Blackjack Theory Component

Problem Definition

The purpose of this project is to create a software version of the popular gambling card game, Blackjack. In addition, my project will be built using HTML, CSS and JS, allowing it to be run in a web browser, will contain functionality for computer-controlled players, customisability of the dealer and computer controlled player decisions, and multiplayer, allowing 7 people to play and chat together in the same lobby

In traditional Blackjack, players first bet, then take turn "hitting" cards, trying to get as close to 21 without going over, which constitutes a "bust", and results in the player losing their bet. Players also have the option to "split" when they have duplicate cards, and to "double" their bet at the cost of forcefully taking one card. At the end, the dealer plays, and everyone compares their hands to the dealers.

Traditionally, Blackjack could only be played in casinos, which are often predatory businesses. Even though statistically the casinos will always make money off the players, many are fooled into believing they can win, and end up in large debt. This digital version of Blackjack allows people to enjoy and experience the game without risking any money.

This digital version of Blackjack will also make the game more accessible, allowing people who to play who would not normally be able to play the game due to reasons such as not living within proximity of a casino. The software being built in a web browser also improves the accessibility, as web browsers can be used on all devices from laptops, desktops, smartphones, and can often be run on older hardware as well. Web applications also have the benefit of not having to download anything substantial, allowing users to simply type in a URL and play quickly.

Currently, most of the web-based Blackjack versions only support single player gameplay against the dealer. While Blackjack is not functionally different with one person or multiple people, allowing multiple people to chat and play together in the same lobby brings a sense of connection and community to the game.

For those who do not have people to play with, the addition of computer-controlled players will still allow them to experience the game of blackjack with multiple players. The customisability of the computer players and dealers also allows players to adjust the difficulty of the game, to make it harder or easier for them.

Needs and Objectives

Required Needs

- Functional Blackjack gameplay, including a betting and money system, options for hitting, standing, splitting, and doubling, and shuffling of the deck
- The addition of a dealer which can follow simple rules, which can be customised by the player
- The addition of at least 3 computer players, which can follow simple rules, which can be customised by the player

- Graphical User Interface allowing the player to see the game state and cards, as well as allowing the player to perform actions such as betting through button presses
- Object Oriented Design methodology is used to build the software, allowing modular development, for future addition of features and testing

Desired Objectives

- The addition of multiplayer, allowing multiple players to play the same game in the same lobby through the Internet
- A chat system and interface, allowing multiple players to chat with each other on a message box through the Internet
- Web based design, allowing users to easily access the application through a web browser
- Simplistic User Interface that looks visually pleasing and is easy to be navigated and used by players.

Feasibility Report

Technical Feasibility

Technical Feasibility involves asking whether the project is possible from a technical perspective, and whether the developers have the technical knowledge to complete the project.

This project is possible from a technical perspective, as many multiplayer web versions of Blackjack already exist on the internet, showing that it is possible. JavaScript is a high-level language capable of supporting Object-Oriented Design, meaning that it is possible for a simple card game such as Blackjack to be built using JavaScript. HTML5 canvas has the capability to draw images, shapes, and text, and update them regularly, and is commonly used in games, therefore showing that it has the capability to display the game state to players. As for the multiplayer web server, Express is a framework capable of establishing a web server and serving web pages, and Socket.IO is a library capable of providing WebSocket connection between servers and users, showing that they can host a Blackjack game server.

I, being the sole developer, can also attest that I have the technical knowledge to complete the project, as I am familiar with all the technologies used in making the Blackjack game, I have followed tutorials and guides to creating multiplayer games and card games, and I have used all these technologies in the past to create a similar software project, that being a multiplayer Space Invaders game, also made using JavaScript, Express and Socket.IO.

Operational Feasibility

Operational Feasibility refers to how well the proposed project would solve the problem and address the needs and objectives.

The proposed project will address all the needs set by the marking criteria, including Object Oriented Design, Blackjack functionality, User Interface, and customisable dealer and computer players. The project also addresses all the desired objectives, including the aesthetically pleasing design and user interface, multiplayer and chat, and a web-based design.

The project will also solve the problems outlined in the problem definition, that being providing accessibility of Blackjack to more people, providing a sense of connection through multiplayer and chat, allowing solitary users to play with computer players, and allowing users to change their difficulty by changing the behaviour of the dealers and computer players.

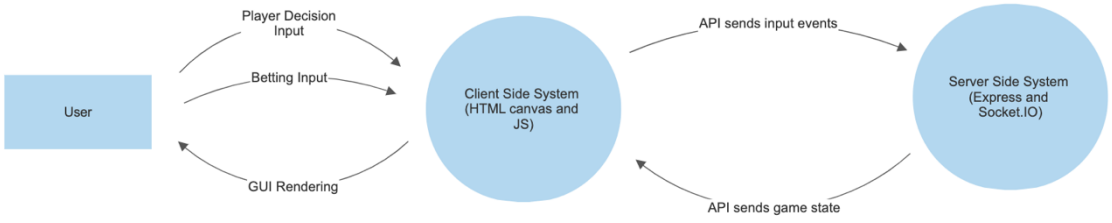
Scheduling Feasibility
Scheduling Feasibility determines whether the project will be finished by a reasonable deadline.

For this project, we were allocated roughly 7 weeks to complete the project. Seeing as I had completed many other software projects of similar difficulty and scale in similar time frames, and that there were no new technologies that I needed to learn, I judged that I could complete the project within the time frame.

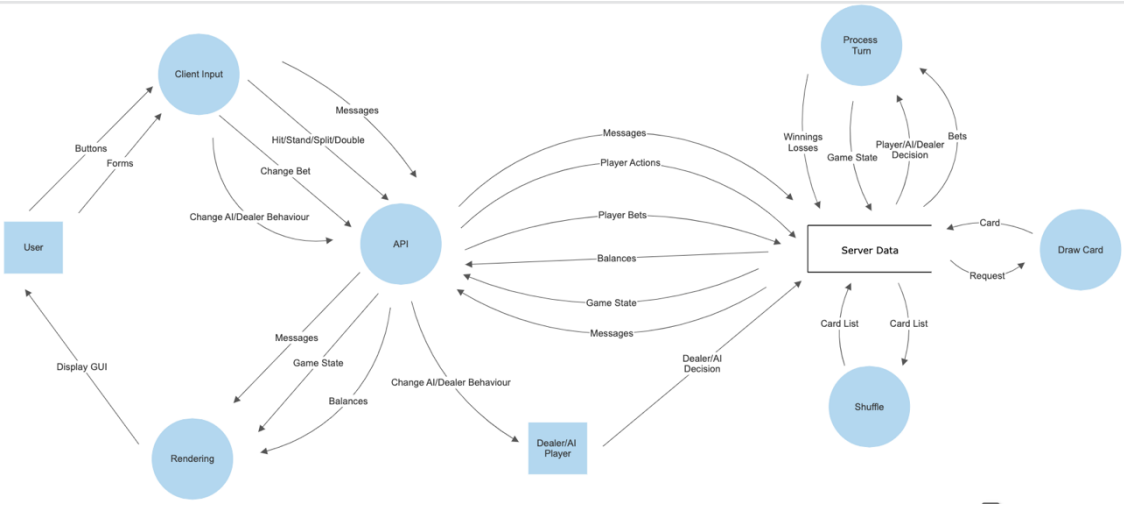
Financial Feasibility
Financial Feasibility analyses the costs, investment, earnings, and profits that are involved in a project. Financial Feasibility then determines whether the project will make a net profit and will be financially beneficial for the developers.

To analyse the Financial Feasibility, I will have to analyse the costs involved in the project, and the potential earnings. Since I am placing this project on the Internet for free, with no advertising, there is no projected revenue to this project, which is expected, as this was a simple school project. Since I am developing this myself, there are no development costs involved in the making of this project. Normally a web server and domain name would cost money to host and maintain. However I am using a service called glitch.com, which provides web hosting and domain names for free users with some limits. Therefore my project will have no costs and no revenue, and so while the project is not technically financially feasible, this was expected for a hobby project done for school.

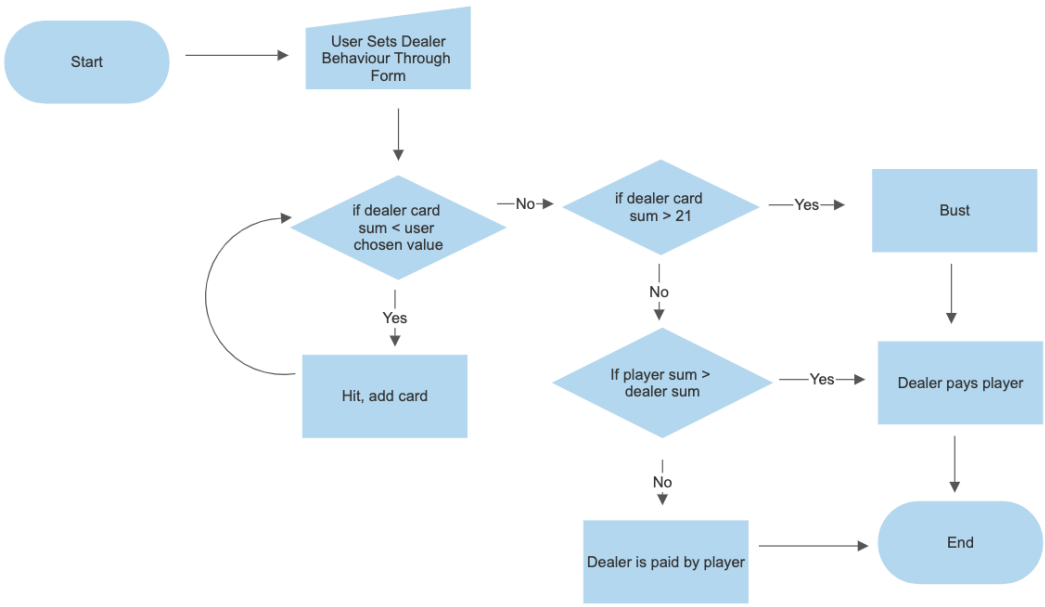
Data Flow Diagram
Level 0 DFD (AKA Context Diagram)



Level 1 DFD



System Flowchart for Dealer Behaviour



Build Log

22/10/2021

Began work on the project. Implemented cards and decks using Object Oriented Design methodologies, with a card representing an object with 2 attributes of rank (Ace, King, 5) and Suit (clubs, diamonds, hearts, spades), and a deck representing an object containing a list of cards and methods to shuffle and draw cards.

```
class Card {
    constructor(suit, rank) {
        this.suit = suit;
        this.rank = rank;
    }
}
```

Card class

```
shuffle() {
    for (let i = 0; i < this.deckList.length; i++) {
        let randomIndex = Math.floor(Math.random() * this.deckList.length);
        let swapA = this.deckList[i];
        let swapB = this.deckList[randomIndex];
        this.deckList[i] = swapB;
        this.deckList[randomIndex] = swapA;
    }
}
pickCard() {
    if (this.deckList.length > 0) {
        return this.deckList.pop();
    }
}
```

Methods for shuffling and picking cards from deck

29/10/2021

Began work on the server side code. Implemented the express web server, which sends HTML files when users request them, along with other features. Implemented socket.IO events, allowing users to emit events to the server and vice versa. Tested socket.IO with a basic chat demonstration, where users can send chat messages to each other.

```
const express = require("express");
const app = express();

const port = 5000;

app.use(express.static("public"));

var server = app.listen(port, function() {
    console.log('Example app listening at http://localhost:${port}')
})

app.get("/", function (request, response) {
    response.sendFile(__dirname + "/public/app/index.html"); //when user requests
webpage, send it
});
```

Express server, which listens to port 5000, and sends the webpage when requested.

7/11/2021

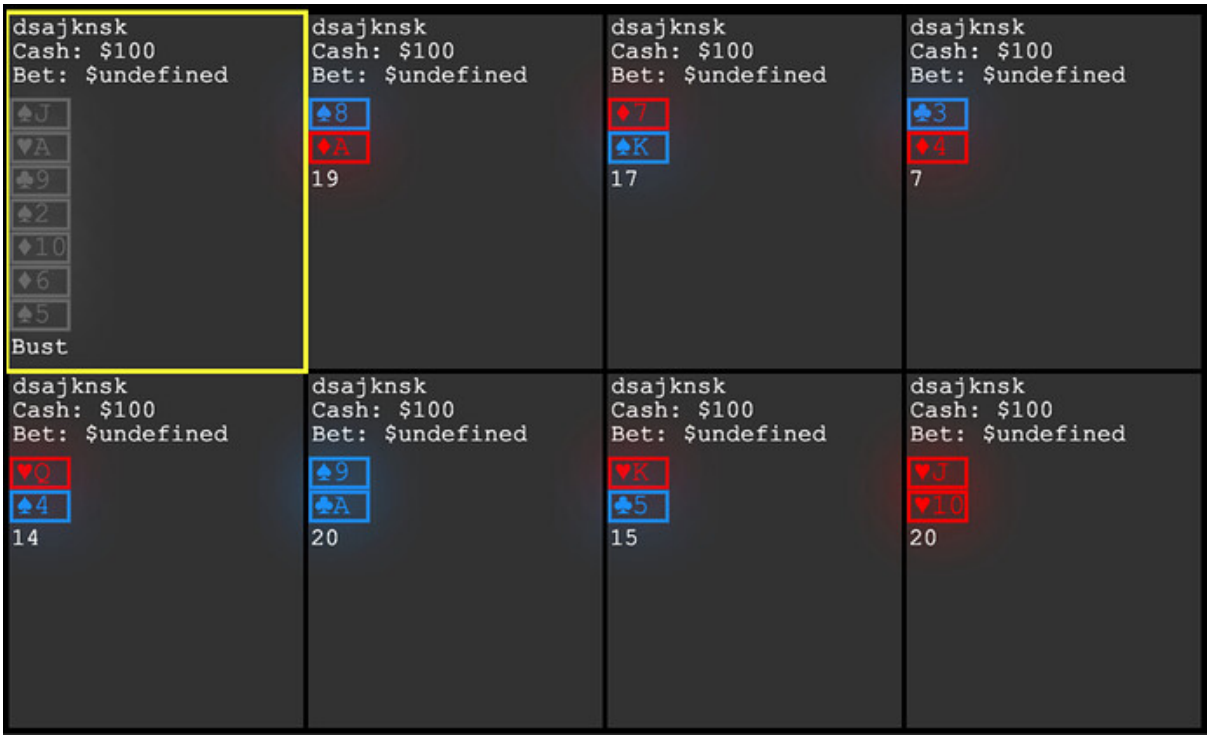
Downloaded some card sprites to use for prototyping purposes. These card sprites will be the stand in until I decide how the GUI should look and I make more visually appealing cards.

Example Card

I have implemented a basic GUI where cards are shown to the player using HTML canvas

15/11/2021
I have implemented the core functionality of the Blackjack game, including the betting system. The player has access to buttons which allow them to choose between hitting, standing or splitting on their turn. The player is also now implemented using Object Oriented methodologies, with methods for hitting, finding the sum of a hand, and attributes containing player name, id, money, and a list of different hands.

2/12/2021



The GUI has been finalised, with simplistic, neon cards against a dark background allowing for a visually pleasing experience. Split hands are shown side by side, and cards in the same hand cascade downwards. There have been some difficulties and limitations, as the screen size only allows for 7 cards to be displayed in each hand, which will have to be a limit hard coded into the game.

3/12/2021

The dealer and computer players have been implemented as child classes of the player class with simple behaviours that can be changed by the player. The multiplayer aspect is now finished, and the game is hosted on https://multiplayer-blackjack.glitch.me/. This presented some challenges, as I had to migrate the files from my own laptop file system, to the cloud through a GitHub Repository, and then finally to the glitch.com project. After some light playtesting with friends, many bugs have been rooted out, including miscounting of hands containing aces, buttons not working, bots not making moves and players not being able to join.

Evaluation of Implementation Methods

When a new piece of software is developed, users must adjust and learn to use the new software, and existing software must also be able to coexist with the new software. Different methods of implementation have different advantages and disadvantages that suit separate scenarios, and so choosing the right one is important for a software project.

Direct Cut-Over

Direct Cut-Over refers to a complete and immediate overhaul of the old system, and replacement with the new system. The old system instantly becomes unavailable and the new system can be used instantly. Anyone undergoing such a swift transition must ensure that all their users are ready for the switch, and know how to use the new software, as otherwise a jarring, unknown change in software may cause issues for users who do not know how to use the new software, and may cause an unpleasant user experience. The development team must also make certain that the new software has been tested extensively, and has no bugs and no issues, as if anything goes wrong, the old software can not be used as a fallback option.

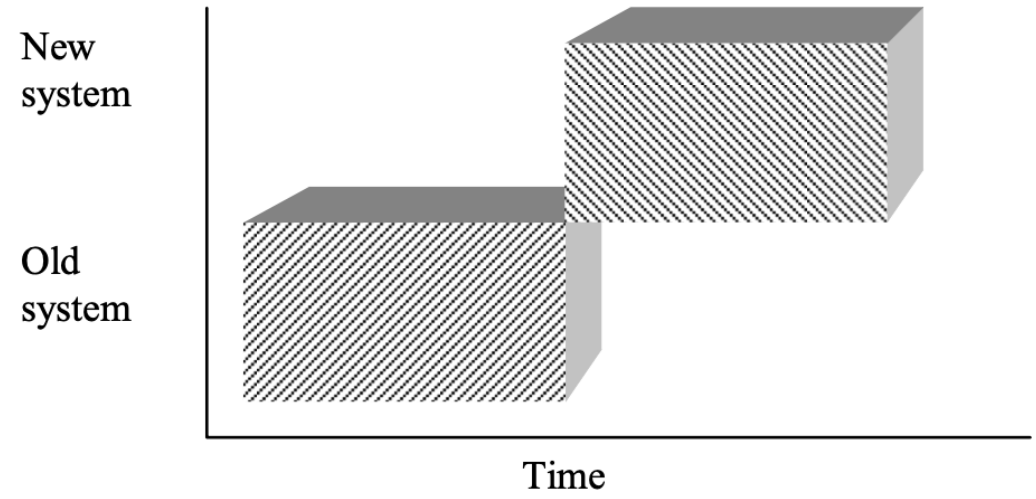


Fig 2.15
The Direct Cut-Over method of installation.

The advantages of direct cut-over is that it is simple, cheap and easy to implement as all that is required is a single change in software. The disadvantages are that it requires all users be fully trained in the new system, otherwise user experience may be damaged. It also has the risk of system-wide failure, and may need a large amount of initial updates and fixes if the changed software has not been extensively tested.

An example of direct cut-over might be a public transport system informing all passengers months in advance of a switch from paper tickets to electronic tickets, and then immediately switching all hardware and software over to the new system. This approach is suitable as it has been thoroughly tested, users have been informed of the change, and it would be impractical to run both pieces of hardware at the same time.

Parallel

Parallel implementation refers to operating both the new and old systems while the transition takes place, allowing users to be familiarised with the new system while still having the old system as a fallback option. Once users are familiar with the new system, the old system is no longer supported.

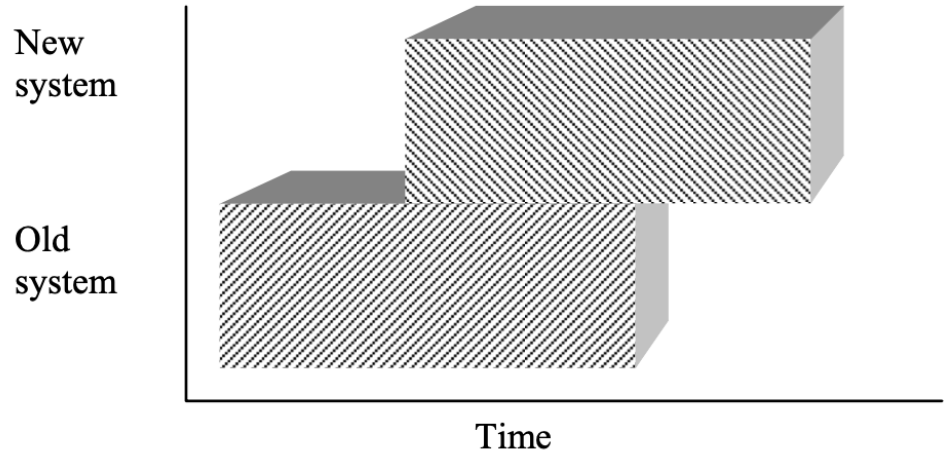


Fig 2.16
The Parallel method of installation.

The advantages of the parallel method is that users can more easily transition from the old to the new system, improving user experience. It also provides a safety net, as if there are any major bugs or issues, the old system can be used as a fallback while the issues are fixed. The disadvantages are that this method requires double the workload for both users and developers, as users have to perform functions on both the old and new system, and developers have to maintain support and functionality for the old and new system.

An example of the parallel method might be a game engine software, which wants to add functionality and update their software. They may allow users to download the new version, while maintaining the old version if they wish, but informing that the old version will no longer be supported in a few months, therefore giving users ample time to switch over to the new software and be familiar with the changes.

Phased
The phased method is similar to the parallel method, however with the phased method, instead of having the full functionality of the new and old system available at once, the old system slowly loses functionality while the new system slowly gains new parts and methods which replace the functionality of the old system

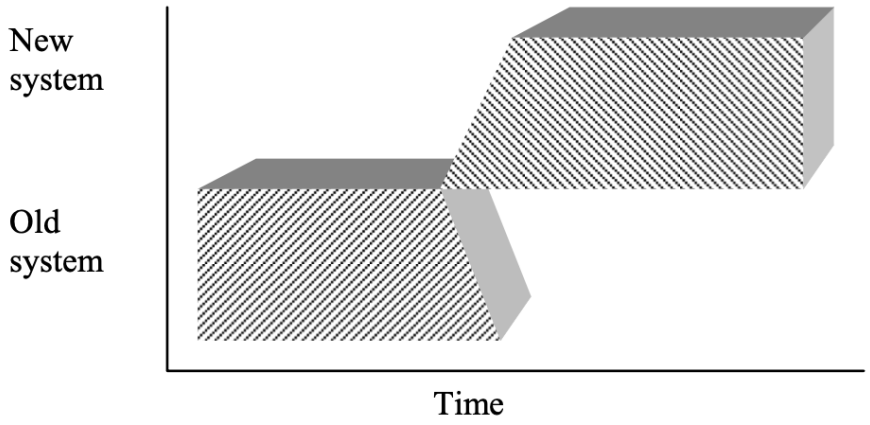


Fig 2.17
The Phased method of installation.

The advantages of the phased method are that it allows conversion to occur while modules are being built, allowing users to transition much faster than if they had to wait for all the modules to be released before converting. It also allows for a smoother transition, as businesses can slowly add individual features, working their way up to a complete transition. The disadvantages are that while each change is less drastic, each individual change can still cause bugs and issues, and since functionality of the old system is also being taken away, there is less of a fallback system.

An example of the phased method would be a company, slowly replacing functions from their API. Slowly they will replace some old API functions with new API functions, thereby allowing a gradual transition for their users, with changes rolling out as soon as they are developed.

Pilot
In the pilot method, a small, test group of users are selected to install the new system. The group will use the new system, test it for issues, and send evaluation back to the developers who can fix issues and adjust it for user feedback. Then once the software has been thoroughly evaluated and tested, the developers roll out the changes to everyone.

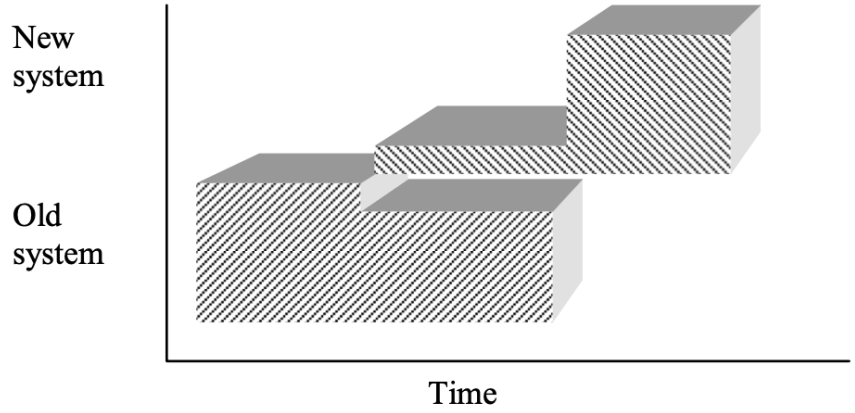


Fig 2.18
The Pilot method of installation.

The advantage of the pilot method is that it allows a larger group of users to act as testers, more efficiently finding bugs and issues. It also acts as a practical test of how the software performs in a real scenario, therefore finding issues that the developers may not have been able to imagine in the real world scenario. The disadvantage is that the old system is no longer there, so there is the small risk of having no fallback option, however this risk is minimal, as the product has already been thoroughly tested by the test group.

An example of the pilot method might be a school providing access to new assessment and timetabling software for a certain amount of students, who are then asked to report bugs and provide feedback, in order to create a more stable, bug free version for the rest of the school to use.

Recommendation
For the project, there are no users of previous software, and there is no existing software that needs to coexist with new software, as this project is being built from scratch. Therefore I think for a small simplistic project such as this, I can go with direct cut-over, as the project has been thoroughly tested for bugs and issues, and the users would be unfamiliar with how to use the product either way, so a drop in user experience is non-existent.