HUDroid
Heads-Up Display on Android

Technical Report

Team Members
Colin Bennett
Kenton Christian
Avezou Petit Frere
Samuel Owen
Robert Turcotte

Harris RF Sponsors
Nathan Ransom
Timothy Lee

Faculty Coach
Ryan Schneider
Table of Contents

1. Project Overview
   1.1 Goals
   1.2 Scope
   1.3 Sponsor
2. Basic Requirements
   2.1 Functional Requirements
      2.1.1 HUD Requirements
      2.1.2 Widget Requirements
      2.1.3 System and Preferences Requirements
   2.2 Non-Functional Requirements
3. Constraints
4. Development Process
5. Project Schedule: Planned and Actual
   5.1 Inception Phase
   5.2 Elaboration
   5.3 Construction Phase
   5.4 Transition Phase
6. System Design
   6.1 MVC Architecture
      6.1.1 View Domain
         6.1.1.1 Four Main View Layers
      6.1.2 Model Domain
      6.1.3 Controller Domain
   6.2 Subsystems
      6.2.1 Overall Class Diagram
      6.2.2 AR Subsystem
      6.2.3 Widget Subsystem
      6.2.4 Preferences Activity
   6.3 Threading
7. Process and Product Metrics
8. Product State at Time of Delivery
9. Project Reflection
10. Glossary of Terms
11. References
1. Project Overview

1.1 Goals

The goal of the HUDroid project is to create a heads-up display (HUD) app on Android. Similar to in video games, where objectives are marked by icons on the screen, HUDroid displays targets on the screen as icons over a video feed from the camera. As the phone is moved around, the targets on-screen move appropriately to remain pointing to the proper locations. When the phone is turned away from a target, the target is represented by an arrow on the side of the screen, which points in the direction of the target. The target icons can convey additional information about the targets they represent, such as the type of target using the shape of the icon (car for ground-based targets, airplane for air-based targets, ship for sea-based targets, etc.), and the faction of the target using the color of the icon (green for friendly, red for enemy, or grey for neutral). Target icons also have the distance to the target displayed above each icon, as well as the name of the target below each icon.

The targets themselves are representations of devices. In order for each of these devices to be displayed, they must be running an Android app called SAndroid, which was created by a previous senior project team. SAndroid stores and receives updates for all information regarding its registered devices. HUDroid reads this information from SAndroid and appropriately displays each target device on-screen. This core functionality of the HUDroid app is often referred to as the augmented reality portion of the project, though this app does not at all interpret data from the camera.

In addition to the augmented reality, HUDroid also displays various widgets on-screen. A widget is simply any small tool or gadget which displays on-screen over the camera and targets. For example, SAndroid provides a compass and tactical chat widgets. These and other widgets can be added, removed, or moved around the screen through the use of the Android interface and touch-screen associated gestures.

The final goal of the project is to provide a method for users to change their preferences regarding the app. Through the use of the Android menu button, users can access controls to change the display of target icons, as well as information which determines the specs of the camera. These preferences are saved so that user-specific preferences will persist when the app is closed and reopened. HUDroid also provides a mechanism for users to restore the default preferences in case the user makes a mistake and cannot figure out how to undo their changes.

1.2 Scope

The scope of this project is strictly within the HUDroid app. More specifically, all of the augmented reality functionality will be implemented from scratch - the team was unable to find any existing augmented reality libraries which were applicable to the project. It is not within the scope of this project to make any changes to SAndroid, including fixing any existing defects, nor to provide additional widgets for the user to choose from. In the rare occurrence that there is an unavoidable defect SAndroid which cannot be overlooked, the project sponsors have given permission to push back the schedule and remove low priority requirements in order to fix the outstanding defect.

In the regards to widgets, HUDroid only provides the default compass and TacChat widgets.
supported by SAndroid. Since SAndroid’s compass and TacChat apps are not directly portable to HUDroid for use as widgets, it is within the scope of this project to create pseudo-widgets which display the compass and TacChat widgets in an acceptable form using SAndroid’s code. Additional widgets may be added by future development teams with access to the HUDroid code, and it is within the scope of HUDroid to facilitate this.

1.3 Sponsor

Harris RF Communications is the leading global supplier of secure radio communications. This includes tactical communication networks and high-grade embedded encryption solutions. These products are used by military, public safety, government, and commercial customers.
2. Basic Requirements

2.1 Functional Requirements

2.1.1 HUD Requirements

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera View</td>
<td>● HUDroid shall display the view of the camera when the application starts.</td>
</tr>
</tbody>
</table>
| Retrieve SAndroid Data | ● HUDroid shall read the SAndroid database to get target data.  
● HUDroid shall update target location information when the database changes.|
| Smooth Data          | ● HUDroid shall use a smoothing algorithm on incoming orientation and target data to decrease jitter.  
● When the phone is held still or near-still, targets shall not move at all. |
| Display Targets      | ● HUDroid shall display targets within the camera’s field of view on the screen as icons.  
● Target icons shall displays as cars, airplanes, or anchors to represent land, air, and sea target respectively. Alternatively, targets may simply display as circular dots.  
● Target icons shall be green, red, or gray to represent friendly, hostile, and neutral targets respectively.  
● Targets shall have their sizes scaled depending on how far away they are.  
● Targets shall have their distances displayed above them in meters, or kilometers if they are more than 1000 meters away.  
● Targets shall have their name/id displayed beneath them. |
| Edge Arrows          | ● HUDroid shall display targets which are not within the camera’s field of view but are within a 180 degree field of view as arrows on the edges of the screen, pointing in the directions of the targets which they represent.  
● Edge arrows shall be green, red, or gray to represent friendly, hostile, and neutral targets respectively. |

2.1.2 Widget Requirements

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAndriod</td>
<td>● HUDroid shall provide access to SAndroid compass as a widget.</td>
</tr>
<tr>
<td>Compass</td>
<td>● Targets on the compass shall correspond with targets on-screen.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| TacChat | ● HUDroid shall provide access to the TacChat service as a widget.  
● If the TacChat service is not running, tapping the TacChat widget shall start it. |
| Widget Picker View | ● HUDroid shall provide the user with a screen to add and move widgets around the screen. This screen shall contain a list of all available widgets.  
● This view shall be accessible through the menu button and via a swipe gesture across the screen.  
● On the provided list of available widgets, tapping a widget shall add to the screen if it is not currently displayed. Otherwise, it shall move the tapped widget around to each corner of the screen. |
| Removing Widgets | ● Performing a long press on a on-screen widget shall remove the widget from the screen. |

### 2.1.3 System and Preferences Requirements

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement(s)</th>
</tr>
</thead>
</table>
| Auto-Start SAndroid | ● If the user starts HUDroid without SAndroid already running, the system shall prompt the user and ask if they want to start SAndroid.  
● If the user selects Yes, the system shall start SAndroid. |
| Preferences View | HUDroid shall provide the user with a view to change preferences. This view shall be accessible through the menu button. Preferences shall be saved across sessions. |
| Basic Tablet Compliance | HUDroid shall provide the user with the ability to select their device’s default orientation, portrait or landscape. This selection shall alter the computations of the system to account for different default orientations, generally useful for running HUDroid on tablets. |

### 2.2 Non-Functional Requirements

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular Widget Subsystem</td>
<td>● It shall be easy for future developers to add new widgets to the system, by only having to add three lines of code, and not having to delete or edit any of the existing code (besides the couple lines of code being added).</td>
</tr>
<tr>
<td>Performance</td>
<td>● HUDroid shall be able display and update multiple targets in real time.</td>
</tr>
<tr>
<td>Follow Android Conventions</td>
<td>HUDroid shall follow the Android conventions for its resources (such as icons) and use of the hardware buttons.</td>
</tr>
</tbody>
</table>
3. Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>At Least One Camera Present</td>
<td>Hardware</td>
</tr>
<tr>
<td>Android 2.2 or Higher</td>
<td>Software</td>
</tr>
<tr>
<td>High Density Screen</td>
<td>Hardware</td>
</tr>
<tr>
<td>SAndroid Pre-Installed with HUDroid Updates</td>
<td>Software</td>
</tr>
<tr>
<td>Accelerometer Present</td>
<td>Hardware</td>
</tr>
<tr>
<td>Magnetometer Present</td>
<td>Hardware</td>
</tr>
<tr>
<td>GPS Present</td>
<td>Hardware</td>
</tr>
<tr>
<td>Cannot Use Copyleft Licenses</td>
<td>Licensing</td>
</tr>
</tbody>
</table>

Given that the HUDroid project is primarily an extension of the SAndroid application, it is constrained to platforms which can install and run SAndroid (the updated HUDroid version). HUDroid is intended to be run on an Android phone or comparable device compatible with Android version 2.2 (named Froyo) or higher. The device must have a back-facing camera, or be receiving a video feed through some other comparable means. HUDroid uses the accelerometer and magnetometer sensors for orientation information, as well as the GPS for location data. It is also highly recommended that the device have a high-density screen and processing/memory specs comparable to or better than the Nexus One.
4. Development Process

The team followed an incremental and iterative process. Components of the Rational Unified Process (RUP) were utilized which were appropriate for the project’s needs. RUP is an adaptable process framework intended to be tailored by development organizations and software project teams.

The team followed a Use Case Driven development; each iteration (phase) delivered a set of use cases. The project was divided into multiple components and later integrated. The team successfully followed the RUP four phases, as seen in the next section. Phase 1 comprised of basic heads up display functionality, where objects’ location received from SANDroid are overlayed on a camera view. Phase 2 contains the widget subsystem main functionalities; which concluded the basic requirements for the project. Phase 3 was made up of any attainable reach goals that the team sets during requirements elicitation.

Development was risk focused, and mitigation for potential risks was put in place early. Some teams members were not able to meet during the weekends, so the team moved all meetings to Tuesdays and Thursdays before the official meeting times. Changing requirements were handled through discussions with the sponsors during the bi-weekly meetings. The team decided with the sponsors what were in/out of scope, and what was manageable in the remaining time.

The sponsors had no preferences with regards to which process the team chose. Once RUP had been decided upon, the sponsors were satisfied with the development process and rate of progress.

One of the concerns RUP addresses is changing requirements; the team was able to handle the changing requirements through communication with the sponsors via weekly emails, in addition to bi-weekly (sometimes weekly, depending on circumstances) meetings.

The roles identified by the team as needed were Team Leader, Technical leader, Requirements Lead, Design Lead, Project Manager, Testing Lead, Risk Management Lead, and Quality Assurance Lead. Roles were filled according to the strengths and domain experience of the team members.

Christian, Kenton: Team Leader and Technical Lead
Owen, Samuel: Requirements and Design Lead
Petit Frere, Avezou: Project Manager and Testing Lead
Turcotte, Robert: Risk Management Lead
Bennett, Colin: Quality Assurance Lead

5. Project Schedule: Planned and Actual

The project schedule was developed with the RUP schedule in mind. The team estimated
each task’s effort value using planning poker, and assigned a deadline to each. There were four phases, each with a set of deliverables. At a high level, the process was broken down as mandated by RUP.

5.1 Inception Phase
In this phase, the team had frequent meetings with the sponsors, and most of the discussion were geared towards understanding the requirements and/or defining what in or out of scope. Once the team had an SRS, the architect started performing an architectural analysis of the system. Afterwards, the team started preparing the project’s environment, and decided on tracking tools and metrics would be used for quality assessment. The deliverable at the end of this phase was the requirements document and the project plan. The test plan was also started at that point.

5.2 Elaboration
In this phase, the team delved into more specific details regarding the work done in the inception phase. This phase began with an analysis of the sponsors’ needs, and from that developed a plan for managing the project’s scope. The architectural analysis transitioned into system and subsystem design. The plan for integration was also decided at this time. At the end of the phase, the team had a solid understanding of the entire design aspects of the project. The deliverable the design document.

5.3 Construction Phase
This was the largest phase in the project. In this phase, the majority of work remaining in the project was to build upon the foundation laid in the previous phases. This phase contained code artifacts such as implementation of the components identified in the elaboration phase, as well as test code. Each iteration in this phase resulted in an executable mini-release. The team had three major releases for this project, but the sponsors had many chances to play around with mini-releases and provide useful feedback.

5.4 Transition Phase
During this phase, the team prepared the project for delivery. All documents were polished and made up-to-date, and all acceptance tests were passed. The team came to an agreement with the sponsors about the finalized requirements for the system and all the features that would be implemented. At the end of this phase, the sponsors were delivered a fully functional product with a user manual, technical report, website information, and all non-code artifacts.

On a more granular level, the team kept track of the estimated time versus the actual time spent on individual tasks via the weekly time-tracking spreadsheets, which are also available on the website. Below is a graph of the second term’s time tracking data up to week seven:
6. System Design

6.1 MVC Architecture

6.1.1 View Domain

The MVC domain of Views is taken care of almost entirely by Android’s native architecture. Android provides an actual View class, which is what must be used to display anything on-screen. HUDroid uses four Views arranged in layers, where each layer displays over layers below it. Amongst the View layers are the displays for the camera feed, the target icons and associated data, widgets, and a touch-sensitive interface to detect user gestures. These are referred to as the Camera View, AR View, Widget View, and Editing View respectively. While these four layers are always displayed, there is another View which usually hidden and is only displayed during (and is responsible for) widget addition. This is referred to as the Widget Picker View. Each of these Views can register itself with Android and knows how to display itself - Android will automatically tell each View to draw and redraw itself, thus allowing for easy separation from the Model and Controller domains.

6.1.1.1 Four Main View Layers

6.1.2 Model Domain

HUDroid needs very little persistent storage. The Model domain consists primarily of data read from a database in SAndroid, which is updated by SAndroid on a continual basis as new target data is received. Most of HUDroid’s Model is encapsulated in the project’s utility classes, which hold data required for the vector math and smoothing algorithms. For example, these
utilities include Point, Vector, and Matrix classes, as well as thread-safe queues which hold recent orientation data for smoothing purposes. The last aspect of HUDroid’s Model is the user preferences. When a user changes their preferences, such as changing the target icon size, this change is persisted via Android built-in shared preferences. As such, HUDroid does not have to store and manage these preferences in any database or file.

6.1.3 Controller Domain

The Controllers in HUDroid consist of the AR Controller, the Rotational Controller, the Widget Controller, and various Action Listeners. The AR Controller is responsible for all the math, logic, and computations involved in taking locations (delivered with latitude, longitude, and altitude) and figuring out what [x,y] pixel locations they map to on-screen. In the course of these computations it is dependent on the Rotational Controller, which runs its own thread and simply processes incoming orientation data which the AR Controller uses in its calculations. The Rotational Controller is responsible for the primary smoothing algorithm. Neither of these classes ever touches the data in the SAndroid data (that is passed to the AR Controller via a Content Provider), and neither of them ever touches a View. Rather, they simply compute [x,y] screen locations of targets and return the results without ever knowing what they will be used for.

The Widget Controller is a middle-man between the Widget Picker View and the Widget View. This Controller receives input taps from the Widget Picker View, figures out what the user is trying to do, and forwards the update to the Widget View. For example, a tap on a widget in the Widget Picker View can signify a request to add the widget to the screen or to move the widget around, and it is up to the Widget Controller to determine which it is. The various Action Listeners in HUDroid can also be considered Controllers since they contain code to be executed when certain input is received. For instance, widgets in the Widget Picker View have Action Listeners registered to them which send widget addition/movement requests to the Widget Controller.

6.2 Subsystems

6.2.1 Overall Class Diagram
6.2.2 AR Subsystem

The AR subsystem consists of the ARController, RotationalController, SAInterpreter, ARActivity, CameraView, and ARView classes. This subsystem represents the core functionality of HUDroid. The ARActivity is an Android Activity which is launched when the HUDroid application is started. It contains the main four View layers, and is responsible for initializing the Controllers. At the lowest level, the CameraView simply displays the video feed from the camera. Whenever the SAndroid database containing target data is updated, the update is pushed to the SAInterpreter via a Content Provider. SAInterpreter converts these database entries into Android Location objects and passes the updated locations to the ARController. Whenever the ARView redraws itself, it asks the ARController for new [x,y] pixel locations to display the targets on-screen. The ARController and the RotationalController work in conjunction (with the help of many utility classes) to map the incoming location data to their corresponding screen coordinates, and return that mapping to the ARView. This comprises the core AR and HUD functionality of the system.

6.2.3 Widget Subsystem

The widget subsystem consists of the WidgetPickerView, WidgetController, WidgetView, and Widget (View) classes themselves. The WidgetView simply contains all on-screen widgets and their locations. The widgets themselves are not actually a HUDroid-specific class, but rather any subclass of the Android View. Since Views know how to display themselves, WidgetView can basically act like a container, and each Widget inside it can draw itself without the WidgetView having to know anything about how it works. The WidgetPickerView contains a list of all the available widgets which the system knows about. By default HUDroid only comes with two widgets, but future developers can add their own widgets very easily by adding their project as a library and adding their widget to the WidgetPickerView’s list (in the code). Any
user input which comes through the WidgetPickerView is passed to the WidgetController for processing. After the exact request from the user has been determined, that request is then executed appropriately on the WidgetView.

6.2.4 Preferences Activity
The user preferences run in their own activity. It is not shown on the overall class diagram because it is not at all coupled with the main system. Rather than launching the preferences through HUDroid-specific classes, the user simply presses the Android menu button and selects “Preferences”. This activity consists of a simple layout (defined in an XML file) for selecting various options. These preferences are saved in the Android shared preferences, not in any sort of database or file to be managed by HUDroid. When the preferences are closed and the ARActivity resumes, the Android shared preferences are checked and various variable are set. At no point does HUDroid or the Preferences Activity have references to each other.

6.3 Threading
HUDroid uses only a single custom thread, however Android provides many entry points into the system, making asynchronous method calls and concurrency an important issue for the system. The first entry point into the system, potentially able to run in its own thread, is the ARActivity. When the user selects to start up the application, the “onCreate” and “onResume” methods in the ARActivity are called by the Android life-cycle, essentially at the user’s request. Once the system has finished being initialized and the ARActivity method calls exit, there are three entry points into the system via data updates, one entry point via the UI updates, and one entry point via the RotationalController which runs in its own thread. Updates from the SAndroid database come in via a Content Provider, while updates from both the accelerometer and magnetometer come in via Sensor Event Listeners. Android hides the threading of these updates, so it is possible that all three may come in on the same thread, but it is also possible that they may come in on up to three separate threads, so the system must be thread-safe enough to account for these possibilities. It is also not guaranteed that the system will receive one update from each in order - it is possible that the magnetometer may update ten times before the SAndroid database updates once. Similarly, the Android UI thread is what prompts the ARView to redraw itself, but it is also unclear whether this runs on its own thread or if it shares threads with the sensor and database updates. The RotationalController, which extends the Java Thread class, simply runs as fast as possible over its available data and runs the smoothing algorithm to keep the smoothed orientation data as up-to-date as possible. Ultimately, the system must be properly designed and implemented to be thread-safe enough to avoid problems caused by asynchronous method calls, while avoiding unnecessary thread locking which could cause performance hits or deadlocks. Please see the HUDroid Project Design Document for more details and use-case diagrams on threading in the system.
7. Process and Product Metrics

The team has primarily employed Acceptance Tests and Estimation Accuracy metrics. The acceptance test metrics have been utilized to measure the progress of the project as a whole. As of the final release, the acceptance tests are 13/14 passed. The one incomplete test was moved out of scope during the re-prioritization of phase three requirements. These tests have confirmed the product quality and the team’s initial requirements break down.

Related to the acceptance test is the defect tracking. The team has kept track of two types of defects - internal to HUDroid, and external with SAndroid. Most of the defects with HUDroid have stemmed from defects in SAndroid. As of release, there are no known outstanding defects with HUDroid. The SAndroid defects were underestimated early in the project, making for a point that did not go as well as planned. After realizing some of SAndroid’s flaws, the team adjusted to be more cautious around integration. In contrast to that, the low number of HUDroid isolated defects were a positive for each round of release. Phase 2 and Phase 3 each had two HUDroid isolated defects, and Phase 1 had zero.

The estimation accuracy metrics were only focused on during the second half of the project in response to a mid-project reflection. These metrics are extremely good. The estimated time has been close, but consistently more than the actual time. This is likely due to overestimating the complexity of the Android ecosystem in some cases.

Overall, the project was very successful, and the acceptance tests help to confirm this. Other non-metrics that have helped this success include the sponsors’ communication in prioritizing requirements, and their freedom with the process and implementation schedule that the team ultimately used.
8. Product State at Time of Delivery

8.1 Completed and Uncompleted Requirements
The application as a whole is completely functional. The user can start the application, view targets, and modify preferences for a more customized experience. During the Phase Three requirements re-prioritization, several new features were added to the system. The following were completed:

- Allow dots as target icons
- Allow the application to auto-detect the Field of View settings based on a pre-configured set of angles
- Allow the user to restore the preferences to their defaults
- Show the distances to each targets above each icon
- Start the TacChat application by tapping the widget once
- Show a dialog when SAndroid is not running which allows the user to launch it

The product has all Phase One and Phase Two features implemented. Of the re-prioritized features for Phase Three, only two features were left unimplemented:

- Moving widgets to any location on the screen
- Configuring the TacChat widget to fade after messages come in

8.2 Rational
The main reason for the new features is that the customers decided they wanted more variety of smaller features rather than one primary larger feature. This was a “low hanging fruit” approach, and the team agreed that it would look slicker to have these many smaller features.

8.3 Possible Future Enhancements
There are many features that the team documented as “nice to have” but were never realized. Some of these are widget ideas that could be great additions for future teams. These widget ideas include:

- Various widgets to dynamically adjust settings, such as icon distance
- Icon tick marks at the top of the screen for a compass - in an FPS type fashion
- Altimeter/horizon line widget

There were some other enhancements that were scoped out at the beginning of the project which were never implemented or incorporated into the requirements, including:

- Multiple display support
  - Control device(s)
- Collapse arrows on the screen when there are many in the same vicinity
  - Include number under the arrow to indicate how many targets are there
9. Project Reflection

9.1 What Went Right?

<table>
<thead>
<tr>
<th>Label</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tangible Progress</td>
<td>Development cycle provided tangible progress to sponsors.</td>
</tr>
<tr>
<td>Consistent Meetings</td>
<td>Initially started with weekend meetings. Evolved into adding an additional hour to weekday meetings.</td>
</tr>
<tr>
<td>Buy-In</td>
<td>Sponsors bought into the project and its goals.</td>
</tr>
<tr>
<td>Cohesion</td>
<td>Teammates worked very well together.</td>
</tr>
</tbody>
</table>

9.2 What Went Wrong?

<table>
<thead>
<tr>
<th>Label</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing</td>
<td>Testing was not exhaustive. SAndroid was more flawed than anticipated. Bugs in HUDroid were not discovered quickly enough.</td>
</tr>
<tr>
<td>Requirements</td>
<td>Most of the requirements for the system were “I’ll know it when I see it” requirements. Sponsors responded with late feature requests and changes. Sponsors had poor vision of what the product would/should look like, and had late buy-in. Wireframe prototypes could have done a better job of ironing out details.</td>
</tr>
<tr>
<td>Task Assignment</td>
<td>Each person was responsible for what they themselves chose and nothing more. Delegation of tasks was not coordinated.</td>
</tr>
<tr>
<td>Android Framework</td>
<td>The team spent some time learning the Android framework at the beginning of the project, but more familiarity with it would have helped with the team exercises.</td>
</tr>
</tbody>
</table>

9.3 Future Adjustments

<table>
<thead>
<tr>
<th>Label</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better Metrics</td>
<td>The team had a difficult time forming metrics for the project given its primary focus on graphics. For future projects like this, it would be good to research some good metrics to use ahead of time so that the team isn’t scrambling and looking for testing data at the last minute.</td>
</tr>
<tr>
<td><strong>Fix SAndroid</strong></td>
<td>The vast majority of issues that the team encountered were due to failures by the SAndroid team. This includes failure to document anything whatsoever, failure to explain design rational, missing functionality which was not documented or known to the sponsors, intentional use of deprecated methods, and various other bugs.</td>
</tr>
<tr>
<td><strong>Movable Widgets</strong></td>
<td>One of the team’s initial Phase Three requirements was to allow the user to drag widgets around to any point on the screen. This requirement was eventually scrapped because it was deemed less important functionality which was more difficult to implement. Therefore, the sponsors decided that the team should work on other requirements instead.</td>
</tr>
<tr>
<td><strong>Advanced TacChat Widget</strong></td>
<td>One of the Phase Three requirements which was introduced very late was for the TacChat widget to have additional functionality. This includes having new messages fade in while older messages faded out, allowing the user to scroll to see older messages, and dynamically determining the direction in which messages fade/scroll depending on where the widget was placed on-screen. This requirement was later scrapped because of its low priority and difficulty in implementing.</td>
</tr>
<tr>
<td><strong>More Modular Widgets</strong></td>
<td>The team’s original design for the widget subsystem was to receive widgets from other programs via Android broadcasts, serialize them, and store them in a database. This turned out to be extremely complicated because Android views are non-serializable, and all attempts to use reflection to work around it failed. After hitting many brick walls with this approach, the sponsors recommended that a simpler subsystem architecture be used, since this was a low priority requirement and extremely difficult and time-consuming.</td>
</tr>
<tr>
<td><strong>Versioning</strong></td>
<td>Although HUDroid works well on Android 2.2 and most higher versions, it has issues when running on tablets. This is primarily Android’s fault since newer versions of the OS should be backwards-compatible. After many hours trying to fix the issue, the team was unable to find any way to solve the problem without explicitly creating separate versions of the application. Since tablets are becomes more and more popular, future improvements on HUDroid should include compatibility with them.</td>
</tr>
</tbody>
</table>
# 10. Glossary of Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>A mobile device OS.</td>
</tr>
<tr>
<td>AR</td>
<td>Augmented Reality</td>
</tr>
<tr>
<td>CoT</td>
<td>Cursor on Target</td>
</tr>
<tr>
<td>FPS</td>
<td>First-Person Shooter</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>HUD</td>
<td>Heads-Up Display</td>
</tr>
<tr>
<td>HUDroid</td>
<td>HUD on Android - An Android application which places targets received from SAndroid into an AR display; the application about which this document is written.</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>RUP</td>
<td>Rational Unified Process - A software development process; part of IBM Rational Software family; an iterative process.</td>
</tr>
<tr>
<td>SA</td>
<td>Situational Awareness</td>
</tr>
<tr>
<td>SAndroid</td>
<td>SA on Android - An Android application developed for Harris Corporation to port their SA and TacChat to the Android OS.</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirements Specification</td>
</tr>
<tr>
<td>TacChat</td>
<td>Tactical Chat application, namely the information protocol used for receiving messages to the system.</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
11. References

1. HUDroid Software Requirements Specification
2. HUDroid Project Design Document
3. HUDroid website: http://hudroid.se.rit.edu/