Testing Quick Reference Handbooks in Flight Simulators

Anthony Berg (200871682)
Supervisor: Dr Leo Freitas

Word Count: 3074

15th May 2024

Preface

Abstract

This is an abstract.

Declaration

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

Acknowledgements

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 section 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

Testing

- Since VDMJ version 4.5, it provides the QuickCheck tool

- This allows for providing counter examples to the model

- The counter examples were helpful to create pre and post conditions to avoid any unexpected results from the model

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
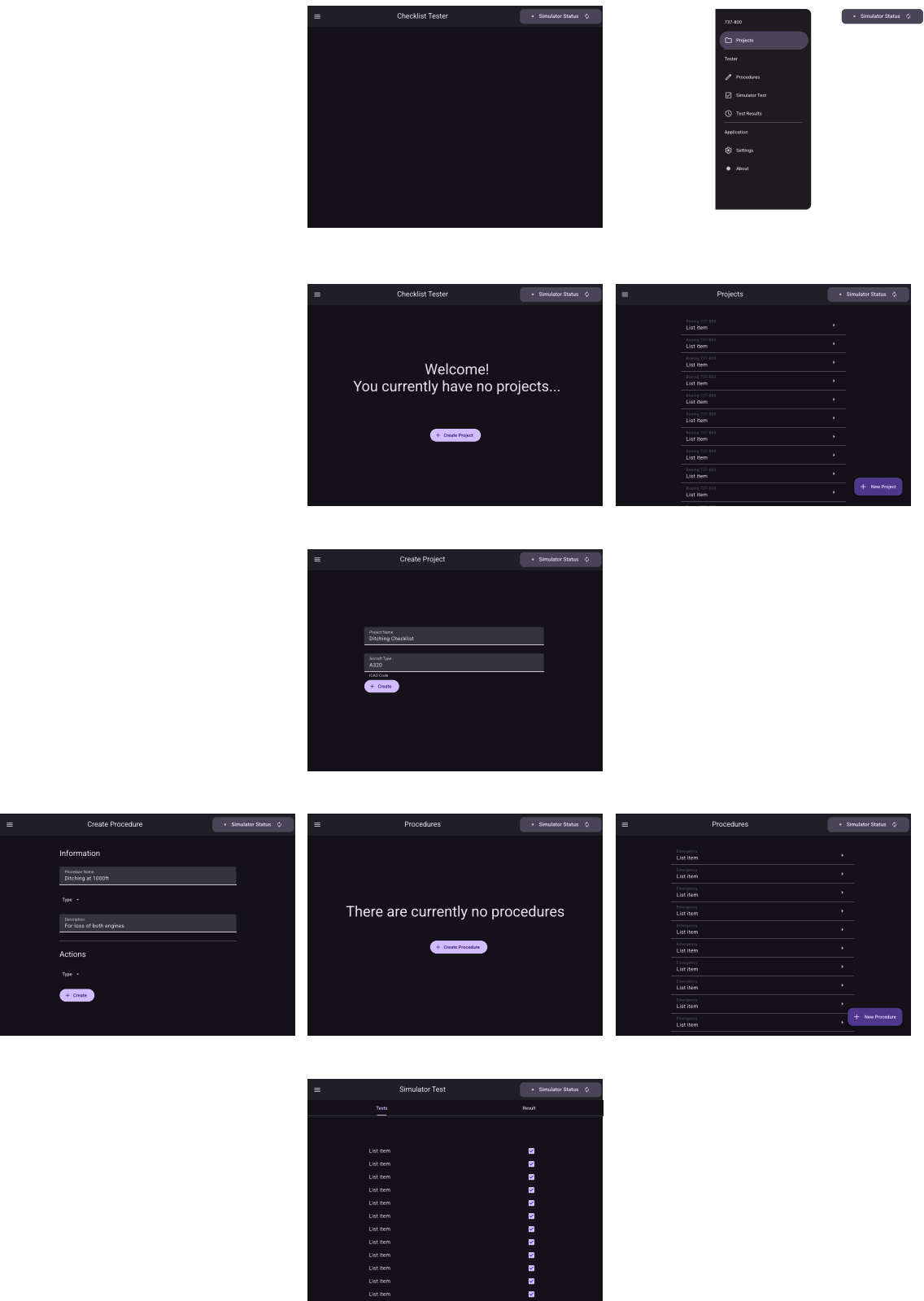
Figure 3.2: Design for the Checklist Connector GUI in Figma

- 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

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

```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()
    }
}
```

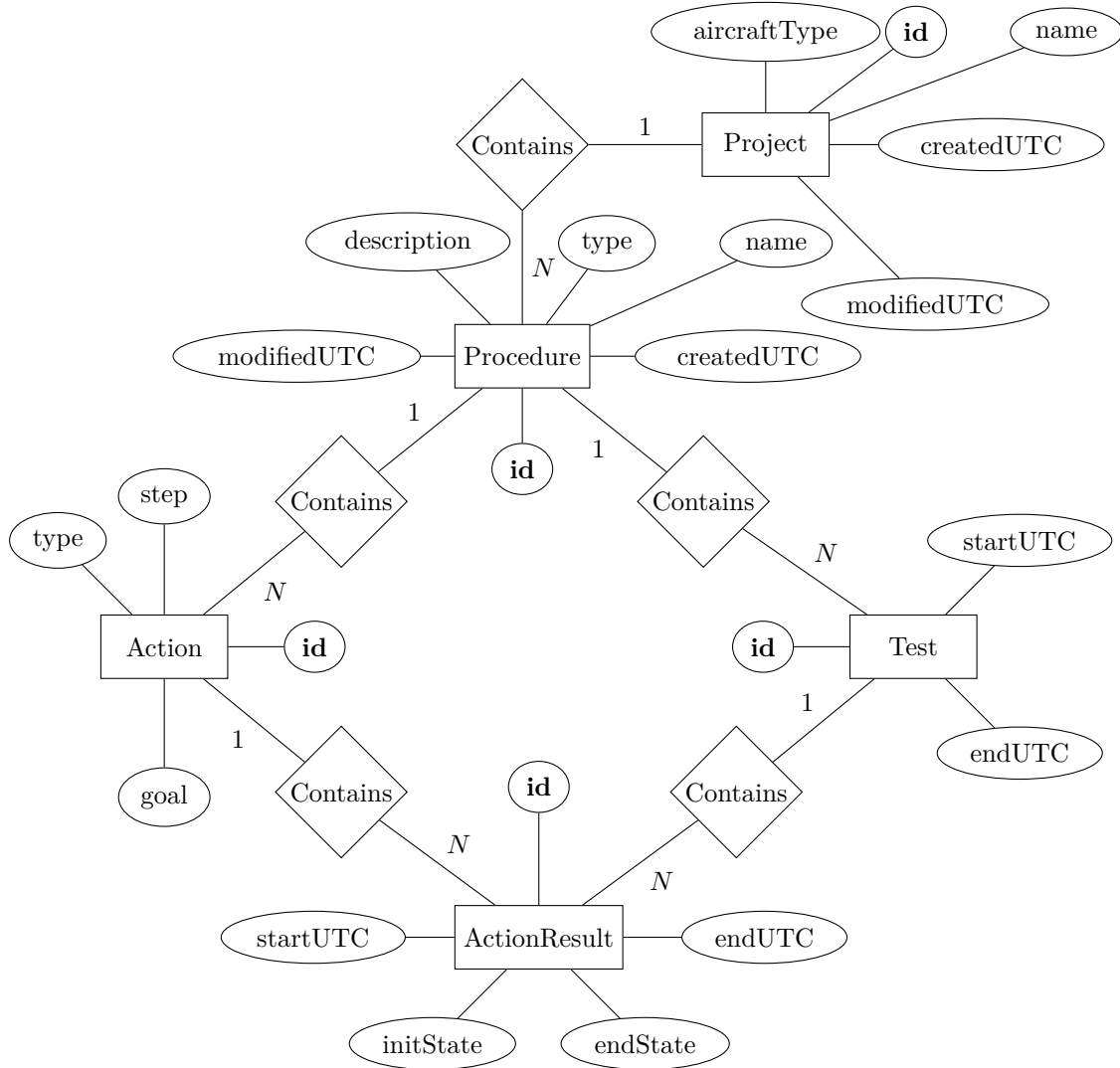Listing 3.1: Example of modular code in Compose

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

3.3.4 VDMJ Wrapper

- VDMJ is written in Java and it is free open source software that is accessible on GitHub

- This allows for VDMJ to be used as per the license, GNU General Public License v3 (GPLv3) [27] [28]. This means that as VDMJ is being used as a library, the code for this project has to be licensed with GPLv3 or any GPLv3 compatible license [29]

Implementing VDMJ

- VDMJ has packages available on Maven Central making adding it as a dependency simple

- The package used was `dk.au.ece.vdmj:vdmj` with version `4.5.0`

- However, initially when implementing VDMJ, `4.5.0-P` was used accidentally and it led to debugging why imports were not working; and therefore the `-P` versions are not suitable

- The initial method of implementation was using a Ktor server that would have run alongside the desktop application, where the server would handle Representational State Transfer (REST) API calls

- This was unnecessary as the *interactive* mode of VDMJ was able to run on the desktop application itself. However, the Ktor was useful for debugging and testing as an API route was created to allow VDMJ commands to be executed through a URL

- To be able to get the outputs from VDMJ, a `ConsolePrintWriter` new had to be created from the `com.fujitsu.vdmj.messages` package; which handles writing to the console *stdout*. This then gets used to replace the `Console.out` and `Console.err` in the `com.fujitsu.vdmj→ .messages` package

- Parsing commands into VDMJ interface - was more difficult [1]

  - Created a `PipedInputStream` object, that gets connected to a `PipedOutputStream→` object by passing the latter object in as a parameter. The `PipedOutputStream` is then used to pass inputs into `PipedInputStream`

  - To be able to write to this stream, a `BufferedWriter` is created by passing the `PipedOutputStream→` with a bridge `OutputStreamWriter` that encodes characters into bytes

  - For VDMJ to be able to read the input stream, a separate object had to be created, `BufferedReader`, where the `PipedInputStream` gets parsed through a bridge, `InputStreamReader` that converts bytes to characters

Handling VDMJ Outputs

- VDMJ outputs are handled using string manipulation

- Created into objects that are replicas of types in VDM-SL

- The string manipulation allows to specify where the outputs of the object go

3.3.5 Connecting to the Flight Simulator

3.3.6 Testing

- Testing can be run with Gradle when it comes to running unit tests

- Decided to use JUnit 5 as it provides additional tools such as statistics, integration with IntelliJ to view code coverage, or being run in continuous integration tests

- The testable components in this project is mostly backend modules as the GUI made in Compose is not the focus of the project and it would require a lot of extra time

- Unit tests have been made for the database and Koin

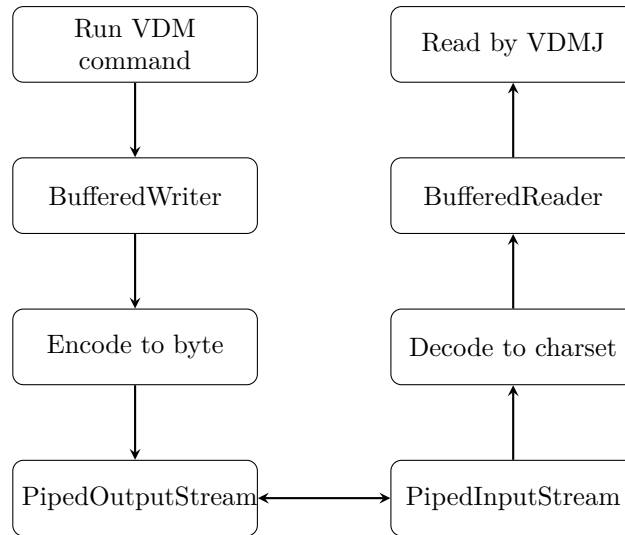[1]The objects created here are provided by the `java.io` package.

Figure 3.4: Flowchart of VDMJ Input/Output Stream handling

- Koin comes with tests that can be automatically be generated

- Ethos when testing was to try and find exploits, act as a user who may mishandle inputs, and stress testing functions by passing parameter with hundreds of objects

Testing for Resource Usage

- The application was tested using the *Profiler* tool on IntelliJ IDEA 2024 (Ultimate Edition) to find potential memory leaks

- One problem found was the initial VDMJ wrapper which would use the execute command instead of the interpreter, which would require reinitializing the entirety of VDMJ, which resulted in a slight memory leak and a massive write usage

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 [30]. 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 [31]

- Therefore, had to find an alternative way to implement

- Jitpack [32]

  - 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 [33]

  - 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 [34]

- 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 [35]

- 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

- Adding the Gradle build tools can be seen as being helpful for others, as it would allow for the XPC library to be added as a dependency, especially if the NASA Ames Research Center Diagnostics and Prognostics Group were to add it to the GitHub repository, it would mean that it would be easier for people to access Maven Packages for XPC

- Therefore, to help improve the experience for other people who would want to develop with the XPC Java library, it would be logical to submit a pull request

- But it did mean making sure that the contribution would be perfect and not contain problems

Testing

- The XPC Java library includes a JUnit 4 test, however, implementing this with Gradle proved useless, as it was not able to get the results from the tests, which would be bad for not being able to catch problems with new builds

- Therefore, the tests were updated to JUnit 5, where most of the changes were adding asserts for throws [36] [2]

GitHub

- Made sure to add generated build files to .gitignore

- Changed the URL of the repository in Gradle to NASA's repository so that the Maven package can be published correctly on the GitHub repository

- From the beginning anyways, made sure to have insightful commit messages

- Submitted the pull request stating the changes made[3]

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

---

[2]The commit including the changes to the tests can be viewed here: https://github.com/smyalygames/XPlaneConnect/commit/e7b8d1e811999b4f8d7230f60ba94368e14f1148

[3]https://github.com/nasa/XPlaneConnect/pull/313

Chapter 4

Results

4.1   Problems Found

4.2   LOC?

4.3   Reflection
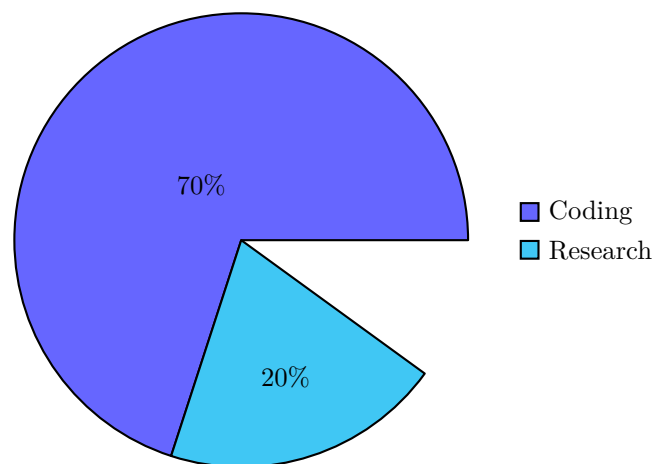
4.4   Time Spent

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

```
1  module Checklist
2  exports all
3  definitions
4
5  values
6      -- Before Start Checklist
7      -- Items in Aircraft
8      -- Flight Deck... (can't check)
9      fuel: ItemObject = mk_ItemObject(<SWITCH>, mk_Switch(<OFF>, false)→
           );
10     pax_sign: ItemObject = mk_ItemObject(<SWITCH>, mk_Switch(<OFF>, →
           true));
11     windows: ItemObject = mk_ItemObject(<SWITCH>, mk_Switch(<ON>, →
           false));
12     -- Preflight steps
13     acol: ItemObject = mk_ItemObject(<SWITCH>, mk_Switch(<OFF>, false)→
           );
14
15     aircraft_panels: Items = {"Fuel␣Pump" |-> fuel, "Passenger␣Signs" →
           |-> pax_sign, "Windows" |-> windows, "Anti␣Collision␣Lights" →
           |-> acol};
16
17     -- Checklist
18     -- Flight Deck... (can't check)
19     fuel_chkl: ChecklistItem = mk_ChecklistItem("Fuel␣Pump", <SWITCH>,→
            <ON>, false);
20     pax_sign_chkl: ChecklistItem = mk_ChecklistItem("Passenger␣Signs",→
            <SWITCH>, <ON>, false);
21     windows_chkl: ChecklistItem = mk_ChecklistItem("Windows", <SWITCH→
           >, <ON>, false);
22     -- Preflight steps
23     acol_chkl: ChecklistItem = mk_ChecklistItem("Anti␣Collision␣Lights→
           ", <SWITCH>, <ON>, false);
24
25     before_start_procedure: Procedure = [fuel_chkl, pax_sign_chkl, →
           windows_chkl, acol_chkl];
26
27     aircraft = mk_Aircraft(aircraft_panels, before_start_procedure);
28  types
29     --@doc The dataref name in X-Plane
30     Dataref = seq1 of char;
31
32     -- Aircraft
```

```
33
34      -- Switches
35      --@doc The state a switch can be in
36      SwitchState = <OFF> | <MIDDLE> | <ON>;
37
38          --@LF why have a type kist as a rename?
39      ItemState = SwitchState; --@TODO | Button | ...
40
41      --@doc A switch, with the possible states it can be in, and the →
            state that it is in
42      Switch ::
43          position : SwitchState
44          middlePosition : bool
45          inv s ==
46              (s.position = <MIDDLE> => s.middlePosition);
47
48      -- Knob
49      Knob ::
50          position : nat
51          --@LF how can a state be an int? perhaps a proper type (i..e. →
                subset of int range or a union?)
52          states : set1 of nat
53          inv k ==
54              k.position in set k.states;
55
56      Lever = nat
57          inv t == t <= 100;
58
59      Throttle ::
60          thrust: Lever
61          reverser: Lever
62          inv t ==
63              (t.reverser > 0 <=> t.thrust = 0);
64
65      --@doc The type that the action of the button is
66      ItemType = <SWITCH> | <KNOB> | <BUTTON> | <THROTTLE>;
67
68      --@doc The unique switch/knob/etc of that aircraft
69      ObjectType = Switch | Knob | Throttle;
70      ItemObject ::
71          type : ItemType
72          object : ObjectType
73          inv mk_ItemObject(type, object) ==
74                  cases type:
75                          <SWITCH> -> is_Switch(object),
76                          <KNOB>   -> is_Knob(object),
77                          <THROTTLE>-> is_Throttle(object),
78                          --<BUTTON> -> true
79                          others -> true
80                  end;
81
82      --@doc Contains each ItemObject in the Aircraft, e.g. Fuel Pump →
            switch
83      Items = map Dataref to ItemObject;
84
85      --@doc Contains the panels (all the items in the aircraft) and the→
             procedure
86      Aircraft ::
```

```
 87            items : Items
 88            procedure : Procedure
 89            inv mk_Aircraft(i, p) ==
 90            ({ x.procedure | x in seq p } subset dom i);
 91
 92        -- Checklist
 93
 94        --@doc Item of a checklist, e.g. Landing gear down
 95        ChecklistItem ::
 96            --@LF again, empty string here doesn't make sense.
 97            procedure : Dataref
 98            type : ItemType
 99            --TODO Check is not only SwitchState
100            check : SwitchState
101            checked : bool;
102
103        --@doc This is an item in the aircraft that complements the item →
                 in the procedure
104        ItemAndChecklistItem ::
105            item : ItemObject
106            checklistItem: ChecklistItem
107            inv i == i.item.type = i.checklistItem.type;
108
109        --@doc A section of a checklist, e.g. Landing Checklist
110        --@LF shouldn't this be non-empty? What's the point to map a →
                 checklist name to an empty procedure? Yes.
111        Procedure = seq1 of ChecklistItem
112            inv p ==
113                --@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).
114                --    I presume this was intended.
115                false not in set {
116                    let first = p(x-1).checked, second = p(x).checked in
117                            --@LF boolean values don't need equality check
118                        second => first--((first = true) and (second = →
                                false))
119                    | x in set {2,...,len p}};
120
121  functions
122      -- PROCEDURES
123      --@doc Finds the index of the next item in the procedure that →
               needs to be completed
124      procedure_next_item_index: Procedure -> nat1
125      procedure_next_item_index(p) ==
126          hd [ x | x in set {1,...,len p} & not p(x).checked ]--p(x).→
                 checked = false]
127      pre
128          -- Checks procedure has not already been completed
129          not procedure_completed(p)--procedure_completed(p) = false
130      post
131          -- Checks that the index of the item is the next one to be →
                 completed
132          --@LF your def is quite confusing (to me)
133          --@LF how do you know that RESULT in inds p? Well, the →
                 definition above okay.
134          --    but you can't know whether p(RESULT-1) will! What if →
                 RESULT=1? p(RESULT-1)=p(0) which is invalid!
```

```
135              (not p(RESULT).checked)
136              and
137              (RESULT > 1 => p(RESULT-1).checked)
138              --p(RESULT).checked = false
139              --and if RESULT > 1 then
140              --     p(RESULT-1).checked = true
141              --else
142              --     true
143              ;
144
145      -- --@doc Checks if all the procedures have been completed
146      -- check_all_proc_completed: Checklist -> bool
147      -- check_all_proc_completed(c) ==
148      --     false not in set { procedure_completed(c(x)) | x in set →
                 {1,...,len c} };
149
150      -- --@doc Gives the index for the next procedure to complete
151      -- next_procedure: Checklist -> nat1
152      -- next_procedure(c) ==
153      --     hd [ x | x in set {1,...,len c} & not procedure_completed(c→
                 (x))]
154      -- post
155      --     RESULT <= len c;
156
157      --@doc Checks if the procedure has been completed
158      procedure_completed: Procedure -> bool
159      procedure_completed(p) ==
160          false not in set { p(x).checked | x in set {1,...,len p} };
161
162      --@doc Checks if the next item in the procedure has been completed
163      check_proc_item_complete: Procedure * Aircraft -> bool
164      check_proc_item_complete(p, a) ==
165          --@LF here you have a nice lemma to prove: →
                 procedure_next_item_index(p) in set inds p!
166          --        I think that's always true
167          let procItem = p(procedure_next_item_index(p)),
168                  --@LF here you can't tell whether this will be true? i→
                        .e. procItem.procedure in set dom a.items?
169              item = a.items(procItem.procedure) in
170
171              --TODO need to be able to check for different types of →
                    Items
172              procItem.check = item.object.position
173      pre
174          procedure_completed(p) = false
175          --@LF perhaps add
176          --and
177          --p(procedure_next_item_index(p)).procedure in set dom a.items→
                 ?
178          ;
179
180      --@doc Marks next item in procedure as complete
181      mark_proc_item_complete: Procedure -> Procedure
182      mark_proc_item_complete(p) ==
183          let i = procedure_next_item_index(p), item = p(i) in
184              p ++ {i |-> complete_item(item)}
185          pre
186              procedure_completed(p) = false;
```

```
187
188        --@doc Completes an item in the procedure
189        do_proc_item: ItemObject * ChecklistItem -> ItemAndChecklistItem
190        do_proc_item(i, p) ==
191            let objective = p.check,
192                checckItem = complete_item(p) in
193                -- Checks if the item is in the objective desired by the →
                       checklist
194                if check_item_in_position(i, objective) then
195                    mk_ItemAndChecklistItem(i, checckItem)
196                else
197                    mk_ItemAndChecklistItem(move_item(i, p.check), →
                           checckItem)
198        pre
199            p.checked = false
200        post
201            -- Checks the item has been moved correctly
202            check_item_in_position(RESULT.item, p.check);
203
204        --@doc Completes a procedure step by step
205        -- a = Aircraft
206        complete_procedure: Aircraft -> Aircraft
207        complete_procedure(a) ==
208            let procedure = a.procedure in
209                mk_Aircraft(
210                    a.items ++ { x.procedure |-> do_proc_item(a.items(x.→
                           procedure), x).item | x in seq procedure },
211                    [ complete_item(x) | x in seq procedure ]
212                )
213        pre
214            not procedure_completed(a.procedure)
215        post
216            procedure_completed(RESULT.procedure);
217
218        -- AIRCRAFT ITEMS
219        --@doc Marks ChecklistItem as complete
220        complete_item: ChecklistItem -> ChecklistItem
221        complete_item(i) ==
222            mk_ChecklistItem(i.procedure, i.type, i.check, true)
223        pre
224            i.checked = false;
225
226        --@doc Moves any type of Item
227        move_item: ItemObject * ItemState -> ItemObject
228        move_item(i, s) ==
229            -- if is_Switch(i) then (implement later)
230                let switch: Switch = i.object in
231                    if check_switch_onoff(switch) and (s <> <MIDDLE>) and →
                           switch.middlePosition then
232                        mk_ItemObject(i.type, move_switch(move_switch(→
                               switch, <MIDDLE>), s))
233                    else
234                        mk_ItemObject(i.type, move_switch(switch, s))
235        pre
236            wf_item_itemstate(i, s)
237            and not check_item_in_position(i, s);
238            -- and wf_switch_move(i.object, s);
239
```

```
240      --@doc Moves a specific switch in the aircraft
241      move_switch: Switch * SwitchState -> Switch
242      move_switch(i, s) ==
243          mk_Switch(s, i.middlePosition)
244      pre
245          wf_switch_move(i, s)
246      post
247          RESULT.position = s;
248
249      --@doc Checks if the switch is in the on or off position
250      check_switch_onoff: Switch -> bool
251      check_switch_onoff(s) ==
252          let position = s.position in
253              position = <OFF> or position = <ON>
254      post
255          -- Only one can be true at a time
256          -- If the switch is in the middle position, then RESULT cannot↪
                be true
257          -- If the switch is in the on/off position, then the RESULT ↪
                will be true
258          (s.position = <MIDDLE>) <> RESULT;
259
260      --@doc Checks if the item is already in position for the desired ↪
            state for that item
261      check_item_in_position: ItemObject * ItemState -> bool
262      check_item_in_position(i, s) ==
263          -- if is_Switch(i) then (implement later)
264              i.object.position = s
265      pre
266          wf_item_itemstate(i,s);
267
268      --@doc Checks if the Item.object is the same type for the ↪
            ItemState
269      wf_item_itemstate: ItemObject * ItemState -> bool
270      wf_item_itemstate(i, s) ==
271          (is_Switch(i.object) and is_SwitchState(s) and i.type = <↪
                SWITCH>)
272          --TODO check that the item has not already been completed ↪
                before moving item
273          --TODO add other types of Items
274          ;
275
276      --@doc Checks if the move of the Switch is a valid
277      wf_switch_move: Switch * SwitchState -> bool
278      wf_switch_move(i, s) ==
279          -- Checks that the switch not already in the desired state
280          i.position <> s and
281          -- The switch has to move one at a time
282          -- Reasoning for this is that some switches cannot be moved in↪
                 one quick move
283          if i.middlePosition = true then
284              -- Checks moving the switch away from the middle position
285              (i.position = <MIDDLE> and s <> <MIDDLE>)
286              -- Checks moving the siwtch to the middle position
287              <> (check_switch_onoff(i) = true and s = <MIDDLE>)
288          else
289              check_switch_onoff(i) and s <> <MIDDLE>;
290
```

```
291
292  end Checklist
293
294  /*
295  //@LF always a good idea to run "qc" on your model. Here is its output→
         . PO 21 and 22 show a problem.
296  //@LF silly me, this was my encoding with the cases missing one →
         pattern :-). I can see yours has no issues. Good.
297
298  > qc
299  PO #1, PROVABLE by finite types in 0.002s
300  PO #2, PROVABLE by finite types in 0.0s
301  PO #3, PROVABLE by finite types in 0.0s
302  PO #4, PROVABLE by finite types in 0.0s
303  PO #5, PROVABLE by finite types in 0.0s
304  PO #6, PROVABLE by finite types in 0.0s
305  PO #7, PROVABLE by finite types in 0.0s
306  PO #8, PROVABLE by finite types in 0.0s
307  PO #9, PROVABLE by finite types in 0.001s
308  PO #10, PROVABLE by finite types in 0.001s
309  PO #11, PROVABLE by direct (body is total) in 0.003s
310  PO #12, PROVABLE by witness s = mk_Switch(<MIDDLE>, true) in 0.001s
311  PO #13, PROVABLE by direct (body is total) in 0.001s
312  PO #14, PROVABLE by witness k = mk_Knob(1, [-2]) in 0.0s
313  PO #15, PROVABLE by direct (body is total) in 0.0s
314  PO #16, PROVABLE by witness t = 0 in 0.0s
315  PO #17, PROVABLE by direct (body is total) in 0.001s
316  PO #18, PROVABLE by witness t = mk_Throttle(0, 0) in 0.001s
317  PO #19, PROVABLE by direct (body is total) in 0.002s
318  PO #20, PROVABLE by witness i = mk_ItemObject(<KNOB>, mk_Knob(1, [-1])→
         ) in 0.002s
319  PO #21, FAILED in 0.002s: Counterexample: type = <BUTTON>, object = →
         mk_Knob(1, [-1])
320  Causes Error 4004: No cases apply for <BUTTON> in 'Checklist' (formal/→
         checklist.vdmsl) at line 119:13
321  ----
322  ItemObject':␣total␣function␣obligation␣in␣'Checklist'␣(formal/→
         checklist.vdmsl)␣at␣line␣118:13
323  (forall␣mk_ItemObject'(type, object):ItemObject'!␣&
324  ␣␣is_(inv_ItemObject'(mk_ItemObject'!(type,␣object)),␣bool))
325
326  PO␣#22,␣FAILED␣by␣direct␣in␣0.005s:␣Counterexample:␣type␣=␣<BUTTON>
327  PO␣#23,␣PROVABLE␣by␣witness␣type␣=␣<KNOB>,␣object␣=␣mk_Knob(1,␣[-1])␣→
         in␣0.002s
328  PO␣#24,␣PROVABLE␣by␣direct␣(body␣is␣total)␣in␣0.001s
329  PO␣#25,␣PROVABLE␣by␣witness␣i␣=␣mk_ItemAndChecklistItem(mk_ItemObject→
         (<KNOB>,␣mk_Knob(1,␣[-1])),␣mk_ChecklistItem([],␣<KNOB>,␣<MIDDLE>,→
         ␣true))␣in␣0.001s
330  PO␣#26,␣MAYBE␣in␣0.003s
331  PO␣#27,␣MAYBE␣in␣0.003s
332  PO␣#28,␣MAYBE␣in␣0.002s
333  PO␣#29,␣PROVABLE␣by␣witness␣p␣=␣[mk_ChecklistItem([],␣<BUTTON>,␣<→
         MIDDLE>,␣true)]␣in␣0.001s
334  PO␣#30,␣MAYBE␣in␣0.002s
335  PO␣#31,␣MAYBE␣in␣0.001s
336  PO␣#32,␣MAYBE␣in␣0.003s
337  PO␣#33,␣MAYBE␣in␣0.002s
338  PO␣#34,␣MAYBE␣in␣0.001s
```

```
339   PO␣#35,␣MAYBE␣in␣0.002s
340   PO␣#36,␣MAYBE␣in␣0.009s
341   PO␣#37,␣MAYBE␣in␣0.008s
342   PO␣#38,␣MAYBE␣in␣0.007s
343   PO␣#39,␣MAYBE␣in␣0.009s
344   PO␣#40,␣MAYBE␣in␣0.002s
345   PO␣#41,␣MAYBE␣in␣0.001s
346   PO␣#42,␣MAYBE␣in␣0.001s
347   PO␣#43,␣MAYBE␣in␣0.002s
348   PO␣#44,␣MAYBE␣in␣0.002s
349   PO␣#45,␣MAYBE␣in␣0.003s
350   PO␣#46,␣MAYBE␣in␣0.002s
351   PO␣#47,␣MAYBE␣in␣0.002s
352   PO␣#48,␣MAYBE␣in␣0.001s
353   PO␣#49,␣MAYBE␣in␣0.001s
354   PO␣#50,␣MAYBE␣in␣0.0s
355   PO␣#51,␣MAYBE␣in␣0.0s
356   PO␣#52,␣MAYBE␣in␣0.005s
357   PO␣#53,␣PROVABLE␣by␣trivial␣p␣in␣set␣(dom␣checklist)␣in␣0.001s
358   PO␣#54,␣MAYBE␣in␣0.006s
359   PO␣#55,␣MAYBE␣in␣0.0s
360   PO␣#56,␣MAYBE␣in␣0.001s
361   PO␣#57,␣MAYBE␣in␣0.001s
362   PO␣#58,␣MAYBE␣in␣0.001s
363   PO␣#59,␣MAYBE␣in␣0.001s
364   PO␣#60,␣MAYBE␣in␣0.001s
365   PO␣#61,␣MAYBE␣in␣0.001s
366   PO␣#62,␣MAYBE␣in␣0.0s
367   PO␣#63,␣PROVABLE␣by␣finite␣types␣in␣0.001s
368   PO␣#64,␣PROVABLE␣by␣finite␣types␣in␣0.001s
369   PO␣#65,␣PROVABLE␣by␣finite␣types␣in␣0.001s
370   PO␣#66,␣MAYBE␣in␣0.001s
371   >
372   */
```

Appendix B

Database

B.1 SQL Schemas

```sql
1  CREATE TABLE IF NOT EXISTS Project (
2      id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
3      name TEXT NOT NULL,
4      aircraftType TEXT NOT NULL,
5      createdUTC TEXT NOT NULL,
6      modifiedUTC TEXT
7  );
8
9  createProject:
10 INSERT INTO Project(name, aircraftType, createdUTC)
11 VALUES (?, ?, ?);
12
13 selectAllProjects:
14 SELECT * FROM Project;
15
16 selectProjectById:
17 SELECT * FROM Project
18 WHERE id = ?;
19
20 countProjects:
21 SELECT COUNT(*) FROM Project;
```

Listing B.2: SQL Schema for Project

```
1   CREATE TABLE IF NOT EXISTS Procedure (
2       id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
3       projectId INTEGER NOT NULL,
4       name TEXT NOT NULL,
5       type TEXT NOT NULL,
6       description TEXT NOT NULL,
7       createdUTC TEXT NOT NULL,
8       modifiedUTC TEXT,
9       FOREIGN KEY (projectId) REFERENCES Project(id)
10  );
11
12  createProcedure:
13  INSERT INTO Procedure(projectId, name, type, description, createdUTC)
14  VALUES (?, ?, ?, ?, ?);
15
16  selectProcedures:
17  SELECT * FROM Procedure
18  WHERE projectId = ?;
19
20  selectProcedureById:
21  SELECT * FROM Procedure
22  WHERE id = ?;
23
24  countProcedures:
25  SELECT COUNT(*) FROM Procedure
26  WHERE projectId = ?;
```

Listing B.3: SQL Schema for Procedure

```
1   CREATE TABLE IF NOT EXISTS Action (
2       id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
3       procedureId INTEGER NOT NULL,
4       step INTEGER NOT NULL,
5       type TEXT NOT NULL,
6       goal TEXT NOT NULL,
7       FOREIGN KEY (procedureId) REFERENCES Procedure(id)
8   );
9
10  createAction:
11  INSERT INTO Action(procedureId, step, type, goal)
12  VALUES (?, ?, ?, ?);
13
14  selectActions:
15  SELECT * FROM Action
16  WHERE procedureId = ?;
17
18  countActions:
19  SELECT COUNT(*) FROM Action
20  WHERE procedureId = ?;
21
22  deleteByProcedure:
23  DELETE FROM Action
24  WHERE procedureId = ?;
```

Listing B.4: SQL Schema for Action

25

```
1   CREATE TABLE IF NOT EXISTS Test (
2       id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
3       procedureId INTEGER NOT NULL,
4       startUTC TEXT NOT NULL,
5       endUTC TEXT,
6       FOREIGN KEY (procedureId) REFERENCES Procedure(id)
7   );
8
9   startTest:
10  INSERT INTO Test(procedureId, startUTC)
11  VALUES (?, ?);
12
13  endTest:
14  UPDATE Test
15  SET endUTC = ?
16  WHERE id = ?;
17
18  lastInsertedRowId:
19  SELECT last_insert_rowid();
```

Listing B.5: SQL Schema for Test

```
1   CREATE TABLE IF NOT EXISTS ActionResult (
2       id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
3       testId INTEGER NOT NULL,
4       actionId INTEGER NOT NULL,
5       initState TEXT NOT NULL,
6       endState TEXT,
7       startUTC TEXT NOT NULL,
8       endUTC TEXT,
9       FOREIGN KEY (testId) REFERENCES Test(id),
10      FOREIGN KEY (actionId) REFERENCES Action(id)
11  );
12
13  startResult:
14  INSERT INTO ActionResult(testId, actionId, initState, startUTC)
15  VALUES (?, ?, ?, ?);
16
17  finishResult:
18  UPDATE ActionResult
19  SET endState = ?, endUTC = ?
20  WHERE id = ?;
21
22  lastInsertedRowId:
23  SELECT last_insert_rowid();
```

Listing B.6: SQL Schema for ActionResult

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] Nick Battle. *vdmj/LICENCE at master · nickbattle/vdmj*. URL: https://github.com/nickbattle/vdmj/blob/master/LICENCE (visited on 14/05/2024).

[28] Free Software Foundation, Inc. *The GNU General Public License v3.0 - GNU Project - Free Software Foundation*. URL: https://www.gnu.org/licenses/gpl-3.0.en.html (visited on 14/05/2024).

[29] Free Software Foundation, Inc. *Frequently Asked Questions about the GNU Licenses - GNU Project - Free Software Foundation*. URL: https://www.gnu.org/licenses/gpl-faq.html#IfLibraryIsGPL (visited on 14/05/2024).

[30] 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).

[31] 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).

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

[33] 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).

[34] 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).

[35] 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.

[36] The JUnit Team. *JUnit 5 User Guide - Migrating from JUnit 4*. URL: https://github.com/smyalygames/XPlaneConnect/commit/e7b8d1e811999b4f8d7230f60ba94368e14f1148 (visited on 15/05/2024).

[37] 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.

[38] 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.

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

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

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