Table of Contents

1. **Scope**
   - 1.1. Project Objectives
   - 1.2. Life Cycle Description
   - 1.3. Features (Work Products)
   - 1.4. Deliverables List

2. **Organizational Structure**
   - 2.1. Team Structure
   - 2.2. Work Breakdown Structure

3. **Resources**
   - 3.1. Software
   - 3.2. People

4. **Quality Plan (Monitoring and Control Procedures)**
   - 4.1. Project Tracking and Control
   - 4.2. Software Quality Assurance
   - 4.3. Testing
   - 4.4. Configuration Management
   - 4.5. Project Communications

5. **Project Estimates**
   - 5.1. Size Estimates
   - 5.2. Effort Estimates
   - 5.3. Cost Estimates

6. **Schedule Estimate**

7. **Risk Assessment Plan**

8. **Assumptions and Constraints**

1. Scope

1.1. Project Objectives

The objective of this project is to produce a Trivial Pursuit computer program, which uses a square board and is Declaration of Independence themed.

1.2. Life Cycle Description

We will be using the Design-to-Schedule life cycle model, since our client has a strict 19 August 2020 deadline for project completion. This means that functionality will be prioritized as high, medium, and low requirements, and will be released in multiple increments that incorporate the highest priority features first. It will also provide regular progress to the client and management through the demonstration of each increment. However, strict timelines mean that there will be little flexibility for any potential changes in customer requirements.

1.3. Features (Work Products)

Each increment will maintain features delivered in upstream steps

- **Skeletal:**
    - Reviewed object-oriented software architecture document
    - High level description of classes, methods, and class variables
    - High level description of interfaces between classes
    - Classes outlined in code with method headers and construction methods
    - Running some scripts should invoke main methods that demonstrate some interconnectivity between classes as signified via print statements reporting which objects are created and which methods are called
    - Focus on interconnecting unfinished classes, little application logic
    - Most, if not all, classes required for the final software product should be outlined via skeletal implementation

- **Minimal:**
    - Gameplay loop where game can be very roughly played via command line interaction
    - Game can be played to completion via command line interactions where print statements describe the game state
    - “Database” for question storage is operational with enough content for playing the game

- **Target:**
    - Application features menu screens for starting the game, configuring player information, and interacting with question “database”
    - Questions can be viewed, edited, and added to
    - Front-end UI displays game state in addition to whose turn it is and supports the entire gameplay experience
    - All user requirements have been addressed
    - Application is free from bugs and all features can be used reliably

- **Dream:**
    - Gameplay can be played in full by 2-6 players/teams using a graphical interface
    - Players can provide custom game piece icons
    - Game pieces have animations and movement
    - Game can be played by multiple players over a remote connection

1.4. Deliverables List

1. Team Charter
    - N.A.

2. Project Plan
    - N.A.

3. Vision Document
    - Clear problem statement with high level description of software and features
    - Itemized list of stakeholder use cases and expected results/behavior
    - Detailed account of UI elements that will support required game functions
    - Descriptions of various user types and explanations of how they may interact differently with the software

4. Software Requirements Specification (SRS)
    - List of project/software-specific terminology and definitions
    - Subsystem descriptions outlining roles and responsibilities, how they are molded into classes, and which data structures support those classes
    - Description of each subsystem’s data encapsulation (game state, score, cards, movement, dice rolls, etc.)
    - Functional description of each subsystem discussing how they will handle assigned information. This may include high-level description of methods available for each class and their effects on instance variables in addition to return types.

- Discussion of interfaces between subsystems and classes, including:
    - Which classes will communicate with others
    - Which classes serve as storage spaces wrapping around instances of others
    - Distinguishing of private vs public methods
    - Method syntax and return types for cross subsystem information sharing

5. Skeletal System Demo

- Implementation deliverables as defined in the SRS documentation:
    - Class definitions present in scripts with methods defined but not implemented
    - Some interfaces connected where messages across subsystems can be demonstrated as being functional. This may include bare bones game play life cycle methods where print statements show which methods are being called in which classes.
    - Focus on interconnecting unfinished classes, little application logic
- Demo Plan (Walkthrough Guidelines), Demo Code, Demo Video
    - Presentation outline as rough bullet points typed up for the group to agree on content and strategy
    - Code tagged with demo version to prevent later development breaking existing functionality
    - Application for recording presentation discussed and agreed upon
    - Presenter has communicated an expected delivery date of recording in case revisions are required

6. Software Design

- Further elaboration on high-level descriptions of subsystems and functions as outlined in the SRS
- Breaking subsystems into supporting classes if they require more detail
- Detailed account of class methods and how they will support the functionality described in the SRS
- Subsystems defined in terms of classes and classes have documented attributes, functions, and interface descriptions
- Use case scenarios described in terms of classes involved and their interactions in supporting functionality

7. Minimal System Demo

- Review project status and identify potential risks
- Review previously documented architecture and discuss how implementation has manifested in working code
- Review documented requirements and identify new ones
- Revise project plan to reflect changes to milestones and deliverables
- Revisit SRS to reflect current implementation and expected
- Expected code functionality
    - Software can be interacted with via command line interface that captures the main gameplay loop

        - Main gameplay elements are supported via working code, including functionality for at least two players to be able to traverse a game-board-like data structure, be presented with an appropriate question according to the type of space occupied, provide the answer, allow the player to input if their answer was correct, and accrue wedges to eventually win the game
    - Demo Plan (Walkthrough Guidelines), Demo Code, Demo Video
        - Presentation goal: Demonstrate functional state of code to address basic user gameplay requirements
        - Presentation outlined in bullet points for group mates to agree on content and strategy
        - Code tagged with demo version to prevent later development breaking existing functionality
        - Presenter communicated an expected delivery date of recording in case revisions are required

8. Target System Demo

- Expected code functionality
    - Complete front-end UI supports playing the game to completion and previous terminal-based interactions are either removed or integrated as an option for the players
    - Menu system for starting game, configuring players, updating questions
    - All design components of SRS have been implemented and tested
- Lessons Learned document
    - Reflect on what did and did not go well, what advice we would give to someone starting this project from the beginning, how we might apply our experience to real projects?
- Demo Plan (Walkthrough Guidelines), Demo Code, Demo Video
    - Presentation goal: Demonstrate functional state of code to address basic user gameplay requirements
    - Presentation outlined in bullet points for group mates to agree on content and strategy
    - Code tagged with demo version to prevent later development breaking existing functionality
    - Presenter communicated an expected delivery date of recording in case revisions are required
- Submit peer evaluations

9. Dream System Demo (Stretch Goal)

- This section is to be flushed out upon the revisiting of the project plan during the Minimal System Demo deliverable
    - Dream features may be added to be included in Target System Demo, including potential functionality like allowing users to connect via multiple computers over a network and customize their player profiles

2. Organizational Structure

2.1. Team Structure

- **Project Manager:** Jeff
- **Lead Architect:** Mauro / Stefan
- **Lead Programmer:** Mauro / Stefan
- **Lead Tester:** Derek
- **Lead SQA (Software Quality Assurance) Engineer:** Derek
- **Lead CM (Configuration Management) Engineer:** Mauro / Stefan
- **Lead / Deliverable Editor:** Derek

2.2. Work Breakdown Structure

See Appendix A.1

3. Resources

3.1. Software

- Google Docs
- Slack
- Zoom
- Google Draw
- Microsoft Project
- Video recording software?
- PyCharm + Python
- GitHub
- Undetermined pre-existing UI framework

3.2. People

- Derek Randall
- Mauro Chavez
- Stefan Doucette
- Jeffrey Pauls

4. Quality Plan (Monitoring and Control Procedures)

4.1. Project Tracking and Control

- Developers will create development branches off the master branch to work on closely related pieces of functionality
- Developers are expected to commit their code often with descriptive commit messages so code base history can be reviewed and easily understood
- Once a piece of functionality has been implemented and is in working condition, a pull request to the master branch will be opened and reviewed by at least one other developer
- After merging a development branch to the master branch, encompassing the addition of a single piece of functionality, the development branch will be deleted
- Development branches will be named after the major piece of functionality they will deliver when merged with the master branch
- Release tags will be created so that stable builds of the software can be set aside for various demos and presentations

4.2. Software Quality Assurance

- **People:** The lead SQA will ensure that all team members document all work done in their respective roles. For software walkthroughs, the lead SQA will provide a list of Walkthrough Guidelines, which will optimize walkthrough lengths by limiting objectives, clearly communicating purpose and expectations, providing a checklist to focus the reviewers' attention, and log identified issues with necessary follow-on actions.

- **Processes:** The lead SQA will ensure that standards are established, per the project plan, and are consistently adhered to throughout the project timeline. The lead SQA will ensure that team standards adhere to IEEE and ISO recommendations, but will be tailored to project requirements. Code structure and architecture will adhere to object-oriented design principles to promote usability and readability.

- **Tools:** Lead SQA will ensure that proper CASE tools are being incorporated to promote work efficiency. Developers will use PyCharm as an IDE with integrated coding standards enforcement to ensure syntax and style.

4.3. Testing

- Since we are using the Design-to-Schedule life cycle model, each increment will include the following four phases: Detailed Design, Code, Test, Deliver
- Since the cost of correcting defects increases exponentially throughout a project timeline, defect detection will occur as soon as possible and increments will not be delivered until thorough acceptance criteria is met in order to reduce costs of correction
- For each increment, the lead tester will draft a document outlining acceptance criteria of all required functionality, including desired behaviors to include and undesired behaviors the program should avoid (positive and negative user tests)
- Tests will incorporate the full range of potential user inputs in order to detect errors and exceptions
- For each increment, completed code and the acceptance criteria document will be shared among the group for alpha testing
- All group members will document any discovered deficiencies to the lead tester, who will compile a consolidated list to provide to the lead programmer for correction
- The phases will reiterate until all desired functionality for that increment passes all tests, at which time code will be ready for delivery
- Developers will deliver code with the expectation that class-level unit tests have been written to cover code use cases more frequently than the Design-to-Schedule life cycle model

4.4. Configuration Management

- While completing project milestones, software architecture and design will be reviewed and discussed to ensure development is on track to meet user requirements
- When developers work on new features, they will incorporate unit tests to demonstrate that code works and satisfies expected use cases
- Before opening a pull request upon the completion of a new feature, developers will run all unit tests in the repo to ensure previous functionality remains operational
- Pull requests will be reviewed by other developers
    - For large pieces of functionality, this may include reviewing the development branch on his/her machine and testing functionality themselves
- Validating pull requests or creating a new software build on a merge to master will not happen automatically, since the DevOps cost of configuring such a system would take away time better spent designing and planning

4.5. Project Communications

Project communications will primarily take place in the Dream Team Slack account. The General channel will be used for generic project communications, while separate, individual channels will be used for each deliverable (documents and demo videos) and function (testing, SQA, etc.).

5. Project Estimates

5.1. Size Estimates

Point values capture the relative difference in class size, and can be roughly converted to a size measured in lines of code. We will define a single point as about 50 lines of code. Based on our rough functional components broken down into classes that appear in the finished code base the size estimates per class are:

- gameBoard: 6-7 points
    - gameBoard command line interface: +1-2 points
    - gameBoard UI interface: +3-4 points
- gamePosition: 1-2 points
- cardDeck: 1 point
- Card: 1 point
- Player: 2-3 points
- Mover: 1 point
- Dice : 2 points
- Database: 1-2 points
- databaseInterface: 2 points

The gameBoard is by far the largest piece of code as it must interface with all the following classes in addition to maintain the data structures that will manage the game state.

Based on the sum of all class estimates, we estimate that the whole project will be 21-27 points in size, which equates to roughly 1,050-1,350 lines of code.

5.2. Effort Estimates

Given the relative task sizing above, we estimate that a single point equates to roughly 2 hours of development time, which encompasses implementation, testing, and code review. These hours will be split across the developer writing the code and the reviewer of the pull request. 2 hours summarizes the total time

from starting development to having the finished and tested code merged into the master branch of our code repository.

Given the estimate that the project is 21-27 points, we forecast 42-54 hours of development time to have a working piece of software that fulfills our target system requirements. Our actual development time will be tracked to provide insight into future estimations.

5.3. Cost Estimates

We estimate that total project cost will be $0, since developers are working free of charge and plan on using free development and project tracking software in addition to open source libraries.

6. Schedule Estimate

See Appendix B.1 and B.2

7. Risk Assessment Plan

Misunderstood expectations of requirements (Uncertainty)

- Likelihood: Medium - the techniques involved in proper Software Engineering are new to the entire team
- Consequence: High - if we get critical pieces wrong, it can steer us in the wrong direction for the entire project
- Mitigate: Elevated level of communication and touchpoints. Use office hours posting when necessary.

Lack of experience (Uncertainty)

- Likelihood: High - we are a new team performing processes for the first time
- Consequence: Medium - we are likely to make mistakes that could ultimately impact our grades which is of consequence. However the product is being built in a controlled scenario.
- Mitigate: Stay up-to-date with modules and readings and constant communication. We have built a good system of checks and balances.

Team chemistry (Uncertainty)

- Likelihood: Low - teams were picked based on initial compatibility judgement and shared work ethic goals.
- Consequence: team chemistry could either increase or decrease productivity

- Mitigate: The team is composed of working professionals that understand the benefit of actively pursuing polite, respectful, and fair means to work together.

Missed Milestones or Deadlines (Knowledge)

- Likelihood: Low - we are given deadlines at the start of the course, which allows us to plan ahead for the semester. Also given the current climate, we have a lower risk of individual's traveling, going on holiday, etc.
- Consequence: High - We are on a strict schedule, and missed deadlines will cause a poor client experience (or low grades in this case)
- Mitigate: Meeting regularly in the middle of each module allows us to plan ahead and also check in and help each other out where necessary

Difficult to meet in real time - Time zones (Concerns)

- Likelihood: High - we live in 4 different time zones and 2 continents, and we all have full-time jobs, so finding time to meet is challenging
- Consequence: Low - Google Drive and Slack are great for sharing and communicating, and we have been able to execute on all milestones
- Mitigate: Slack and Google Drive allow us to work independently, even when we cannot meet

Unexpected Drop in Capacity (Concerns)

- Likelihood: Medium - we have all been working remotely and have settled into a rhythm, however there is always a chance of work or life commitments requiring more attention at any moment
- Consequence: Medium - The rest of the team could cover in case of emergency, but it would put a heavier lift on the rest of the team members
- Mitigate: We have backups in place for each of our roles and can share responsibilities across the remaining team members.

Time Constraints (Issues)

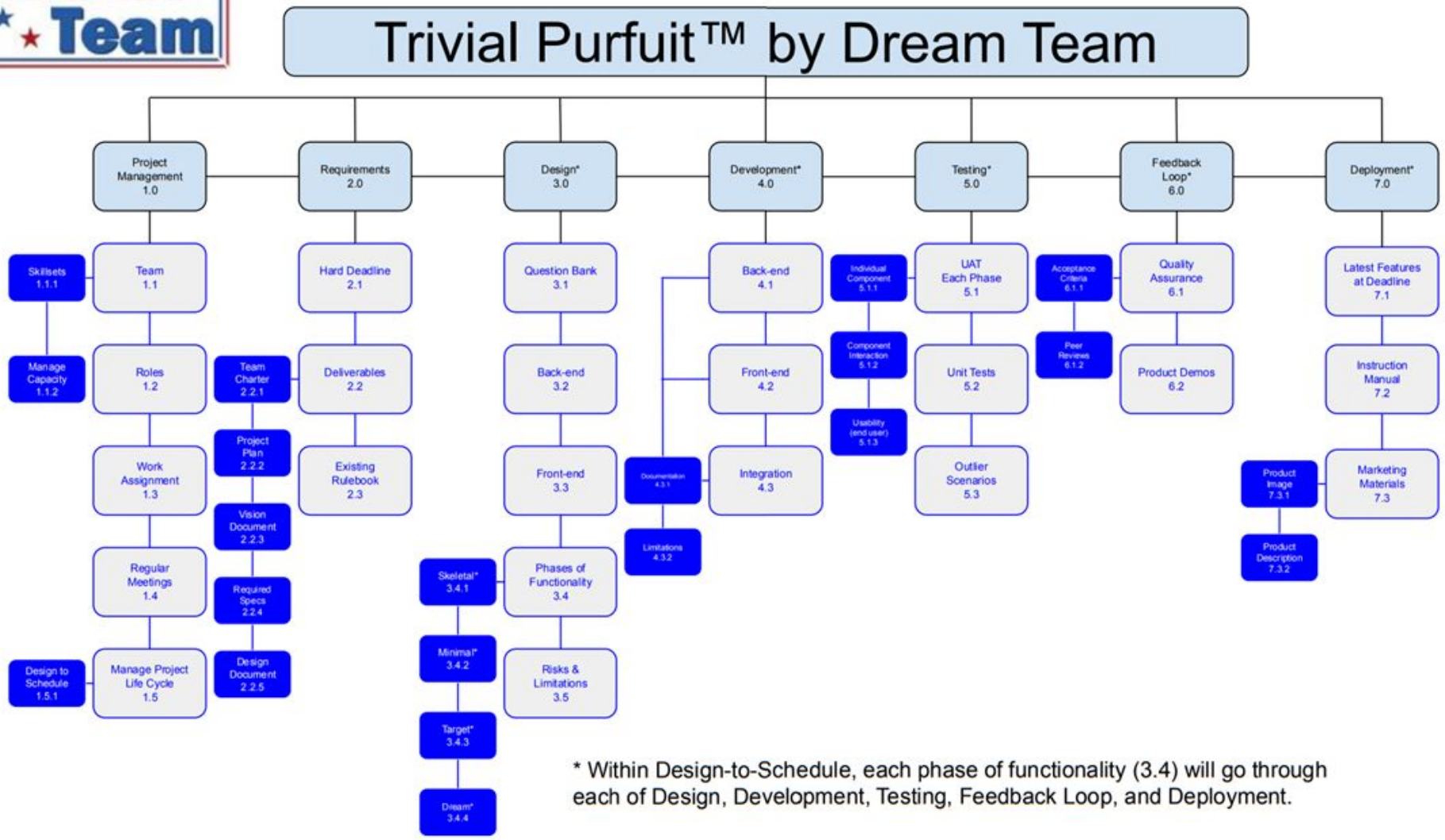
- Likelihood: High - given our schedule, we are anticipating to hit our dream scenario after the class has ended
- Consequence: Medium - client expectations will be aligned with our target scenario, however our design may not be ideal due to constraints. If our assumptions prove to be pessimistic, the identified dream scenario functionality will be integrated.
- Mitigate: Schedule to design - get something working at least, needs resolution by building plan to get something working early, then enhance

8. Assumptions and Constraints

The following are considerations that may impact scheduling estimates and development velocity:

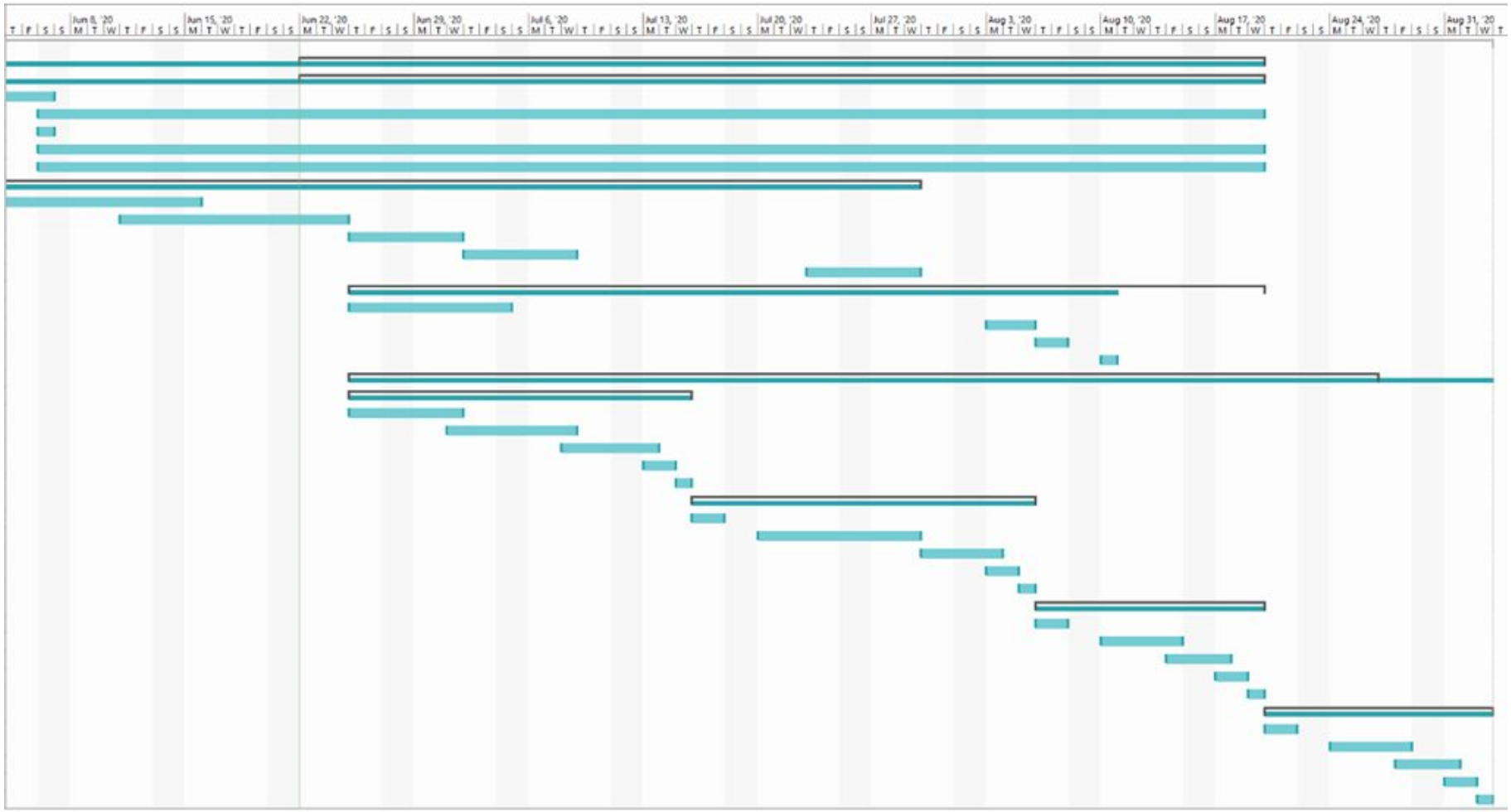
- **Holidays:** Schedule estimates assume that team members will not be working on holidays.
- **Alternate Work Schedule:** Some weeks, team members may not be able to work the full allotment of time due to primary work schedules.
- **Family Emergencies:** Some weeks, team members may have family emergencies that divert time from project work.
- **Illnesses:** Illnesses, especially given the current COVID-19 pandemic, may impact team member work schedules.
- **Software Restrictions:** The high cost of certain project management software will restrict the team to using cheaper, alternative software that may have limited functionality. This may impact project management and the development of documentation and videos.

Trivial Purfuit™ by Dream Team



* Within Design-to-Schedule, each phase of functionality (3.4) will go through each of Design, Development, Testing, Feedback Loop, and Deployment.

Task Name	Duration	Start	Finish
Trivial Purfuit™ by Dream Team	**65 days**	**Thu 6/4/20**	**Wed 9/2/20**
Project Management	**43 days**	**Mon 6/22/20**	**Wed 8/19/20**
Team	**43 days**	**Mon 6/22/20**	**Wed 8/19/20**
Understand Backgrounds/Skillsets	3 days	Thu 6/4/20	Sat 6/6/20
Manage Capacity	54 days	Sat 6/6/20	Wed 8/19/20
Roles	1 day	Sat 6/6/20	Sat 6/6/20
Work Assignment	54 days	Sat 6/6/20	Wed 8/19/20
Regular Meetings	54 days	Sat 6/6/20	Wed 8/19/20
Deliverables	**40 days**	**Thu 6/4/20**	**Wed 7/29/20**
Team Charter	8 days	Thu 6/4/20	Mon 6/15/20
Project Plan	10 days	Thu 6/11/20	Wed 6/24/20
Vision Document	5 days	Thu 6/25/20	Wed 7/1/20
Requirements Specs	5 days	Thu 7/2/20	Wed 7/8/20
Design Document	5 days	Thu 7/23/20	Wed 7/29/20
Materials	**40 days**	**Thu 6/25/20**	**Wed 8/19/20**
Question Bank	8 days	Thu 6/25/20	Sat 7/4/20
Instruction Manual	3 days	Mon 8/3/20	Wed 8/5/20
Product Description	2 days	Thu 8/6/20	Fri 8/7/20
Product Image	1 day	Mon 8/10/20	Mon 8/10/20
Phases of Functionality	**45 days**	**Thu 6/25/20**	**Wed 8/26/20**
Skeletal	**15 days**	**Thu 6/25/20**	**Wed 7/15/20**
Design	5 days	Thu 6/25/20	Wed 7/1/20
Development	6 days	Wed 7/1/20	Wed 7/8/20
Testing	4 days	Wed 7/8/20	Mon 7/13/20
Feedback Loop	2 days	Mon 7/13/20	Tue 7/14/20
Deployment	1 day	Wed 7/15/20	Wed 7/15/20
Minimal	**15 days**	**Thu 7/16/20**	**Wed 8/5/20**
Design	2 days	Thu 7/16/20	Fri 7/17/20
Development	8 days	Mon 7/20/20	Wed 7/29/20
Testing	3 days	Thu 7/30/20	Mon 8/3/20
Feedback Loop	2 days	Mon 8/3/20	Tue 8/4/20
Deployment	1 day	Wed 8/5/20	Wed 8/5/20
Target	**10 days**	**Thu 8/6/20**	**Wed 8/19/20**
Design	2 days	Thu 8/6/20	Fri 8/7/20
Development	5 days	Mon 8/10/20	Fri 8/14/20
Testing	2 days	Fri 8/14/20	Mon 8/17/20
Feedback Loop	2 days	Mon 8/17/20	Tue 8/18/20
Deployment	1 day	Wed 8/19/20	Wed 8/19/20
Dream	**10 days**	**Thu 8/20/20**	**Wed 9/2/20**
Design	2 days	Thu 8/20/20	Fri 8/21/20
Development	5 days	Mon 8/24/20	Fri 8/28/20
Testing	2 days	Fri 8/28/20	Mon 8/31/20
Feedback Loop	2 days	Mon 8/31/20	Tue 9/1/20
Deployment	1 day	Wed 9/2/20	Wed 9/2/20


Jun 8, '20
Jun 15, '20
Jun 22, '20
Jun 29, '20
Jul 6, '20
Jul 13, '20
Jul 20, '20
Jul 27, '20
Aug 3, '20
Aug 10, '20
Aug 17, '20
Aug 24, '20
Aug 31, '20