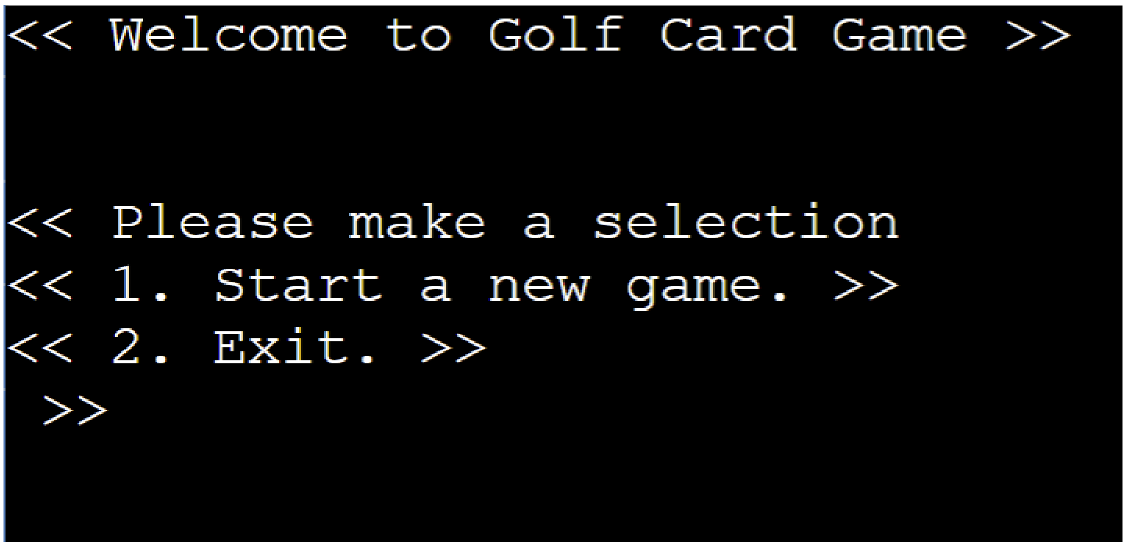
The following section(s) describe the non-functional requirements. The form of these requirements is the same as for the functional requirements as described in section 9 above. Your team can add additional non-functional sections as you see a need.

5. A user should be able to start a new game from the prompt with cards dealt to the user or exit the game.

Priority or Implementation Order	tbd
Use Cases	tbd
Rationale:	The customer or player can play the quick game of Golf
Fit Criterion	Load time of the game at any given time should not be more than 3 seconds to avoid unnecessary wait time.
Story Points	tbd

8. Look and Feel Requirements

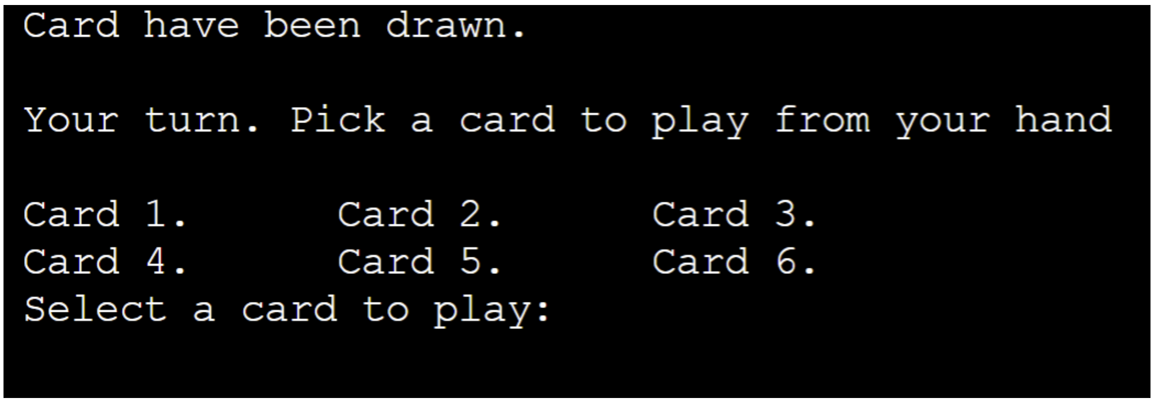
The section contains requirements relating to the spirit of the product. This section captures the requirements for the appearance.



```
<< Welcome to Golf Card Game >>


<< Please make a selection
<< 1. Start a new game. >>
<< 2. Exit. >>
 >>
```

Include images of high-fidelity, horizontal prototypes of the expected interface, with accompanying descriptions



```
Card have been drawn.

Your turn. Pick a card to play from your hand

Card 1.       Card 2.       Card 3.
Card 4.       Card 5.       Card 6.
Select a card to play:
```

.

9. Risks

As with developing any project, there is always the risk of something going wrong. It is not necessarily a bad thing unless ignored by the developers. Some common risks are inaccurate estimations, missing project deadlines, missing members and so on.

All projects involve risk—namely, the risk that something will go wrong. Risk is not necessarily a bad thing, as no progress is made without taking some risk. Risk is only a bad thing if the risks are ignored and they become problems. Risk management entails assessing which risks are most likely to apply to the project, deciding a course of action if they become problems, and monitoring projects to give early warnings of risks becoming problems.

This section of the specification contains a list of the most likely and the most serious risks for your project. For each risk, include the probability of it becoming a problem and any contingency plans.

Risks will undoubtedly change during the lifetime of a project. The better you understand the requirements the better you can identify which risks are most serious for your project. The project manager determines how to manage the risks but the requirements specialists and developers provide input on new risks and which risks are turning into problems.

Use your knowledge of the requirements as input to discover which risks are most relevant to your project. Also, consider the following categories of project risks:

1. *Requirements/Design/Estimation*
    a. *The team planned a project that is too large (i.e. "eyes bigger than stomach").*
    b. *The team underestimated how long parts of the project would take.*
    c. *Major changes to design are needed during implementation.*
2. *People*
    a. *Addition or loss of team member (i.e. someone dropped the course, a new person joins the team)*
    b. *Unproductive team member(s)*
    c. *Team member(s) lacking expected technical background (e.g. don't know C++)*
    d. *Major life events (e.g. long-term illness, death of family member, birth of a child)*
3. *Learning & Tools*
    a. *Inexperience with new tools*

b. *Learning curve for tools steeper than expected*
c. *Tools don't work together in an integrated way*