**Software Qualities**

**Team A**

**Performance:**

Performance is crucial in the Drone Wars game. In order for the user to have a good experience playing the game, server performance is a must. The server must successfully relay multiple 720p video streams to Android smartphone devices. If there is significant latency in the video transmission then the game will become unplayable. Likewise, if the client cannot dispatch commands to the server, which will then pass them to their respective drones, in a reasonable amount of time then the game will not function properly. Performance is also important in the image detection process of our application. When the user presses the fire button the Android application has to detect whether a target was actually in the target-area. If there was a delay of say 10 seconds before the image detection was actually started then it is possible that the targeted drone is no longer in the target-area and false-negative, determining that the enemy drone was not hit, would be produced.

To obtain the desired performance that we need in our application and server we implement critical parts in low-level high performance languages. Our server is done in nodejs using socketIO. SocketIO provides very low level port access which allows for quick transmission speeds. The low level nature of socketIO reduces the costs of latency produced by an equivalent high-level language. Similarly, the video processing is performed using a C library so that there are smaller overhead costs compared to using a java equivalent library.

(Dake) Our app involves target detection via camera and we use our mobile device to process the target detection. In order to get a fast speed in target detection, instead of analyze the video stream, we choose to enable our app to take a screenshot and then process a single frame of picture, which could save a lot of time and CPU resource.

**Maintainability:**

Maintainability has been pretty important in Drone Wars. Drone Wars needs to be maintainable since changes to the server and application occur very quickly. Without maintainability these constants changes would not be manageable.

To obtain the maintainability we implemented our game using the Model View Controller software design. This design separates the different aspects of the game into three different parts: drones, server, and mobile application. This design allows for separation of concerns. The drones are in the Model part of the design and simply provide data to the user and receive input from the user. The drones have no logic associated to them. The server is the Controller part of the design. This is where the majority of the logic occurs. The server controls the game and links the Model and View parts of the design. Finally, the phones and our Android application make up the View part. This part is used to only visualize the data provided by the Model. This design separates the concerns and makes them almost completely independent of each other. Since they are independent of each other they are all extremely maintainable. Changing code on the server did not affect anything in the Android application and vice versa.

In order for the code itself to be maintainable we used a strict coding convention. We made sure to break the code up into packages. This made the code very modular and easily maintained. This concept is similar to the Model View Controller. The packages allowed up break the code into pieces which were related to other pieces in the package but unrelated to another package. If work needed to be done in one package, we had the security of knowing that another package would be untouched. Included in this strict convention was the use of meaningful variables and methods. We made sure to name the variables and methods such that their names explained their meanings and what they were used for. This made the code more readable and maintainable.

**Portability**

For a software system built on a mobile platform such as Android, portability is one of the most important aspects pertaining to our project. As this is a multi-part project, the decision was made to write the application in both Android and Node Java Script (JS). The decision to use Android is based upon the java runtime environment and its ability to run inside of a virtual machine. This means that the application is hardware independent and does not have to be compiled for every type of device configuration on the market. All the user has to do is go into the play store and install the application on their device running Android KitKat or newer.

The server component is written in Node JS for the exact reason of portability. Node JS is an open-source, cross-platform runtime environment for server side and networking applications. It is extremely lightweight and modular in design. This aspect of Node JS was very appealing when the design choice was made. Like Android, Node JS has the ability to run in a virtual machine independent of any hardware configurations. We designed the server to be contained and run on a single Raspberry Pi computer. The fact that both the server and client side are both written in software that run in a virtual environment guarantees portability. Google’s Android Operating System is found on many consumer products and Linux distributions are free to install on any machine. This allows for project to maintain portability and keep costs of hardware down. There is no need for special hardware outside of the drone.

**Repairability**

The system was designed around repairability. This was accomplished by having a completely modular design. Each component of the project is independent of the other. The android application can run without the server and the server can run without the android app. This allows the components to be fixed independently. If we want to work on the android application you don’t have to have a raspberry pi running the server to do so. If we need to work on the server, we don’t have to have an android application to run it. This is an integral part of the repairability of the project. Another aspect of the project which attributes to a good level of repairability is the languages to which the project components are written in. The android application is obviously written in Android which is a java based language. Java is a hardware independent language and runs in its own virtual machine.This allows the team to run the app on any android device which is running Android Kitkat or newer. Not to mention that Java is one of the most well-known languages in the mobile world. These properties of our project ensure that if there is in fact an issue with the product, the troubleshooting and risk of changes are minimal given that it is modular, hardware independent, and written in a well-known object oriented language.

**User Friendliness**

The Drone Wars application is designed with user friendliness in mind. Our server software is integrated with the DWA application to make our users’ experience as smooth and seamless as possible. The DWA game server broadcasts its own WI-FI network for the players’ phones and drones to connect with. Once a player opens the play screen they are greeted with a drone selection screen. This is where they select their own drone from a list of those connected on the server. Since the game UI was designed with user friendliness as our top priority, when a drone is selected the DWA UI automatically transitions to the next activity and the drones’ LED lights flash to visually confirm to the user that a connection was established. As the user sets up the game the phone sends the server commands to prevent others from picking the same drone and colors.

Our application limits the user input needed to set up and begin the game. This design was chosen because it’s easy to understand and eliminates confusion for the users. Once the game commences there are easy to identify symbols assigned to ammunition, lives, and joysticks. Easily recognizable symbols help reduce the user's learning curve. To further reduce the learning curve we designed instruction screens to help users better understand the game mechanics.

**Understandability**

The success of a good game is for it to be challenging and enjoyable while requiring little effort by the user to learn how to play. A game’s success can hinge on how easy or hard it is to learn to play the game; if it is over complicated, players will not enjoy the game and will most likely not continue to play. The DWA application and server was designed to minimize user input and automatically set up most of the core aspects of the game. Each screen the users see has clear and precise instructions for the players. Once a user’s input is detected from the DWA application it is sent to the DWA game server where, the server will set up the game with the correct settings automatically. The game can be started by most users with just two clicks, one to select your drone and another to choose your drone’s color. The game server will choose the first player that connected to their drone to be the admin for the game. The admin user is presented with a screen that allows them to set up the game’s time limit and number of lives each player is started with. While the admin is setting up the game the rest of the users are sent to a waiting screen.

Once the admin user is finished the players in the waiting screen will automatically transition to the play screen. The play screen was designed to be easy to understand with readily identifiable symbols such as two joysticks to control the drone, a heart icon to symbolize lives remaining and a bullet icon used to symbolize ammo remaining. The DWA team designed the application to be as minimal as possible for two reasons: one, so that the user will not have too many distractions on the screen while trying to fly the physical drone and two, to make it as easy to understand as possible. If the user ever has any questions, the DWA application was made with an instruction screen. The instruction screen reviews what each symbol means and how to connect to the drone and game-server, and how to control the drone. With the instruction screens and the readily identifiable icons used in the play screens, the user should have all the crucial information needed to play Drone Wars without any difficulty.

**Reliability**

The reliability of Drone Wars will be entirely dependent on the server which is used to manage the whole gameplay experience. It does everything from managing all connected phones to controlling all the connected AR.Drones. To ensure reliability the server software follows design patterns that simplify the design while decreasing complexity. The server avoids software faults by utilizing simple thoroughly tested functions to communicate with attached devices. The data transmission size is kept at the minimum in order to allow more devices to be connected at the same time. The Drone Wars server and application have been tested with four AR.Drones and four Android devices connected at the same time. With all the connected devices there was no communication delay recorded. This means that maximum player could be increased in the future since the communication medium is not saturated and the server can keep up with processing all requests and relaying data to appropriate devices.

Reliability is also increased by utilizing an event driven architecture which makes the system very responsive and able to handle unpredictable events that would otherwise crash the system.

**Evolvability**

In order to improve usability and ensure long term enjoyment and prolonged life of Drone Wars it is very important to make the software very evolvable. Many game systems rely on the developers to implement very specific content such as gameplay modes. The problem with implementing specific gameplay modes is that the users end up getting bored due to limited number of modes available and then the developer is required to update the system to create additional modes. Drone Wars goes about this a bit differently, instead of creating specific modes the user is given the ability to modify many gameplay parameters such as game time, starting number of lives in order to create a much customized gameplay experience. This allows the end user to come up with numerous unique gameplay experiences.

With never iterations of the software additional parameters will be added to create many more gameplay possibilities. This approach will keep the end user engaged for a longer amount of thie thus prolonging the life of Drone Wars.

Beyond creating additional gameplay parameters the software architecture will follow an event driven architecture. This allows the whole system to be loosely coupled which facilitates more responsiveness and normalizes the system to handle unpredictable and asynchronous events.

**Productivity**

Over the course of development of our application, we have been using UConn’s GitHub to govern our code. We were aiming for Android application for our first product. As of now, we are developing DWA-v2.5. The 1.0 version of our application was designed and built from ground up during the design phase. For the 1.0 version of our application, we were using eclipse to write our code. We divided it into three major parts: user interface, logics in play mode, and the instruction. During the winter, we moved our application to Android Studio and developed the 2.0 version. Because the new features we added into the application requires to be using Android Studio. Our application is a multi-players game, so it requires a server to govern all the players’ app and their drones. Then our application comes to v2.5. We developed the server that can pick up the Wi-Fi signals broadcasted by the drones. The server is also available on UConn’s GitHub. Since the design phase of our product, all of our team members meet regularly at least once a week to keep the productivity.

Our product is a multi-player game; it needs a piece of hardware to run the server. Each player needs to have a parrot AR drone 2.0. The game needs a designated area of playing space.

**Reusability**

We made our codes reusable and easy to implement so that if we want to add a feature, we can use the functions we already implemented. For one reason, reuse the functions we already implemented would not increase the size of our application too dramatically. Another reason is, if we want to update or upgrade the functions that implement reusable functions, we can just update or upgrade the reusable functions. To have reusable codes makes updates much easier. In the early stage of v2.0, some of the new features were developed in Android Studio, whereas our old codes were written in Eclipse. This is a perfect example that shows our code is reusable. As we came to v2.5, we have to merge our old code in Eclipse and the newly developed features in Android Studio. Since we made our code reusable, it was very easy to merge and integrate them into a more advanced version.

**Robustness**

One of the characteristics of Drone War game is that it requires real-time strategy, which means the participants position and maneuver their drones and structures under their control to secure areas of the map and/or destroy their opponents' drone. To ensure this key point our app and server should be able to transmit the video feed and data process quickly and stably.

To make sure the robustness, we will need a server that can broadcast Wi-Fi signal in a stable way. We use the newest version of Raspberry Pi B with a Wi-Fi dangle such that it can broadcast Wi-Fi within a certain area. In order to allow user to fully enjoy the game and meanwhile keep the fluency of the game. We must test the furthest distance within which the drone can connect to the server’s Wi-Fi with a decent manner. By using stress testing in hardware we are able to know the furthest distance with the server’s Wi-Fi capability.

Another to the server and Wi-Fi robustness is the loss of control, namely it could be the user side app crash, or the drone is disconnected the Wi-Fi. The first scenario could come into being from a phone call or user’s interruption in the middle of the game. To solve this problematic scenario, the server stops data transmission to the user and send an auto landing commend to drone and enable the drone to land safely. For the rest two, the drone has its own emergency mechanism and will automatically land when it find the Wi-Fi is lost.

Meanwhile, a threat to software robustness may lay in the database. To eliminate this threat, we decide to keep our database schema as simple as possible since it will only be used to store some basic statics of the user and the simplicity can prevent instability and uncertainty that caused by complexity.

In our app we also use the open source libraries such as OpenCV and some library provided by Parrot. To ensure the robustness of the third party libraries, we will also use the newest version of the libraries and use the documents provided the authors to detect the potential bugs and we will also be aware of the new bugs or flaws report by other users.

**Interoperability**

To allow the user to be able to play with others in most places our server deployment has a good interoperability. Because our code can be deployed on any machine that can broadcast Wi-Fi signal. Our server does not require some specific hardware or software, so even a tiny computer as raspberry Pi can work well. Our database is also designed to be simple so it should has a good interoperability.

On the other hand, our App use the ARDrone SDK, which is specifically develop for AR Drone 2.0 so the player using other versions of drone may not be able to use our app.