Testing Quick Reference Handbooks in Flight Simulators

Anthony Berg (200871682)
Supervisor: Dr Leo Freitas

12th 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
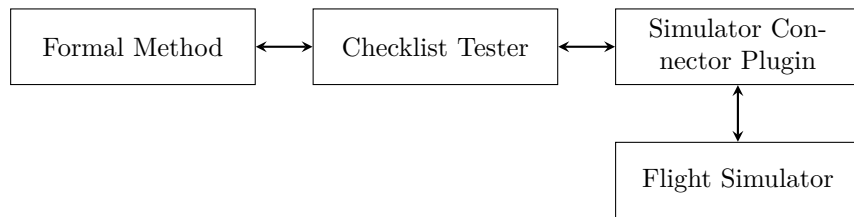
Figure 3.1: Abstract layout of components

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

3.2 Model

- 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 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

3.4 Decisions

Chapter 4

Results

4.1   Problems Found

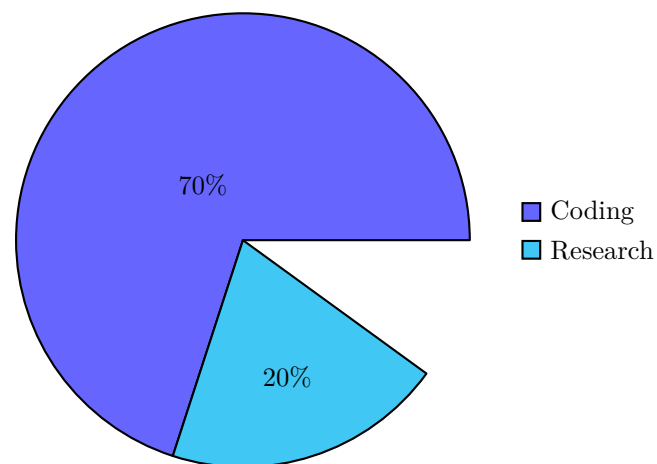## 4.2   LOC?

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

```
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!
```

11

```
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   */
```

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] 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.

[19] 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.

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

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

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