Testing Quick Reference Handbooks in Flight Simulators

Anthony Berg (200871682)
Supervisor: Dr Leo Freitas

14th May 2024

Preface

Abstract

This is an abstract.

Declaration

I declare that this dissertation represents my own work except where otherwise stated.

Acknowledgements

I would like to thank my supervisor Leo Freitas for supporting, guiding, and providing with areas of improvement for me throughout the project.

Contents

Chapter 1

Introduction

1.1 Scene

- Designing Emergency Checklists is difficult
- Procedures in checklists must be tested in simulators [1], which usually means trained pilots test it, as the tests need to work consistently [2] (making sure it's not lengthy, concise and gets critical procedures)
- Checklists are usually carried out in high workload environments, especially emergency ones

1.2 Motivation

- Testing procedures in checklists are often neglected [1]
- There are some checklists that may not be fit for certain scenarios - e.g. ditching (water landing) checklist for US Airways Flight 1549 assumed at least one engine was running [3], but in this scenario, there were none
- Some checklists may make pilots 'stuck' - not widely implemented, could be fixed with 'opt out' points. e.g. US Airways 1549, plane below 3000ft, could have skip to later in the checklist to something like turn on APU, otherwise plane will have limited control [3].
- Checklists may take too long to carry out - Swissair 111

1.3 Aim

The goal of this project is to test checklists in Quick Reference Handbooks (QRH) for flaws that could compromise the aircraft and making sure that the tests can be completed in a reasonable amount of time by pilots. It is also crucial to make sure that the tests are reproducible in the same flight conditions and a variety of flight conditions.

1.4 Objectives

1. Research current checklists that may be problematic and are testable in the QRH tester being made
2. Implement a formal model that runs through checklists, with the research gathered to produce an accurate test
   (a) Understand the relative states of the aircraft that need to be captured
   (b) Ensure that the results of the checklist procedures are consistent
3. Implement a QRH tester manager that
   - Runs the formal model and reacts to the output of the formal model

- Connect to a flight simulator to run actions from the formal model
- Implement checklist procedures to be tested, run them, and get feedback on how well the procedure ran

Chapter 2

Background

2.1 Hypothesis

- Checklists can be tested in a simulated environment to find flaws in checklist for things like
  - Can be done in an amount of time that will not endanger aircraft
  - Provides reproducible results
  - Procedures will not endanger aircraft or crew further (Crew referring to Checklist Manifesto with the cargo door blowout)
- Results in being able to see where to improve checklists

2.2 Safety in Aviation

2.2.1 History

- 70-80% of aviation accidents are attributed to human factors [4]

2.2.2 Checklists

- Checklists have been shown to aid in minimising human errors [2]
- However, checklists can be misleading and compromise the safety of the aircraft due to them being either too confusing or taking too long to complete [1]
- That is why testing checklists are important to avoid these situations

2.3 Formal Methods

2.3.1 History

2.3.2 Application

2.4 Solution Stack

- There would be around 3 main components to this tester
  - Formal Model
  - Flight Simulator plugin
  - Checklist Tester (to connect the formal model and flight simulator)
- As VDM-SL is being used, it uses VDMJ to parse the model [5]. This was a starting point for the tech stack, as VDMJ is also open source.
- VDMJ is written in Java [5], therefore to simplify implementing VDMJ into the Checklist Tester, it would be logical to use a Java virtual machine (JVM) language.

2.4.1 Formal Model

- There were a few ways of implementing the formal model into another application
- Some of these methods were provided by Overture [6]
  - RemoteControl interface
  - VDMTools API [7]
- However, both of these methods did not suit what was required as most of the documentation for RemoteControl was designed for the Overture Tool IDE. VDMTools may have handled the formal model differently
- The choice was to create a VDMJ wrapper, as the modules are available on Maven

2.4.2 Checklist Tester

JVM Language

- There are multiple languages that are made for or support JVMs [8]
- Requirements for language
  - Be able to interact with Java code because of VDMJ
  - Have Graphical User Interface (GUI) libraries
  - Have good support (the more popular, the more resources available)
- The main contenders were Java and Kotlin [9]
- Kotlin [9] was the choice in the end as Google has been putting Kotlin first instead of Java. Kotlin also requires less boilerplate code (e.g. getters and setters) [10]

Graphical User Interface

- As the tester is going to include a UI, the language choice was still important
- There are a variety of GUI libraries to consider using
  - JavaFX [11]
  - Swing [12]
  - Compose Multiplatform [13]
- The decision was to use Compose Multiplatform in the end, due to time limitations and having prior experience in using Flutter [14]
- Compose Multiplatform has the ability to create a desktop application and a server, which would allow for leeway if a server would be needed

2.4.3 Flight Simulator Plugin

- There are two main choices for flight simulators that can be used for professional simulation
  - X-Plane [15]
  - Prepar3D [16]
- X-Plane was the choice due to having better documentation for the SDK, and a variety of development libraries for the simulator itself
- For the plugin itself, there was already a solution developed by NASA, X-Plane Connect [17] that is more appropriate due to the time limitations and would be more likely to be reliable as it has been developed since 2015

Chapter 3

Design/Implementation

3.1 Components

Splitting up the project into multiple components has been useful for

- Aiding in planning to make the implementation more efficient
- Delegating specific work tasks
- Making the project modular, for example, allowing for a different simulator to be implemented with minimal need to refactor other parts of the codebase

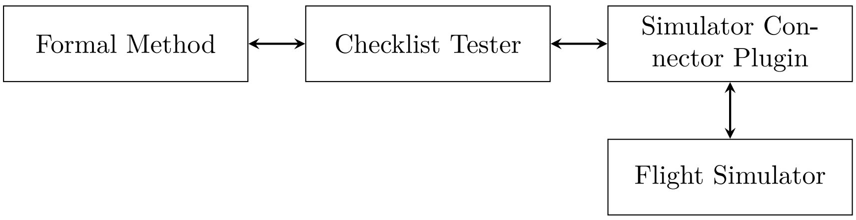

Figure 3.1: Abstract layout of components

Each of the components in Figure 3.1 will be covered in detail in this chapter.

3.2 Formal Method

- Formal modelling is the heart of the logic for testing checklists
- Formal model created using VDM-SL
- It allows to test that the checklists have been completed in a proper manner - and that it is provable
- Model keeps track of
  - Aircraft state
  - Checklist state
- If an error were to occur in the model, this can be relayed that there was something wrong with running the test for the checklist, such as:
  - Procedure compromises integrity of aircraft
  - There is not enough time to complete the procedure
  - There is a contradiction with the steps of the checklist

3.3 Checklist Tester

Brief overview of what it is supposed to do...

3.3.1 Designing

- Used Figma to create a design for the application
- Allows for implementing the front end to be faster because:
  - They act as a requirement for code
  - You do not forget what needs to be implemented
  - Keeps everything consistent
  - Allows to think about making parts of the GUI modular and what components can be reused
- Figma allows for plugins such as Material 3 colours and Material 3 components
- Figure 3.2 is the final design that will be used for the program

Limitations of Figma

- The Material 3 Components in Figma do not include features that are available in Jetpack Compose
- In this project, the 'Simulator Test' at the bottom of Figure 3.2 does not include a leading icon [18], and therefore had to be a trailing checkbox
- This was overcome by adding comments in Figma as a reminder of how the actual implementation should be like
- Another limitation is that in Figure 3.2, the title of the screen in the top app bar [19] is not centered, and that is because the auto layout feature in Figma allows for equal spacing, rather than having each in a set position

3.3.2 Compose Multiplatform

Setup

- Used the *Kotlin Multiplatform Wizard* to create projects as it allows for runtime environments to be specified (in this case, Desktop and Server)
- Provides necessary build configurations in Gradle
- Planning what to implement important as Compose is designed to use modular components, otherwise a nested mess would occur as Compose is designed to have Composable functions passed in to a Composable function and therefore by design function nests will occur and the code will be harder to read if not managed correctly. Listing 3.1 shows example of using modular code from the Actions screen in project (with code omissions shown in comments)
- Used Voyager [20] to handle screens
- Used Koin [21] for dependency injection, to be able to get data from the database and VDMJ
  - Chose to use it because of Voyager integration with Koin [22]
  - Required as the application will be unresponsive when making requests to the database and/or VDMJ
  - Used asynchronous coroutines to prevent the program from being blocked

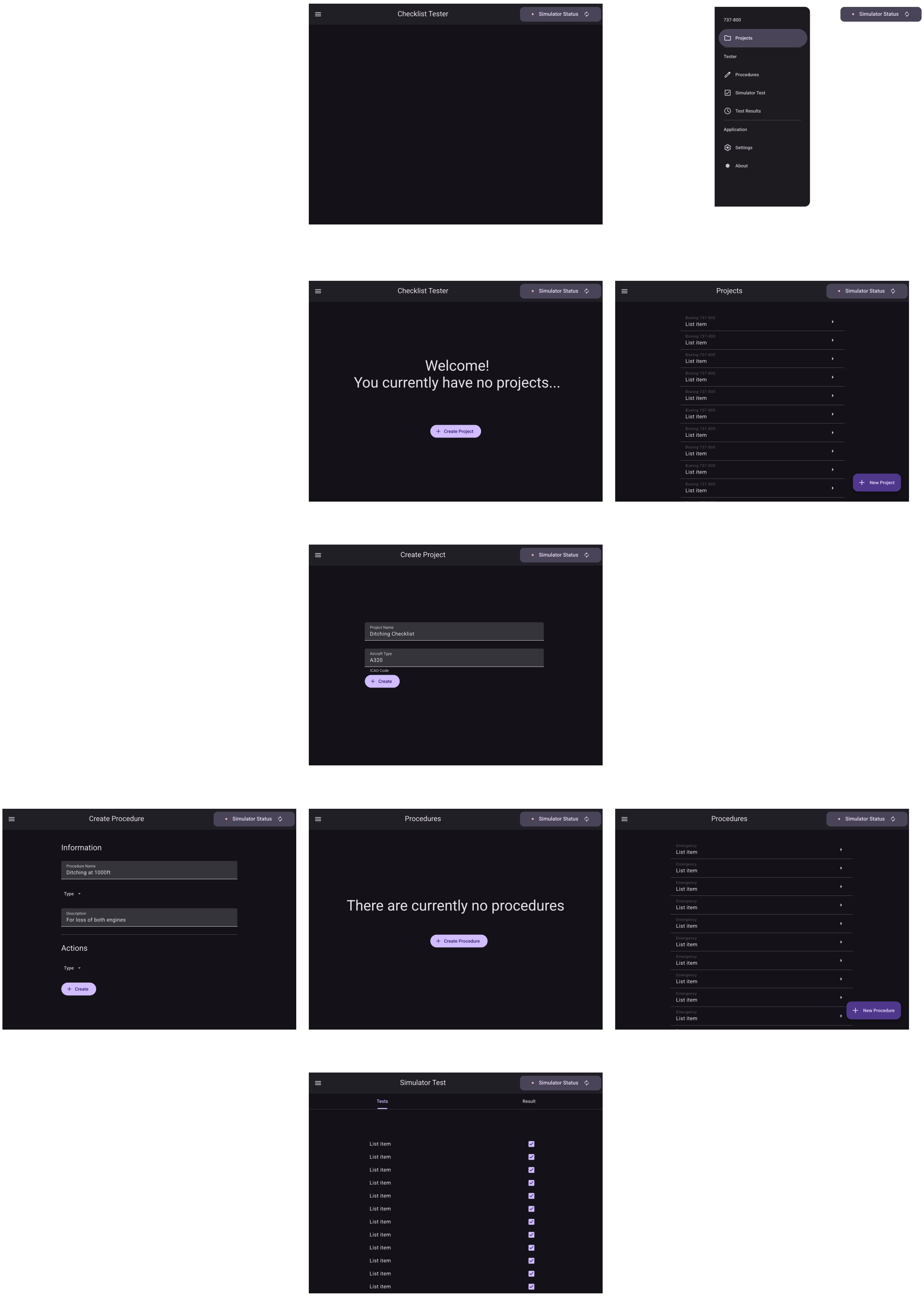

Figure 3.2: Design for the Checklist Connector GUI in Figma

```kotlin
@Composable
override fun Content() {
    // Content variables...

    Scaffold(
        topBar = {/* Composable content... */},
    ) {
        Column(/* Column option parameters... */) {
            Box(/* Box option parameters... */) {
                LazyColumn(/* LazyColumn option parameters... */) {

                    item {
                        Header()
                    }

                    items(
                        items = inputs,
                        key = { input -> input.id }
                    ) { item ->
                        ActionItem(item)
                    }
                }
            }
        }
    }
}

@Composable
private fun Header() {
    Text(text = "Edit Actions")
}

@Composable
private fun ActionItem(item: Action) {
    Column (/* Column option parameters... */) {
        Row(/* Row option parameters... */) {
            Text(text = "Action ${item.step + 1}")

            IconButton(/* IconButton definition parameters... */) {
                Icon(
                    Icons.Outlined.Delete,
                    // Rest of Icon options...
                )
            }
        }

        Row(/* Row option parameters... */) {
            OutlinedTextField(/* TextField definition parameters... */)

            OutlinedTextField(/* TextField definition parameters... */)
        }

        HorizontalDivider()
    }
}
```

Listing 3.1: Example of modular code in Compose

3.3.3 Storing Data

- SQLDelight was used to handle the database by allowing for typesafe Kotlin APIs when interacting with the database. Specifically chosen as it provides support for Compose Multiplatform [23]
- It only allows for queries to be written in SQL, which would allow for more complex SQL queries if needed
- SQLite was used for the Relational Database Management System (RDBMS) as it is an embedded database [24], meaning that the database will run in the application, rather than running on a server, either remotely or through local containerization through something like Docker [25], which could take more time and add complexity as it means implementing additional dependencies
- SQLite also has 100% [26] which necessary for ensuring that the database will not cause artefacts to the results

Designing the Database

- The database could be thought as having 2 sections
  - The user inputs to control the tester, i.e. the steps a procedure contains. The tables for these are *Project*, *Procedure*, and *Action*
  - The test results for a procedure which are in the *Test*, and *ActionResult* tables
- The design of the database had relationships in mind as the goal was to have a detailed tracking of statistics for each step in the procedure, hence in Figure 3.3
- A *Procedure* can have multiple *Tests*, where each *Test* each contains the result of how each *Action* in *ActionResults*
- The choice of a *Project* was to allow for the segregation of testing different aircrafts, as each aircraft has different cockpit layouts and different systems

Implementing into Compose Multiplatform

- Compose Multiplatform has support for different runtime environments, however as this project was only being developed for Desktop, the JVM SQLite driver only had to be considered
- However, the functions for the database were written in the *shared/commonMain* module as there may be a potential for adding Android and iOS support as it as it may be helpful run the tests on a tablet
- A database transaction had two modules
  - A class handling SQLDelight API calls only; meaning no conversion of types, which are functions only accessible within module internally, which is located in *io.anthonyberg.connector.shared.database*
  - SDKs that can handle multiple tables, such as *TestTransaction* which handles database calls when checklists are being tested in the application. And allows for converting types, such as `Int` to `Long`
- The separation of these modules was to have in mind unit testing, as it will make it easier to debug if a problem is with SQLDelight transactions are handled, or if there is a conversion error occurring

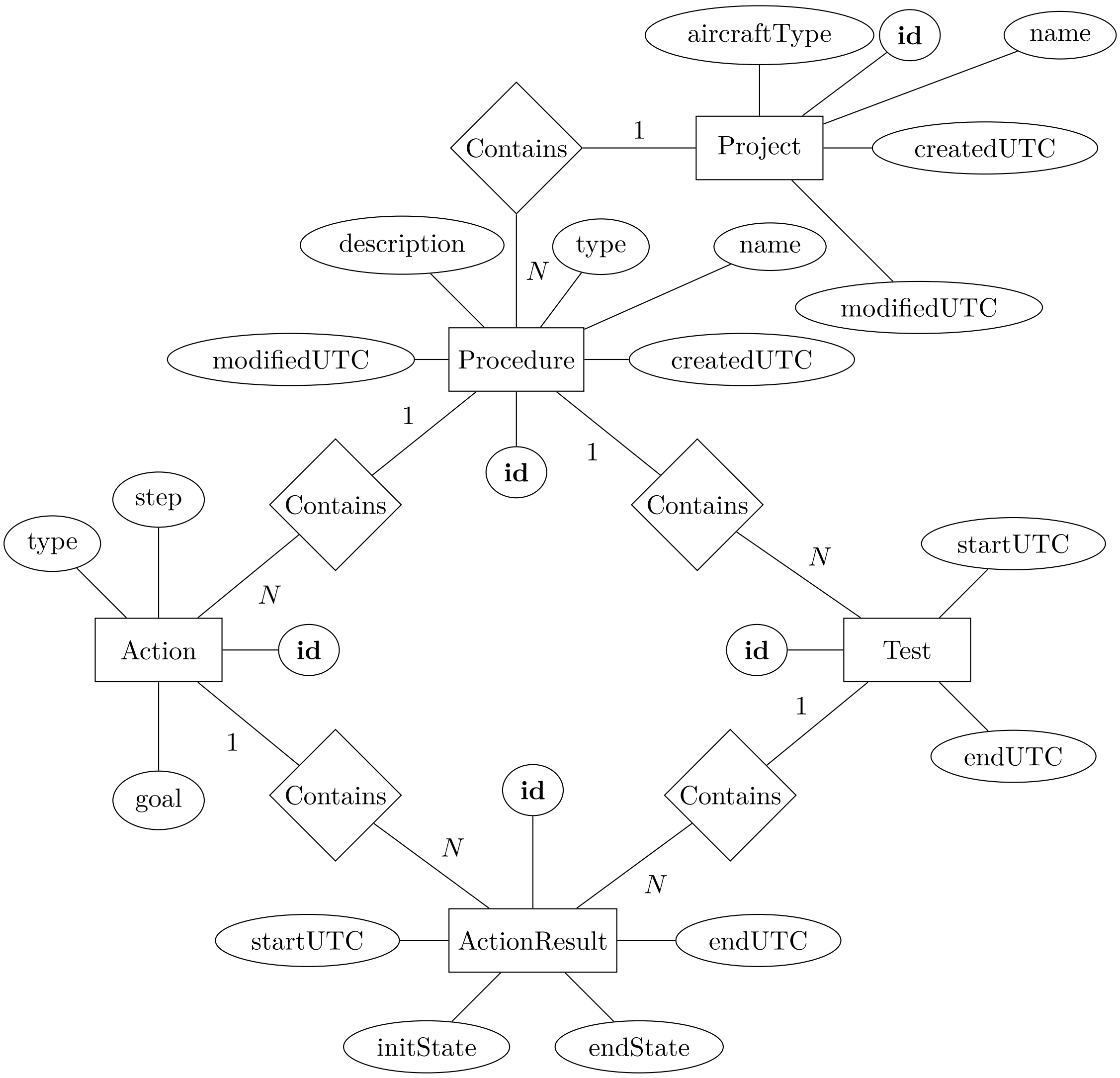

Figure 3.3: Entity Relationship Diagram for the database in Checklist Connector

3.3.4 VDMJ Wrapper

3.3.5 Connecting to the Flight Simulator

3.3.6 Testing

3.4 Simulator Connector Plugin

3.4.1 Creating Maven Package

- XPC package is not published on a public Maven repository
- There has been a pull request that was merged to the *develop* branch that provides Maven POMs [27]. However, the maintainer for the project, at the time, did not have enough time to figure out the process of publishing the package to a Maven repository [28]
- Therefore, had to find an alternative way to implement
- Jitpack [29]
  - In theory, simple to publish a repository, all that is required is a GitHub repository and searching if one has already been created on JitPack or build and publish a specific version
  - However, due to the structure of the XPC repository, JitPack could not locate the build tools (Apache Maven in this case) as JitPack only searches on the root directory for the compatible build tools
- Gradle gitRepository [30]
  - There was not a lot of documentation
  - Ambiguous on how to define directory for where the Java library is located in the Git repository
  - However, as XPC was only built with Maven, Gradle was not able add the dependency as `gitRepository()` only works with Gradle builds [31]
- Resorted to using a compiled Jar file and adding the dependency to Gradle
- Not happy about that because it means maintaining it will be more difficult as it is not as simple as just changing the version number
- Later, resorted to adding Gradle build files to XPC
- Used automatic conversion from Maven to Gradle using `gradle init` command [32]
- Had to add local dependencies due to how Gradle works differently
- Had to fix previous structure of Maven POM as the grouping as not good

3.4.2 Submitting a Pull Request

3.5 Scenarios

- Use a Quick Reference Handbook (QRH) to find potential list of checklists to test
- Look at previous accident reports that had an incident related to checklists and test it with my tool to see if it will pick it up
- These previous accident reports can be good metrics to know what statistics to look out for

3.6 Decisions

Chapter 4

Results

4.1 Problems Found

4.2 LOC?

4.3 Reflection

4.4 Time Spent

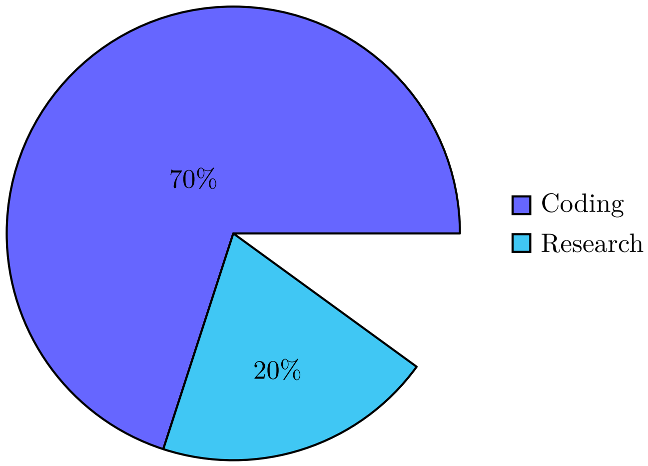

Figure 4.1: Time spent on sections of project

Chapter 5

Conclusion

5.1 Changes

5.2 Objectives

5.3 What Next

Appendix A

Formal Model

```
module Checklist
exports all
definitions

values
    -- Before Start Checklist
    -- Items in Aircraft
    -- Flight Deck... (can't check)
    fuel: ItemObject = mk_ItemObject(<SWITCH>, mk_Switch(<OFF>, false));
    pax_sign: ItemObject = mk_ItemObject(<SWITCH>, mk_Switch(<OFF>, true));
    windows: ItemObject = mk_ItemObject(<SWITCH>, mk_Switch(<ON>, false));
    -- Preflight steps
    acol: ItemObject = mk_ItemObject(<SWITCH>, mk_Switch(<OFF>, false));

    aircraft_panels: Items = {"Fuel Pump" |-> fuel, "Passenger Signs" |-> pax_sign, "Windows" |-> windows, "Anti Collision Lights" |-> acol};

    -- Checklist
    -- Flight Deck... (can't check)
    fuel_chkl: ChecklistItem = mk_ChecklistItem("Fuel Pump", <SWITCH>, <ON>, false);
    pax_sign_chkl: ChecklistItem = mk_ChecklistItem("Passenger Signs", <SWITCH>, <ON>, false);
    windows_chkl: ChecklistItem = mk_ChecklistItem("Windows", <SWITCH>, <ON>, false);
    -- Preflight steps
    acol_chkl: ChecklistItem = mk_ChecklistItem("Anti Collision Lights", <SWITCH>, <ON>, false);

    before_start_procedure: Procedure = [fuel_chkl, pax_sign_chkl, windows_chkl, acol_chkl];

    aircraft = mk_Aircraft(aircraft_panels, before_start_procedure);
types
    --@doc The dataref name in X-Plane
    Dataref = seq1 of char;

    -- Aircraft
```

```

    -- Switches
    --@doc The state a switch can be in
    SwitchState = <OFF> | <MIDDLE> | <ON>;

        --@LF why have a type kist as a rename?
    ItemState = SwitchState; --@TODO | Button | ...

    --@doc A switch, with the possible states it can be in, and the state that it is in
    Switch ::
        position : SwitchState
        middlePosition : bool
        inv s ==
            (s.position = <MIDDLE> => s.middlePosition);

    -- Knob
    Knob ::
        position : nat
        --@LF how can a state be an int? perhaps a proper type (i..e. subset of int range or a union?)
        states : set1 of nat
        inv k ==
            k.position in set k.states;

    Lever = nat
        inv t == t <= 100;

    Throttle ::
        thrust: Lever
        reverser: Lever
        inv t ==
            (t.reverser > 0 <=> t.thrust = 0);

    --@doc The type that the action of the button is
    ItemType = <SWITCH> | <KNOB> | <BUTTON> | <THROTTLE>;

    --@doc The unique switch/knob/etc of that aircraft
    ObjectType = Switch | Knob | Throttle;
    ItemObject ::
        type : ItemType
        object : ObjectType
        inv mk_ItemObject(type, object) ==
                cases type:
                        <SWITCH> -> is_Switch(object),
                        <KNOB>   -> is_Knob(object),
                        <THROTTLE>-> is_Throttle(object),
                        --<BUTTON> -> true
                        others -> true
                end;

    --@doc Contains each ItemObject in the Aircraft, e.g. Fuel Pump switch
    Items = map Dataref to ItemObject;

    --@doc Contains the panels (all the items in the aircraft) and the procedure
    Aircraft ::
```

```
        items : Items
        procedure : Procedure
        inv mk_Aircraft(i, p) ==
        ({ x.procedure | x in seq p } subset dom i);

    -- Checklist

    --@doc Item of a checklist, e.g. Landing gear down
    ChecklistItem ::
        --@LF again, empty string here doesn't make sense.
        procedure : Dataref
        type : ItemType
        --TODO Check is not only SwitchState
        check : SwitchState
        checked : bool;

    --@doc This is an item in the aircraft that complements the item in the procedure
    ItemAndChecklistItem ::
        item : ItemObject
        checklistItem: ChecklistItem
        inv i == i.item.type = i.checklistItem.type;

    --@doc A section of a checklist, e.g. Landing Checklist
    --@LF shouldn't this be non-empty? What's the point to map a checklist name to an empty procedure? Yes.
    Procedure = seq1 of ChecklistItem
        inv p ==
            --@LF the "trick" for "false not in set S" is neat. It forces a full evaluation, rather than short circuited (i.e. stops at first false).
            --    I presume this was intended.
            false not in set {
                let first = p(x-1).checked, second = p(x).checked in
                        --@LF boolean values don't need equality check
                    second => first--((first = true) and (second = false))
                | x in set {2,...,len p}};

functions
    -- PROCEDURES
    --@doc Finds the index of the next item in the procedure that needs to be completed
    procedure_next_item_index: Procedure -> nat1
    procedure_next_item_index(p) ==
        hd [ x | x in set {1,...,len p} & not p(x).checked ]--p(x).checked = false]
    pre
        -- Checks procedure has not already been completed
        not procedure_completed(p)--procedure_completed(p) = false
    post
        -- Checks that the index of the item is the next one to be completed
        --@LF your def is quite confusing (to me)
        --@LF how do you know that RESULT in inds p? Well, the definition above okay.
        --    but you can't know whether p(RESULT-1) will! What if RESULT=1? p(RESULT-1)=p(0) which is invalid!
```

```
        (not p(RESULT).checked)
        and
        (RESULT > 1 => p(RESULT-1).checked)
        --p(RESULT).checked = false
        --and if RESULT > 1 then
        --    p(RESULT-1).checked = true
        --else
        --    true
        ;

    -- --@doc Checks if all the procedures have been completed
    -- check_all_proc_completed: Checklist -> bool
    -- check_all_proc_completed(c) ==
    --     false not in set { procedure_completed(c(x)) | x in set {1,...,len c} };

    -- --@doc Gives the index for the next procedure to complete
    -- next_procedure: Checklist -> nat1
    -- next_procedure(c) ==
    --     hd [ x | x in set {1,...,len c} & not procedure_completed(c(x))]
    -- post
    --     RESULT <= len c;

    --@doc Checks if the procedure has been completed
    procedure_completed: Procedure -> bool
    procedure_completed(p) ==
        false not in set { p(x).checked | x in set {1,...,len p} };

    --@doc Checks if the next item in the procedure has been completed
    check_proc_item_complete: Procedure * Aircraft -> bool
    check_proc_item_complete(p, a) ==
        --@LF here you have a nice lemma to prove: procedure_next_item_index(p) in set inds p!
        --       I think that's always true
        let procItem = p(procedure_next_item_index(p)),
                --@LF here you can't tell whether this will be true? i.e. procItem.procedure in set dom a.items?
            item = a.items(procItem.procedure) in

            --TODO need to be able to check for different types of Items
            procItem.check = item.object.position
    pre
        procedure_completed(p) = false
        --@LF perhaps add
        --and
        --p(procedure_next_item_index(p)).procedure in set dom a.items?
        ;

    --@doc Marks next item in procedure as complete
    mark_proc_item_complete: Procedure -> Procedure
    mark_proc_item_complete(p) ==
        let i = procedure_next_item_index(p), item = p(i) in
            p ++ {i |-> complete_item(item)}
        pre
            procedure_completed(p) = false;
```

```

	--@doc Completes an item in the procedure
	do_proc_item: ItemObject * ChecklistItem -> ItemAndChecklistItem
	do_proc_item(i, p) ==
		let objective = p.check,
			checkckItem = complete_item(p) in
			-- Checks if the item is in the objective desired by the checklist
			if check_item_in_position(i, objective) then
				mk_ItemAndChecklistItem(i, checkckItem)
			else
				mk_ItemAndChecklistItem(move_item(i, p.check), checkckItem)
	pre
		p.checked = false
	post
		-- Checks the item has been moved correctly
		check_item_in_position(RESULT.item, p.check);

	--@doc Completes a procedure step by step
	-- a = Aircraft
	complete_procedure: Aircraft -> Aircraft
	complete_procedure(a) ==
		let procedure = a.procedure in
			mk_Aircraft(
				a.items ++ { x.procedure |-> do_proc_item(a.items(x.procedure), x).item | x in seq procedure },
				[ complete_item(x) | x in seq procedure ]
			)
	pre
		not procedure_completed(a.procedure)
	post
		procedure_completed(RESULT.procedure);

	-- AIRCRAFT ITEMS
	--@doc Marks ChecklistItem as complete
	complete_item: ChecklistItem -> ChecklistItem
	complete_item(i) ==
		mk_ChecklistItem(i.procedure, i.type, i.check, true)
	pre
		i.checked = false;

	--@doc Moves any type of Item
	move_item: ItemObject * ItemState -> ItemObject
	move_item(i, s) ==
		-- if is_Switch(i) then (implement later)
			let switch: Switch = i.object in
				if check_switch_onoff(switch) and (s <> <MIDDLE>) and switch.middlePosition then
					 mk_ItemObject(i.type, move_switch(move_switch(switch, <MIDDLE>), s))
				else
					 mk_ItemObject(i.type, move_switch(switch, s))
	pre
		wf_item_itemstate(i, s)
		and not check_item_in_position(i, s);
		-- and wf_switch_move(i.object, s);

```

```
    --@doc Moves a specific switch in the aircraft
    move_switch: Switch * SwitchState -> Switch
    move_switch(i, s) ==
        mk_Switch(s, i.middlePosition)
    pre
        wf_switch_move(i, s)
    post
        RESULT.position = s;

    --@doc Checks if the switch is in the on or off position
    check_switch_onoff: Switch -> bool
    check_switch_onoff(s) ==
        let position = s.position in
            position = <OFF> or position = <ON>
    post
        -- Only one can be true at a time
        -- If the switch is in the middle position, then RESULT cannot be true
        -- If the switch is in the on/off position, then the RESULT will be true
        (s.position = <MIDDLE>) <> RESULT;

    --@doc Checks if the item is already in position for the desired state for that item
    check_item_in_position: ItemObject * ItemState -> bool
    check_item_in_position(i, s) ==
        -- if is_Switch(i) then (implement later)
            i.object.position = s
    pre
        wf_item_itemstate(i,s);

    --@doc Checks if the Item.object is the same type for the ItemState
    wf_item_itemstate: ItemObject * ItemState -> bool
    wf_item_itemstate(i, s) ==
        (is_Switch(i.object) and is_SwitchState(s) and i.type = <SWITCH>)
        --TODO check that the item has not already been completed before moving item
        --TODO add other types of Items
        ;

    --@doc Checks if the move of the Switch is a valid
    wf_switch_move: Switch * SwitchState -> bool
    wf_switch_move(i, s) ==
        -- Checks that the switch not already in the desired state
        i.position <> s and
        -- The switch has to move one at a time
        -- Reasoning for this is that some switches cannot be moved in one quick move
        if i.middlePosition = true then
            -- Checks moving the switch away from the middle position
            (i.position = <MIDDLE> and s <> <MIDDLE>)
            -- Checks moving the siwtch to the middle position
            <> (check_switch_onoff(i) = true and s = <MIDDLE>)
        else
            check_switch_onoff(i) and s <> <MIDDLE>;
```

```

end Checklist

/*
//@LF always a good idea to run "qc" on your model. Here is its output. PO 21 and 22 show a problem.
//@LF silly me, this was my encoding with the cases missing one pattern :-). I can see yours has no issues. Good.

> qc
PO #1, PROVABLE by finite types in 0.002s
PO #2, PROVABLE by finite types in 0.0s
PO #3, PROVABLE by finite types in 0.0s
PO #4, PROVABLE by finite types in 0.0s
PO #5, PROVABLE by finite types in 0.0s
PO #6, PROVABLE by finite types in 0.0s
PO #7, PROVABLE by finite types in 0.0s
PO #8, PROVABLE by finite types in 0.0s
PO #9, PROVABLE by finite types in 0.001s
PO #10, PROVABLE by finite types in 0.001s
PO #11, PROVABLE by direct (body is total) in 0.003s
PO #12, PROVABLE by witness s = mk_Switch(<MIDDLE>, true) in 0.001s
PO #13, PROVABLE by direct (body is total) in 0.001s
PO #14, PROVABLE by witness k = mk_Knob(1, [-2]) in 0.0s
PO #15, PROVABLE by direct (body is total) in 0.0s
PO #16, PROVABLE by witness t = 0 in 0.0s
PO #17, PROVABLE by direct (body is total) in 0.001s
PO #18, PROVABLE by witness t = mk_Throttle(0, 0) in 0.001s
PO #19, PROVABLE by direct (body is total) in 0.002s
PO #20, PROVABLE by witness i = mk_ItemObject(<KNOB>, mk_Knob(1, [-1])) in 0.002s
PO #21, FAILED in 0.002s: Counterexample: type = <BUTTON>, object = mk_Knob(1, [-1])
Causes Error 4004: No cases apply for <BUTTON> in 'Checklist' (formal/checklist.vdmsl) at line 119:13
----
ItemObject':␣total␣function␣obligation␣in␣'Checklist'␣(formal/checklist.vdmsl)␣at␣line␣118:13
(forall␣mk_ItemObject'(type, object):ItemObject'!␣&
␣␣is_(inv_ItemObject'(mk_ItemObject'!(type,␣object)),␣bool))

PO␣#22,␣FAILED␣by␣direct␣in␣0.005s:␣Counterexample:␣type␣=␣<BUTTON>
PO␣#23,␣PROVABLE␣by␣witness␣type␣=␣<KNOB>,␣object␣=␣mk_Knob(1,␣[-1])␣in␣0.002s
PO␣#24,␣PROVABLE␣by␣direct␣(body␣is␣total)␣in␣0.001s
PO␣#25,␣PROVABLE␣by␣witness␣i␣=␣mk_ItemAndChecklistItem(mk_ItemObject(<KNOB>,␣mk_Knob(1,␣[-1])),␣mk_ChecklistItem([],␣<KNOB>,␣<MIDDLE>,␣true))␣in␣0.001s
PO␣#26,␣MAYBE␣in␣0.003s
PO␣#27,␣MAYBE␣in␣0.003s
PO␣#28,␣MAYBE␣in␣0.002s
PO␣#29,␣PROVABLE␣by␣witness␣p␣=␣[mk_ChecklistItem([],␣<BUTTON>,␣<MIDDLE>,␣true)]␣in␣0.001s
PO␣#30,␣MAYBE␣in␣0.002s
PO␣#31,␣MAYBE␣in␣0.001s
PO␣#32,␣MAYBE␣in␣0.003s
PO␣#33,␣MAYBE␣in␣0.002s
PO␣#34,␣MAYBE␣in␣0.001s
```

```
PO␣#35,␣MAYBE␣in␣0.002s
PO␣#36,␣MAYBE␣in␣0.009s
PO␣#37,␣MAYBE␣in␣0.008s
PO␣#38,␣MAYBE␣in␣0.007s
PO␣#39,␣MAYBE␣in␣0.009s
PO␣#40,␣MAYBE␣in␣0.002s
PO␣#41,␣MAYBE␣in␣0.001s
PO␣#42,␣MAYBE␣in␣0.001s
PO␣#43,␣MAYBE␣in␣0.002s
PO␣#44,␣MAYBE␣in␣0.002s
PO␣#45,␣MAYBE␣in␣0.003s
PO␣#46,␣MAYBE␣in␣0.002s
PO␣#47,␣MAYBE␣in␣0.002s
PO␣#48,␣MAYBE␣in␣0.001s
PO␣#49,␣MAYBE␣in␣0.001s
PO␣#50,␣MAYBE␣in␣0.0s
PO␣#51,␣MAYBE␣in␣0.0s
PO␣#52,␣MAYBE␣in␣0.005s
PO␣#53,␣PROVABLE␣by␣trivial␣p␣in␣set␣(dom␣checklist)␣in␣0.001s
PO␣#54,␣MAYBE␣in␣0.006s
PO␣#55,␣MAYBE␣in␣0.0s
PO␣#56,␣MAYBE␣in␣0.001s
PO␣#57,␣MAYBE␣in␣0.001s
PO␣#58,␣MAYBE␣in␣0.001s
PO␣#59,␣MAYBE␣in␣0.001s
PO␣#60,␣MAYBE␣in␣0.001s
PO␣#61,␣MAYBE␣in␣0.001s
PO␣#62,␣MAYBE␣in␣0.0s
PO␣#63,␣PROVABLE␣by␣finite␣types␣in␣0.001s
PO␣#64,␣PROVABLE␣by␣finite␣types␣in␣0.001s
PO␣#65,␣PROVABLE␣by␣finite␣types␣in␣0.001s
PO␣#66,␣MAYBE␣in␣0.001s
>
*/
```

References

[1] Immanuel Barshi, Robert Mauro, Asaf Degani et al. *Designing Flightdeck Procedures*. eng. Ames Research Center, Nov. 2016. URL: https://ntrs.nasa.gov/citations/20160013263.

[2] Atul Gawande. *The Checklist Manifesto: How To Get Things Right*. Main Edition. Profile Books, July 2010. ISBN: 9781846683145.

[3] National Tranportation Safety Board. *Loss of Thrust in Both Engines After Encountering a Flock of Birds and Subsequent Ditching on the Hudson River*. Technical Report PB2010-910403. May 2010. URL: https://www.ntsb.gov/investigations/Pages/DCA09MA026.aspx.

[4] William R. Knecht and Michael Lenz. *Causes of General Aviation Weather-Related, Non-Fatal Incidents: Analysis Using NASA Aviation Safety Reporting System Data*. Tech. rep. DOT/FAA/AM-10/13. FAA Office of Aerospace Medicine Civil Aerospace Medical Institute, Sept. 2010.

[5] Nick Battle. *VDMJ*. URL: https://github.com/nickbattle/vdmj (visited on 21/04/2024).

[6] Peter Gorm Larsen, Kenneth Lausdahl, Peter Jørgensen et al. *Overture VDM-10 Tool Support: User Guide*. TR-2010-02. Apr. 2013. Chap. 16, pp. 81–98. URL: https://raw.github.com/overturetool/documentation/editing/documentation/UserGuideOvertureIDE/OvertureIDEUserGuide.pdf.

[7] Kyushu University. *The VDM Toolbox API*. Version 1.0. 2016. URL: https://github.com/vdmtools/vdmtools/raw/stable/doc/api-man/ApiMan_a4E.pdf.

[8] Raoul-Gabriel Urma. 'Alternative Languages for the JVM'. In: *Java Magazine* (July 2014). URL: https://www.oracle.com/technical-resources/articles/java/architect-languages.html (visited on 05/05/2024).

[9] JetBrains s.r.o. *Kotlin Programming Language*. URL: https://kotlinlang.org/ (visited on 21/04/2024).

[10] Google LLC. *Kotlin and Android | Android Developers*. URL: https://developer.android.com/kotlin (visited on 21/04/2024).

[11] OpenJFX. *JavaFX*. URL: https://openjfx.io/ (visited on 21/04/2024).

[12] FormDev Software GmbH. *FlatLaf - Flat Look and Feel | FormDev*. URL: https://www.formdev.com/flatlaf/ (visited on 21/04/2024).

[13] JetBrains s.r.o. *Compose Multiplatform UI Framework | JetBrains | JetBrains: Developer Tools for Professionals and Teams*. URL: https://www.jetbrains.com/lp/compose-multiplatform/ (visited on 21/04/2024).

[14] Google LLC. *Flutter - Build apps for any screen*. URL: https://flutter.dev/ (visited on 21/04/2024).

[15] Laminar Research. *X-Plane | The world's most advanced flight simulator*. URL: https://www.x-plane.com/ (visited on 21/04/2024).

[16] Lockheed Martin Corporation. *Prepar3D – Next Level Training. World class simulation. Be ahead of ready with Prepar3D*. URL: https://www.prepar3d.com/ (visited on 21/04/2024).

[17] NASA Ames Research Center Diagnostics and Prognostics Group. *X-Plane Connect*. URL: https://github.com/nasa/XPlaneConnect (visited on 21/04/2024).

[18] Google LLC. *Lists – Material Design 3*. URL: https://m3.material.io/components/lists/guidelines (visited on 13/05/2024).

[19] Google LLC. *Top app bar – Material Design 3*. URL: https://m3.material.io/components/top-app-bar/guidelines (visited on 13/05/2024).

[20] Adriel Café. *Overview | Voyager*. URL: https://voyager.adriel.cafe/ (visited on 13/05/2024).

[21] Koin and Kotzilla. *Koin - The pragmatic Kotlin Injection Framework - developed by Kotzilla and its open-source contributors*. URL: https://insert-koin.io/ (visited on 13/05/2024).

[22] Adriel Café. *Koin integration | Voyager*. URL: https://voyager.adriel.cafe/screenmodel/koin-integration (visited on 13/05/2024).

[23] Square, Inc. *Overview - SQLDelight*. Version 2.0.2. URL: https://cashapp.github.io/sqldelight/2.0.2/ (visited on 14/05/2024).

[24] Hipp, Wyrick & Company, Inc. *About SQLite*. URL: https://www.sqlite.org/about.html (visited on 14/05/2024).

[25] Docker Inc. *What is a Container? | Docker*. URL: https://www.docker.com/resources/what-container/ (visited on 14/05/2024).

[26] Hipp, Wyrick & Company, Inc. *How SQLite Is Tested*. URL: https://www.sqlite.org/testing.html (visited on 14/05/2024).

[27] Mike Frizzell. *Maven Folder Structure Re-org by frizman21 · Pull Request #227 · nasa/X-PlaneConnect*. URL: https://github.com/nasa/XPlaneConnect/pull/227 (visited on 13/05/2024).

[28] Jason Watkins. *Publish Java library to maven repo · Issue #223 · nasa/XPlaneConnect - Comment*. URL: https://github.com/nasa/XPlaneConnect/issues/223#issuecomment-870819396 (visited on 13/05/2024).

[29] JitPack. *JitPack | Publish JVM and Android libraries*. URL: https://jitpack.io/ (visited on 13/05/2024).

[30] Gradle Inc. *gitRepository*. URL: https://docs.gradle.org/current/kotlin-dsl/gradle/org.gradle.vcs/-source-control/git-repository.html (visited on 13/05/2024).

[31] Jendrik Johannes. *Git repository at <url> did not contain a project publishing the specified dependency*. URL: https://discuss.gradle.org/t/git-repository-at-url-did-not-contain-a-project-publishing-the-specified-dependency/34019/2 (visited on 13/05/2024).

[32] Gradle Inc. *Migrating Builds From Apache Maven*. Version 8.7. 2023. URL: https://docs.gradle.org/current/userguide/migrating_from_maven.html#migmvn:automatic_conversion.

[33] Barbara Burian. 'Design Guidance for Emergency and Abnormal Checklists in Aviation'. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 50 (Oct. 2006). DOI: 10.1177/154193120605000123.

[34] Quinn Kennedy, Joy Taylor, Daniel Heraldez et al. 'Intraindividual Variability in Basic Reaction Time Predicts Middle-Aged and Older Pilots' Flight Simulator Performance'. In: *The Journals of Gerontology: Series B* 68.4 (Oct. 2012), pp. 487–494. ISSN: 1079-5014. DOI: 10.1093/geronb/gbs090. eprint: https://academic.oup.com/psychsocgerontology/article-pdf/68/4/487/1520662/gbs090.pdf.

[35] Civil Aviation Authority. *Aircraft Emergencies: Considerations for air traffic controllers*. CAP745. Mar. 2005. URL: https://www.caa.co.uk/cap745.

[36] The Overture Project. *The Vienna Development Method*. URL: https://www.overturetool.org/method/ (visited on 23/02/2024).

[37] Google LLC. *Jetpack Compose UI App Development Toolkit - Android Developers*. URL: https://developer.android.com/develop/ui/compose (visited on 21/04/2024).