Clue-Less Project Plan

The Butler Did It

Johnathan Gilday, Michael Murawsky, Gianting Yeh

Table of Contents

Introduction

This document will serve as the initial project plan for the Clue-Less game to be implemented by The Butler Did It. The project plan addresses project milestones, schedule, high-level architecture, the team's aggregate risk assessment, configuration management, and the quality assurance plan.

Work Breakdown Structure

The Clue-Less effort may be measured in four milestones.

Skeletal System

Serves to scaffold the architecture and define the subsystems and their interactions

- Subsystems of the larger system are defined
- The interfaces between subsystems are defined
- Clue-Less objects have been defined as domain entities
- Game rules have been captured in requirements

Minimal System

Serves to provide a minimally functional Clue-Less game

- Create a minimally functional user interface. This interface is expected to be largely textual input and serves to exercise the early system design
- Capture game's rules as business logic in the domain entities and across entities in the heart of the application
- System supports up to 3 players
- System is capable of determining a winner

Target System

The goal for this effort, the target system is a multiplayer game with graphical input

- Create a graphical user interface capable of communicating with the Clue-Less server component
- Each player can access the Clue-Less game from their individual clients
- System supports up to 6 players

Dream System

If the team had more time and resources, the dream system is an option for further work

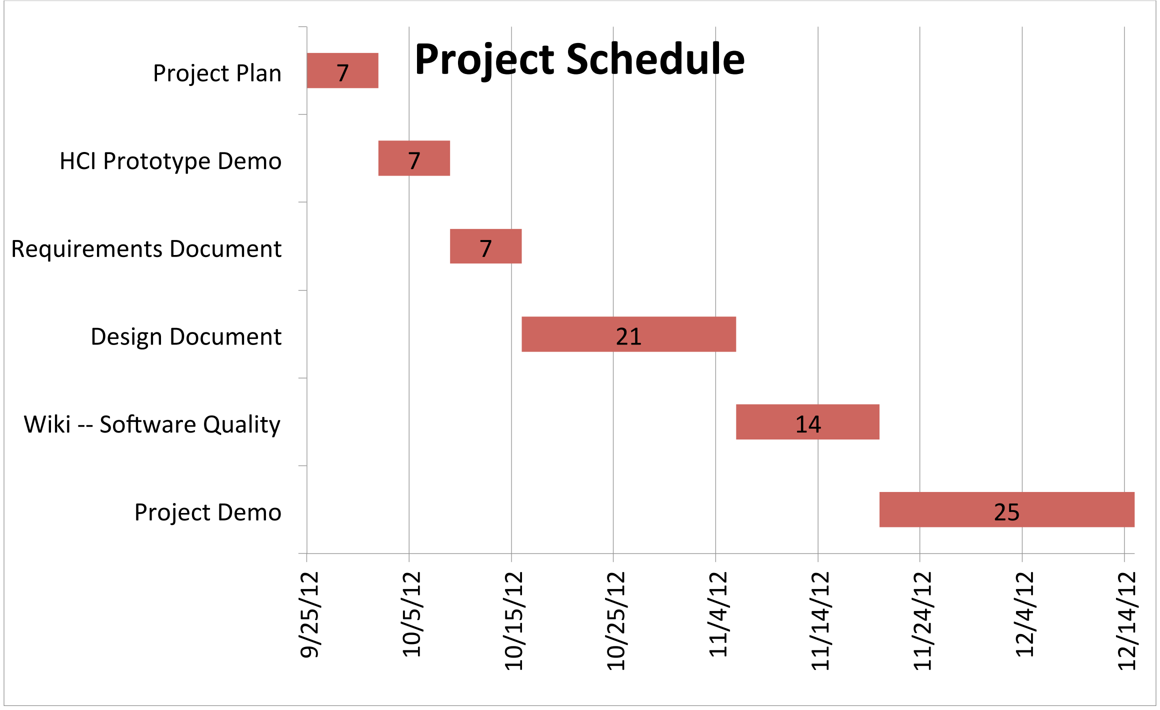
- Create a mobile version of the game for popular mobile devices
- Graphical interface is replaced by a 3D interactive interface rendered by a game engine
- Implement a scoring system for gamers to compete for the highest Clue-Less score
- Integrate the scoring system with popular social networks

Depth 1	Depth 2	Depth 3	Depth 4
Clueless!	Design	Requirements	
		Risk assessment Plan	
		Quality Plan	Quality Assurance
			Testing
			Configuration Manager
			Storyboards
		Software Architecture	Information Domain
			Functional Domain
			Interface
			Implementation Constraints
		Design Documentation	Explain Functions
			Explain Classes
		HCI Prototype	PowerPoint Slides
			Prototype may change over time.
		Plan Unit Testing	
	Build	Complete Skeletal	Validates the architecture
			Messages between subsystems are defined
		Complete Minimal	Prove out architecture
			Update the plan along the way
		Complete Target	
	Testing	Architectural testing	
		Stress testing	
		Corner case testing	

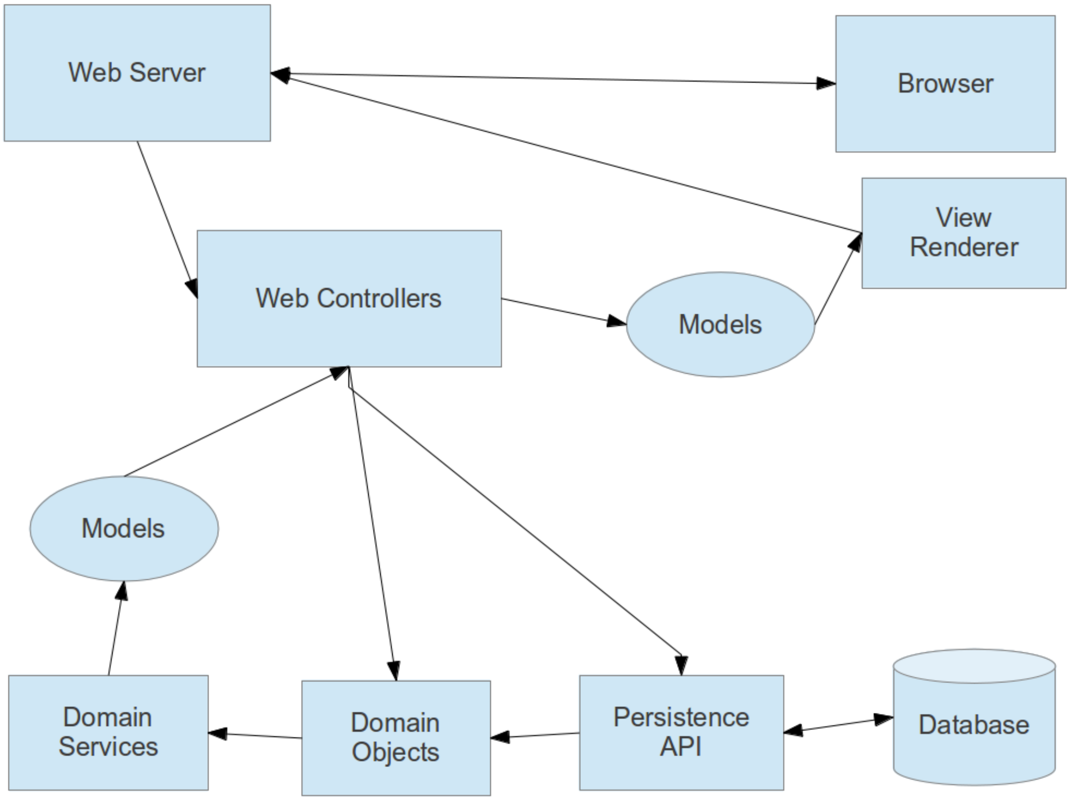
	Demo	Complete project and demo to class	

Schedule

The work breakdown structure will be laid across this semester according to the following schedule.



Architecture



Risk Assessment Plan

The Risk Assessment plan identifies risks to the system's success, noting the cause of these risks, scoring their likelihood and impact, and developing mitigation strategies. The plan is influenced by the Risk Management Guide for DoD Acquisition[i]. Likelihood was scored from 1 to 5, the lower bound being "rare" and the higher bound being "highly likely". Impact is scored from 1 to 5, the lower bound being "minimal consequence" and the higher bound being "catastrophic". Each team member provided scores along these guidelines and the following is the average.

Risk Title	Causal Factor	Likelihood	Impact	Mitigation Plan
Team collaboration	Team members are distanced from one another	5	3	Use three-way video conferencing to enable virtual face-to-face collaboration at least weekly. Use software version control to help merge branches.
Architecture fails to support product	Little time for technology exploration	2	5	Leave time for emergency redesign effort. Identify an additional, diverse technology candidate in case our first choice doesn't work out
Schedule	Team could not finish target implementation by deadline	2	5	Scope a practical amount of work for the target implementation and delegate outstanding items to the dream implementation.
No common development environment	Team members all have different configurations on development machines	4	3	Whenever possible, use Free and Open Source Software tools which are designed for cross platform use.
Computer network availability	Distanced team members rely 100% on connectivity	1	5	Frequent commits to distributed version control system in case team members go offline for any significant amount of time
Team Health	Flu season is coming	4	3	Team members can seek flu shots and maintain their health as best of possible. Finish planned work early to reduce stress.

Configuration Management

"The Butler Did It" will employ Github.com heavily to assist in configuration management. Distributed version control will help the team share source code commits to a common development thread. Git's simple branching feature allows the team to develop features in parallel efforts and merge feature branches to integration branches for testing. The team will employ a branching strategy inspired by "A successful Git branching model"[ii]. Our take on this branching strategy employs the following code branches:

1. *Master Branch*
   This branch holds the project releases. Merging on to this branch means a deliverable is ready.
2. *Development Branch*
   This is the main development thread and integration branch
3. *feature—branches*
   Feature branches are split from the `development` branch. These branches hold progress towards some feature until it is ready to be merged on to the `development` branch.
4. *hotfix—branches*
   Hotfix branches are split from the `development` branch. These branches hold commits towards bug fixes. These bug fixes are integrated back into `development` and possibly `master` and `feature-- branches`.

Quality Plan

The team will focus its code quality efforts on maintainability, loose coupling, and testability.

Consistency

The team will conform to an industry best practice code format style. In order to help the team conform to the chosen style, the project will include instructions for applying this style in popular integrated development environments so that the developer tools may assist in formatting code.

Automated Testing

Testability is important since the same characteristics that make code amenable to unit testing are the characteristics of maintainable, loosely coupled code that follows the single responsibility principle. In order to encourage quality code, "The Butler Did It" will follow Test Driven Development practices when reasonable. The automated tests will help the team verify that the application meets its requirements.

Bug Tracking

Despite our automated testing efforts, software bugs will emerge in development. When a bug surfaces, the team will use Github.com's Issues feature to document the bug, explain how to reproduce it, and associate the source code commit with the fix that closes the issue.

[i] Risk Management Guide for DoD Acquisition, 2006, http://www.dau.mil/pubs/gdbks/docs/RMG 6Ed Aug06.pdf

[ii] "A Successful Git Branching Model", http://nvie.com/posts/a-successful-git-branching-model/