Software Masters

Project Plan

June 20, 2016

Team

Rachel Glockenmeier

Marc Julian

Rob Miceli

John Parrish

Melisa Satili

Heather Zellner

Software Masters

Project Plan

Contents

---

1. Work-breakdown structure (WBS)

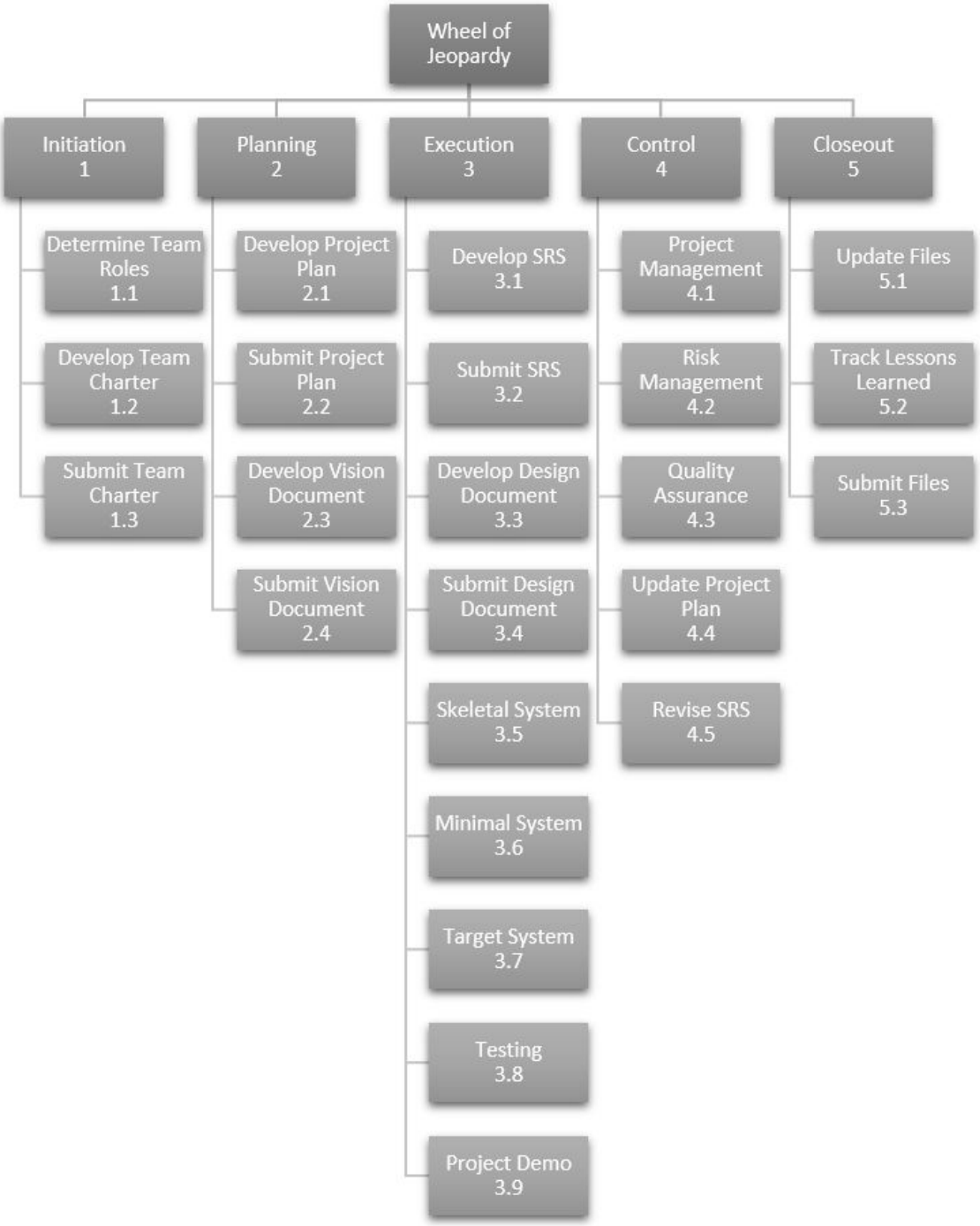

Figure 1: *Work Breakdown Structure*

Level	WBS Code	WBS Code	Definition
-	-	Wheel Of Jeopardy	All work to implement the Wheel of Jeopardy game system.
1	1	Initiation	The work to initiate the project.
2	1.1	Determine Team Roles	Project Team meeting to determine which members will take on which roles for the project.
2	1.2	Develop Team Charter	Project Team to develop the Team Charter.
2	1.3	Submit Team Charter	Team Charter is delivered to the
1	2	Planning	The work for the planning process for the project.
2	2.1	Develop Project Plan	Project Team to develop the Project Plan.
2	2.2	Submit Project Plan	The Project Manager submits the Project Plan for approval.
2	2.3	Develop Vision Document	Project Team to develop the Vision Document.
2	2.4	Submit Vision Document	The Project Manager submits the Vision Document for approval.
1	3	Execution	The work involved to execute the project.
2	3.1	Develop SRS	Project Team to develop the Software Requirements Specification.
2	3.2	Submit SRS	The Project Manager submits the Software

			Requirements Specification for approval.
2	3.3	Develop Design Document	Project Team to develop the Design Document.
2	3.4	Submit Design Document	The Project Manager submits the Design Document for approval.
2	3.5	Skeletal System	Project Team to complete Skeletal System and validate architecture.
2	3.6	Minimal System	Project Team to complete Minimal System and prove out architecture.
2	3.7	Target System	Project Team to complete Target System and ensure all specifications have been met.
2	3.8	Testing	Project Team to conduct series of tests on
2	3.9	Project Demo	Project Team will record a demo the final system.
1	4	Control	The work involved for the control process of the project.
2	4.1	Project Management	Overall project management for the project.
2	4.2	Risk Management	Risk management efforts as defined in the Risk Assessment Plan in the Project Plan.
2	4.3	Quality Assurance	Quality assurance efforts as defined in the Quality Assurance section in the Project Plan
2	4.4	Update Project Plan	Project Team updates the Project Plan as the project progresses.
2	4.5	Revise SRS	Project Team revises the Software

			Requirement Specifications as the project progresses.
1	5	Closeout	The work to close out the project.
2	5.1	Update Files	All files are updated to reflect the final Wheel of Jeopardy game system.
2	5.2	Track Lessons Learned	Project Team discusses lessons learned for the project.
2	5.3	Submit Files	All project related files and documents are submitted.

Table 1: *Work Breakdown Structure Dictionary*

2. Software Increments

2.1 Skeletal

For the first increment of software development, we plan to write pseudocode and skeletal functions that convey the program functionality without fully implementing the details of each sub-system. This will provide an underlying capability to plan for the software architecture and layout. At the end of the skeletal increment, we will have:

- A main() method that contains pseudocode for calling the highest level functionality
- Empty methods that are appropriately name based on their functionality and contain short comments explaining their purpose
- A file structure that organizes all the methods by their functionality. For example, all the GUI methods could be contained in the same file
- Identification of all subsystems and the messages between them

2.2 Minimal

The minimal system involves development of each subsystem to prove out the architecture. Software Masters will ensure the most important functionality of the system is implemented and functional. This version will have a minimal GUI and mostly text-based interactions. Additionally, a single set of questions and corresponding answers will be used to prove the board functionality. This implementation will allow at least two players to participate to prove the turn-based system.

On a user's turn, one of the categories on the wheel will be automatically chosen at random as their "spin". The category and question will appear to the user who can then type in an answer. The system will then validate the user's answer against the stored correct answer; points will be added or subtracted appropriately. If the user does not answer within a specified time limit, the system will move on to the next player's turn. For opponent and player's choice wheel sectors, the appropriate player will be prompted to type in the category of their choice.

Each user's score will be kept track of on a scoreboard which will also include number of free spin tokens. The system will automatically end the round when 50 "spins" of the wheel have occurred or no questions remain unanswered. At the end if the game, the user with the most amount of points will be displayed as the winner. A game will consist of a single round for this implementation to prove the minimal system's architecture operates as expected.

Upon reaching this milestone, risks are assessed, the Project Plan and SRS are updated, and the target system design is modified.

2.3 Target

The target system Software Masters plans to implement includes the following features:

- GUI that displays the wheel, Jeopardy! Question board, scoreboard, and the players participating in the game.
- Wheel with 12 distinct sectors to land on. The sectors will be randomly distributed on the wheel. These sectors include:
    - Six categories that correspond to the Jeopardy board

    - Lose a turn
    - Free turn
    - Bankrupt
    - Player's choice
    - Opponents' choice
    - Spin again
- Recognition of what type of sector the player's spin has landed on and the appropriate action to take. This includes:
    - Lose a turn: the player will be asked if they want to use a free spin token (if available) or else proceed to the next player.
    - Bankrupt: all of the player's earned points will be lost.
    - Free spin: one free spin token will be allocated to that player.
    - Player's choice: the player will be prompted to choose a category from the board to answer a question from.
    - Opponents' choice: other players in the game will be prompted to choose a category from the board for the player to answer a question from.
- Jeopardy board with six categories, five questions per category, and their assigned point values, increasing in increments of 200. Points are added to a player's score for correct answers or subtracted for incorrect answers.
- A timer that counts down when a question is displayed and results in losing a turn when time expires
- Two round gameplay
    - A round is deemed to be over when the total number of spins of the wheel has reached 50 or all questions on the board have been answered.
    - Each round, players begin with zero money. Points earned from the first round will be kept track of to total at the end of the game.
    - All points on the board will be doubled during the second round.
    - The winner of the game will be determined by calculating the sum of the points earned in the first and second round.

2.4 Dream

If Software Masters were not bound by scheduling limitations, there are several extra features that the team would include. This dream system implementation would include:

- High Score page - The system would save the top 10 scores to disk and display the list at the end of each game. This will encourage the users to play another round of the game to compete with previous scores.
- Sound effects - The system would play sound effects when the wheel lands on a special sector of the wheel. For example, when the player lands on the bankrupt sector, the system will play a sad sound effect. This will add more sensory information and engage the players with the game.
- Enhanced graphics - The system would animate certain graphics to create a more engaging gameplay environment. For example, when selecting an answer box from the Jeopardy board, the system would zoom into the selected box. This will make the game more exciting and keep the players interested. It will be important to have smooth animations so that the players do not focus on them. Additionally, the graphics should not overwhelm the players and cause them to be distracted from the actual game.
- Watson integration - The system would allow the players to compete against the artificial intelligence supercomputer, Watson. This will allow a single player to play the game and will provide more competition for the player(s).

3. Resources

3.1 People

- Rachel Glockenmeier - Lead CM
- Marc Julian - Lead SQA
- Rob Miceli - Lead Tester
- John Parrish - Lead Programmer
- Melisa Satili - Lead Architect
- Heather Zellner - Project Manager

3.2 Software

- Java SE 8
- Google Drive, Docs, and Hangouts
- GitHub
- Adobe Connect

4. Schedule

4.1 Milestones and Deliverables

Milestone	Deliverable	Date
Team Charter Complete	Team Charter	June 6, 2016
Project Plan Complete	Project Plan	June 20, 2016
Vision Document Complete	Vision Document	June 27, 2016
Software Requirements Specified	Requirements Document	July 5, 2016
Skeletal System Complete	N/A	July 17, 2016
Minimal System Complete	N/A	July 25, 2016
Design of Target System Complete	Design Document	July 25, 2016
Target System Complete	N/A	August 4, 2016
Project Demo Recorded	Demo of Target System	August 15, 2016

Table 2: *Schedule of Milestones and Deliverables*

The proposed schedule is deadline-driven based on when work products must be delivered but also incorporates consideration for the available resources, effort, and dependencies between tasks.

4.2 Effort Estimates

As per the syllabus, team members should be able to devote at least three hours per week to the project. This number is adjusted as outlined below to account for project meetings and rework. Meetings are currently scheduled to occur weekly on Thursdays at 8pm and last for about half an hour.

- Time available per week: 6 people x 3 hours per person = 18 person-hours
- Time allotted for weekly meetings: 6 people x 0.5 hours = 3 person-hours
- Assumed productivity rate, accounting for rework = 60%
- One scheduled week = (18-3) * 0.6 = 9 person-hours

Therefore, each calendar day in the schedule represents about 1.3 person-hours of work to be accomplished.

4.3 Life Cycle and Dependencies

The first few phases of our schedule, from initiation to design, follow a waterfall model where each is dependent on input from the work product produced in the previous phase. This approach seems best because requirements are detailed at the start of the project and are unlikely to change. Thorough understanding of the requirements before any coding begins should help us mitigate the risk of design flaws. We then transition to an iterative cycle of software design, coding each software increment, and testing. This will help us discover and fix errors as early as possible to lessen their impact on the system and schedule. Each iteration is dependent on the completion of a software increment milestone. We will also implement control tasks or umbrella processes that are ongoing from planning through execution.

Figure 2 shows the estimated schedule and duration of tasks with dependencies specified in the Predecessors column and indicated by arrows.

Software Masters

Project Plan

	Name	Duration	Start	Finish	Predecessors
1	**⊟Initiation**	**7d?**	**05/31/2016**	**06/06/2016**	
2	Determine Team Roles	2d?	05/31/2016	06/01/2016	
3	Develop Team Charter	5d?	06/02/2016	06/06/2016	2
4	Submit Team Charter	0d?	06/06/2016	06/06/2016	3
5	**⊟Planning**	**21d?**	**06/07/2016**	**06/27/2016**	
6	Develop Project Plan	14d?	06/07/2016	06/20/2016	4
7	Submit Project Plan	0d?	06/20/2016	06/20/2016	6
8	Develop Vision Document	7d?	06/21/2016	06/27/2016	7
9	Submit Vision Document	0d?	06/27/2016	06/27/2016	8
10	**⊟Requirements Analysis**	**8d?**	**06/28/2016**	**07/05/2016**	
11	Develop SRS	8d?	06/28/2016	07/05/2016	7,9
12	Submit SRS	0d?	07/05/2016	07/05/2016	11
13	**⊟Design**	**20d?**	**07/06/2016**	**07/25/2016**	
14	Develop Design Document	20d?	07/06/2016	07/25/2016	12
15	Submit Design Document	0d?	07/25/2016	07/25/2016	14
16	**⊟Execution**	**34d?**	**07/09/2016**	**08/11/2016**	
17	Build Skeletal System	7d?	07/09/2016	07/15/2016	14SS
18	Build Minimal System	9d?	07/16/2016	07/24/2016	17
19	Build Target System	9d?	07/25/2016	08/02/2016	15,18
20	Testing	25d?	07/11/2016	08/04/2016	17SS+0.67d
21	Record Demo of Target System	7d?	08/05/2016	08/11/2016	19
22	**⊟Control**	**66d?**	**06/07/2016**	**08/11/2016**	**5SS**
23	Project Management	66d?	06/07/2016	08/11/2016	
24	Risk Management	52d?	06/21/2016	08/11/2016	7
25	Quality Assurance	52d?	06/21/2016	08/11/2016	7
26	Update Project Plan	52d?	06/21/2016	08/11/2016	7
27	Revise SRS	28d?	07/06/2016	08/02/2016	12
28	**⊟Closeout**	**2d?**	**08/12/2016**	**08/13/2016**	**16**
29	Update Files	2d?	08/12/2016	08/13/2016	
30	Track Lessons Learned	2d?	08/12/2016	08/13/2016	
31	Submit Target System Demo	0d	08/13/2016	08/13/2016	21

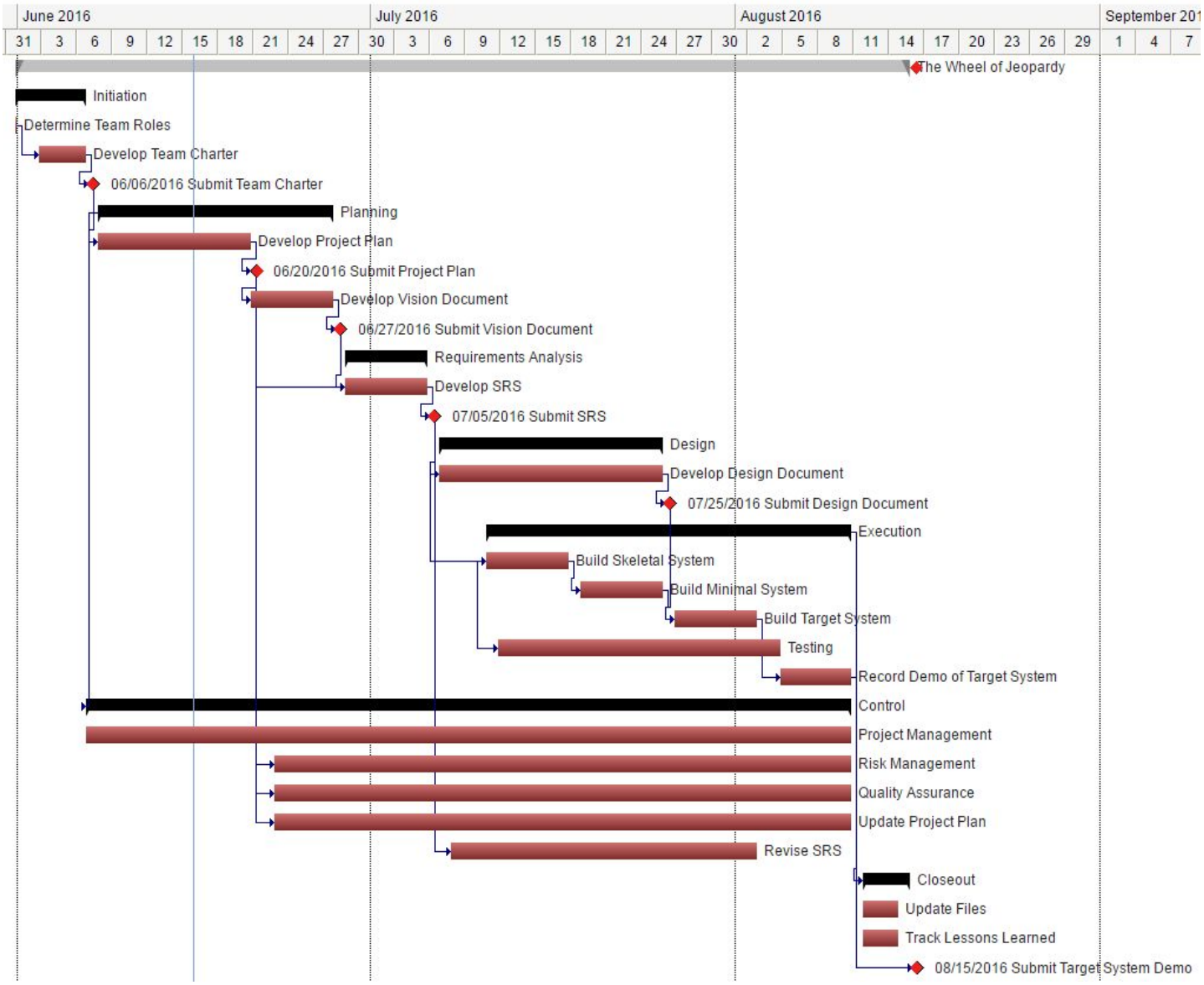

Figure 2: *Project Gantt Chart (via gantter for Google Drive)*

5. Risk Assessment Plan

Risk	Likelihood	Impact	Mitigation
Ambiguous requirements	Low	Design flaws	Discuss requirements to ensure all team members have the same understanding
Schedule delays	Moderate	Failure to deliver work products on time	Update schedule regularly, plan for rework
Conflict between team members	Moderate	Uneven labor division, schedule delays	Regular open communication among team
Inaccurate estimates	Moderate	Schedule delays	Avoid guesswork, revise project plan regularly
Changing requirements	Low	Rework, schedule delays	Good coding practice to allow for design changes
Under communication	Moderate	Misunderstanding of requirements and expectations	Weekly meetings
Program design lacks flexibility	Moderate	Changes will be difficult to make as development progresses	Code testing and review
Failure to integrate system sub-components	Moderate	Code rewrite and hacking	Planning and periodic code review

Table 3: *Risk Assessment Matrix*

6. Quality Plan

6.1 Quality Assurance

Software Masters will assess the quality of the software throughout the course of the life cycle. First the requirements of the game must established which have already been established. Throughout the planning process, the requirements must be considered and met throughout the phase. It is very important that Software Masters meet each requirement and that all requirements be functional. It is very important that the requirements be documented including any additional requirements that may arise throughout the project.

Once coding has begun, it is important that we test each functionality as it is programmed. Every functionality that is added must be documented as a configuration upgrade so that, in the event that an added functionality disrupts the entire software, the problem can be easily mitigated to the time that the configuration was made. The code will be reviewed along the way in order to maintain functionality and maintain efficiency in our software.

The implementation phase will include another review to ensure that our software meets all functional requirements and that the software meets all of the customers needs. If changes needed to be made to the software then the requirements will be re-established and the coding phase will begin again in order to incorporate the new changes.

6.2 Testing

Testing will be performed once the development begins and will continue throughout the project. It will be important to test the software incrementally so bugs will be caught earlier and more easily.

Ideally, each developer will test their code before committing it to the GitHub repository. At a minimum, the code should compile without errors before committing. The lead tester will be responsible for ensuring that the code functions appropriately, performs correctly, and remains organized. Additionally, the tester will write and perform a series of standard tests to ensure that code is properly monitored, measured, and controlled. When compilation, runtime, or logic errors are found in the code, the lead

tester will be responsible for filling an issue on GitHub to explain the error and it's source. The team member who wrote the offending code will be responsible for resolving the issue.

6.3 Configuration Management

For this project, we will be using a combination of GitHub and Google Documents. Using these pieces of software will allow the group to see the changes as other team members make them and manage as well as limit the number of conflicts between member's versions. Specifically, the code will be maintained in a GitHub repository since the team had familiarity with this service. Also by using GitHub, the software will be able to be baselined frequently to ensure that everyone is using the most up to date functional code.

Throughout the project lifecycle, Software Masters will follow a standard project management process and maintain and update all software documentation by keeping track of changes and updates arising from software requirements, schedule, risk, milestone achievements, and releases.