SOFTWARE TESTING: TUTORIAL 1 DISCUSSION

Tutorial 1 is difficult to get through in as much detail in a single sitting, so here is a sample solution.

Few used symbols may not be supported by every browser, so if cannot see a right-pointing arrow here (→), a set membership symbol here (∈), non-membership here (∉), a multiplication symbol here (×) or a not equal to symbol here (≠) then this page may not make sense: there'll be stuff missing.

ACTIVITIES 1-3: ITFS

Note that ITF is interpreted in some of the course source materials as “Independently Testable Feature,” in others as “Independently Testable Function” — the two are the same thing. It refers to an aspect or part of the system under test which can be tested in isolation. See p.170 of P&Y for more discussion of the term.

For much of the example material we deal with on this course, the system under test is so small/simple that one might identify the entire system as having just one ITF. This is arguably the case here: the Valid Connection function is intended to do to one thing, and one thing only: check whether two flights form a valid connection --- whether someone trying to make a longer journey can practically get off the first flight and onto the second. The decisions necessary involve establishing that the second flight departs from the airport at which the first arrives, and that the time gap between arrival and departure is sufficient for the appropriate transfer time for that airport (international transfers being treated differently to domestic transfers).

Three suggested refinements of the function's operation that came out of tutorials were as follows:

1. The function correctly identifies valid connections in situations where valid connections are offered. We'll call this ITF **VALID**.
2. The function correctly identifies invalid connections in situations where non-connecting flights (either due to mismatched airports or insufficient transfer time) are submitted. We'll call this ITF **INVALID**.
3. The function correctly identifies and flags error conditions. We'll call this ITF **ERROR**.

You could arrive at these from an examination of the rules that apply to the validity check, or from the function's return codes — in particular, looking at the return codes, the first item above corresponds to return code 0; the second item corresponds to return codes 15 and 16; and the third item corresponds to return codes 10 and 20. Use of documented system behaviour in order to break tests down like this is often a good idea. Surely however we could continue this process to break it down into one ITF per return code? It's possible, but we're beginning to split hairs, as will become apparent as we go on to develop characteristics for partitioning below. Let's stick with the above three ITFs for the moment.

ACTIVITIES 4-6: PARAMETERS, ENVIRONMENT, CHARACTERISTICS

Next we identify environmental factors and input parameters relevant to the ITFs. The input parameters are clearly identified as the arriving and departing flight data structures.

We could break these out into their structural fields (flight code, originating airport, departure time, destination airport, arrival time), but let's not do that yet — just keep working at a fairly high level for the time being.

Environmental factors are less clear. Remembering that the environment describes external factors which you need to configure in particular ways in order to specify and execute tests which will fully exercise the system, the major environmental factor here is definitely the database: any test will need to look the arriving and departing flights' origin and destination airports up in the database in order to decide whether the connection is a domestic connection or an international one, so we'll need to specify what data is in the airport database when testing.

Other possible environmental factors include the following:

- System memory: the specification notes that the Valid Connection function uses an *in-memory* table of airports. Could the system run out of memory? If we consider the number of airports in the world (about 44,000 in 2009 according to the CIA World Factbook) and the amount of data stored per airport (looks like only four small fields — a few bytes), this shouldn't be a problem on a modern PC.
- Start-up time: the specification mentions loading the airport database at system initialisation. This might cause responsiveness problems if initialisation takes a long time. However since the specification doesn't make any statements requiring the system to respond in a timely manner we can ignore this issue.
- Locale: for this kind of application, variations in time zone can cause huge amounts of trouble. The specification however states that times are expressed in universal time, so locale probably isn't an issue.

… and so on. Something you should always bear in mind when designing tests is your stopping point. We've already established that there are a very large number of factors which could affect system behaviour. We also know that there are usually a near-infinite number of possible inputs and control paths within even fairly simple software systems. Consequently you could continue testing most systems until the end of the universe without running out of tests. However you should always be on the lookout for ways to get these infinities under control. *Plausible* arguments to amalgamate or throw out classes of test are very important here, and the above three points give an example of the kind of thought process you might follow in order to justify ignoring certain issues. Particularly given that this exercise has been specified as a 50 minute tutorial (!), it seems reasonable to stop looking for more environmental factors beyond the airport database.

So:

- Environment:
  - Airport Database
- Input parameters:
  - Arriving Flight
  - Departing Flight

Remember too that in a larger system we might have different environmental factors and inputs for each ITF. Here however all three of our ITFs are affected by all three of the above factors.

CHARACTERISTICS

Many characteristics which will affect the outcome of your tests can be lifted directly from the system specification. Again, these will vary from ITF to ITF.

ITF 1 (VALID)

Characteristics relevant to the function's behaviour when generating a valid result (code 0) include:

- Arriving and Departing Flight's originating and destination airports must all have entries in the airport database.
- Arriving Flight's destination airport must match Departing Flight's originating airport.
- If flight is international (either flight's destination airport zone is different to its originating airport zone), then the connecting airport must allow international transfers (airport database record does not have an international connect time of -1).
- After the Arriving Flight's arrival time, there must be a gap no smaller than the appropriate (domestic or international) connect time before the Departing Flight's departure time.

ITF 2 (INVALID)

Characteristics relevant to the function's behaviour when generating an invalid connection result (codes 15 and 16) include:

- Arriving and Departing Flight's originating and destination airports must all have entries in the airport database.
- At least one of the following must be true:
  - Arriving Flight's destination airport doesn't match Departing Flight's originating airport.
  - If flight is international (either flight's destination airport zone is different to its originating airport zone), then the connecting airport doesn't allow international transfers (airport database record has an international connect time of -1).
  - The Departing Flight must leave before the Arriving Flight lands, or so soon afterwards that there's not enough (domestic or international) connect time.

Note that the first point is identical to ITF 1, and the second (with its three sub-points) is simply the logical inverse of the other three points in ITF 1. This suggests that really we should consider bundling ITF 1 and ITF 2 together since they form a logical pair within the space of well-formed input data. So, instead of testing for VALID and INVALID connections separately, we should test jointly for CORRECT behaviour during normal operation. Let's call this new ITF “ITF 4 (CORRECT)”, and forget about ITFs 1 and 2.

ITF 3 (ERROR)

Characteristics which would generate errors (codes 10 and 20) could include:

- At least one of the Arriving and Departing Flight's originating and destination airports doesn't have an entry in the airport database.
- Various other input data errors, such as invalid arrival or departure times or badly formed flight codes or airport codes (overlaps a bit with the first point), or missing (null?) Arriving or Departing Flight parameters.
- Various database content errors, such as badly formed airport zone codes or invalid connect times.
- More fundamental database problems, such as the database not being available.

Some of the above might be difficult to test without more information: without knowing what programming language we're working with, we don't know how to provide a null (or equivalent) Arriving Flight for example.

Again, it's worth noting that these characteristics are almost all just bad values for the same information we're using to partition ITF 4; if we include invalid data of various kinds in our partitions for ITF 4, then we'll actually cover ITF 3 pretty well. Let's do that: by including invalid inputs in ITF 4 we can forget about ITF 3, and just have a single ITF.

ACTIVITY 7: PARTITIONS/VALUE CLASSES

Recall that value classes are essentially the same thing as partitions. First, let's introduce some shorthand notations so we're not writing so much:

Acronym	Meaning
AF	Arriving Flight
DF	Departing Flight
IFC	Identifying Flight Code
OAC	Originating Airport Code

SDT	Scheduled Departure Time
DAC	Destination Airport Code
SAT	Scheduled Arrival Time
AZ	Airport Zone
DCT	Domestic Connect Time
ICT	International Connect Time
CT	Connect time

So now we write “AF→DAC→DCT” instead of “Arriving Flight's Destination Airport's Domestic Connect Time”…

Here are the partitions we derive from the characteristics we identified above. Note that we're now splitting the inputs out into some of their subfields so we can see when we're choosing between a set of mutually exclusive alternatives for a particular characteristic.

Characteristic	Partition	Value Classes	Comment
Database ok?	**P1**	Database ok	
	P2	Database not ok	Connect failure or similar
AF→OAC in DB?	**P3**	AF→OAC ∈ database	Arriving flight's originating airport is in database
	P4	AF→OAC ∉ database	Arriving flight's originating airport not in database
AF→DAC in DB?	**P5**	AF→DAC ∈ database	Arriving flight's destination airport is in database
	P6	AF→DAC ∉ database	Arriving flight's destination airport not in database
DF→OAC in DB?	**P7**	DF→OAC ∈ database	Departing flight's originating airport is in database
	P8	DF→OAC ∉ database	Departing flight's originating airport not in database
DF→DAC in DB?	**P9**	DF→DAC ∈ database	Departing flight's destination airport is in database
	P10	DF→DAC ∉ database	Departing flight's destination airport not in database
Flights meet?	**P11**	AF→DAC = DF→OAC	Flights meet at the same airport
	P12	AF→DAC ≠ DF→OAC	Flights don't meet at the same airport
AF international?	**P13**	AF→OAC→AZ = AF→DAC→AZ	Arriving flight is domestic (airport zones match)
	P14	AF→OAC→AZ ≠ AF→DAC→AZ	Arriving flight is international (airport zones don't match)
DF international?	**P15**	DF→OAC→AZ = DF→DAC→AZ	Departing flight is domestic (airport zones match)
	P16	DF→OAC→AZ ≠ DF→DAC→AZ	Departing flight is international (airport zones don't match)
Connect time ok?	**P17**	CT = -1	(International) transfers disallowed
	P18	DF→SDT - AF→SAT < CT	Departing flight leaves too early
	P19	DF→SDT - AF→SAT = CT	Departing flight leaves as early as permitted
	P20	DF→SDT - AF→SAT > CT	Departing flight leaves enough time to transfer

ACTIVITY 8: CONSTRAINTS

On the face of it this leaves us with 2 × 2 × 2 × 2 × 2 × 2 × 2 × 2 × 4 = 1,024 combinations (two possibilities for each characteristic except connect time, for which we have four). Many partition combinations are impossible, and remember that all error values need to be tested on their own with all other inputs and environment factors set to valid values. Let's annotate our partitions accordingly:

Characteristic	Partition	Value Classes	Conditions/properties
Database ok?	**P1**	Database ok	
	P2	Database not ok	[error]
AF→OAC in DB?	**P3**	AF→OAC ∈ database	
	P4	AF→OAC ∉ database	[error]
AF→DAC in DB?	**P5**	AF→DAC ∈ database	
	P6	AF→DAC ∉ database	[error]
DF→OAC in DB?	**P7**	DF→OAC ∈ database	
	P8	DF→OAC ∉ database	[error]
DF→DAC in DB?	**P9**	DF→DAC ∈ database	
	P10	DF→DAC ∉ database	[error]
Flights meet?	**P11**	AF→DAC = DF→OAC	
	P12	AF→DAC ≠ DF→OAC	[error]
AF international?	**P13**	AF→OAC→AZ = AF→DAC→AZ	
	P14	AF→OAC→AZ ≠ AF→DAC→AZ	[property International]
DF international?	**P15**	DF→OAC→AZ = DF→DAC→AZ	
	P16	DF→OAC→AZ ≠ DF→DAC→AZ	[property International]

Connect time ok?	**P17**	CT = -1	[if International]
	P18	DF→SDT - AF→SAT < CT	
	P19	DF→SDT - AF→SAT = CT	[single]
	P20	DF→SDT - AF→SAT > CT	

Note we've annotated P19 as “single” just to test the boundary between too little connect time and enough connect time; the specification isn't clear on whether *exactly* the right connect time is safe or not, but let's assume that it will be. We use “single” because it's a special case that we think only needs investigating once; arguably we could do the same for P18 and P20 — there's a bit of value judgement/intuition here — but the idea is that not making these two single ensures that CT is tested thoroughly (too little CT and enough CT) for both domestic and international flights.

Now: if every “single” or “error” condition needs to be tested on its own with otherwise valid inputs, then we need seven such tests. Then we're left with non-error, non-single tests: our first six characteristics only have one remaining valid value and our connect time partition has three, so we now have 1 × 1 × 1 × 1 × 1 × 1 × 2 × 2 × 3 = 12 combinations left to test, for a total of 7 + 12 = 19 tests. In fact it's even less than that if we look at the “International” condition: for combination (P13, P15) (a domestic flight), International is false, so we don't test P17. We do test P17 for the three remaining International combinations (P13, P16), (P14, P15) and (P14, P16). This modifies our product from (2 × 2) × 3 to (1) × 2 + (3) × 3 = 11 (the (1) and (3) are a breakdown of (2 × 2) according to which combinations produce domestic and international flights). This saves us one test, so we're now down to 18.

ACTIVITY 9: TEST CASE SPECIFICATIONS

In this table we enumerate all 18 test case specifications; any characteristic for which we don't specify a partition must be chosen from one of its valid (non-error) partitions (but we don't care which). I make this explicit for test 7, the first non-error test, so we can see that all 20 partitions are covered.

Test	Specification	Partitions covered
1	Database not ok	P2
2	AF→OAC ∉ database	P4
3	AF→DAC ∉ database	P6
4	DF→OAC ∉ database	P8
5	DF→DAC ∉ database	P10
6	AF→DAC ≠ DF→OAC	P12
7	Domestic; shortest possible connect	P1, P3, P5, P7, P9, P11, P13, P15, P19
8	Domestic; connect too short	P13, P15, P18
9	Domestic; connect long enough	P13, P15, P20
10	Int'l DF; int'l disallowed	P13, P16, P17
11	Int'l DF; connect too short	P13, P16, P18
12	Int'l DF; connect long enough	P13, P16, P20
13	Int'l AF; int'l disallowed	P14, P15, P17
14	Int'l AF; connect too short	P14, P15, P18
15	Int'l AF; connect long enough	P14, P15, P20
16	Both int'l; int'l disallowed	P14, P16, P17
17	Both int'l; connect too short	P14, P16, P18
18	Both int'l; connect long enough	P14, P16, P20

TEST CASES

Finally, we can write out our test cases; here in order to save some effort I'll use the same database for all test cases (but switch it off for test case 1, where the database is “broken” in some way):

Environment: database contents			
Airport code	**Airport Zone**	**Domestic connect time**	**International connect time**
GLA	EU	30	90
EDI	EU	35	95
SKL	EU	25	-1
NRT	JP	29	85
JFK	US	40	120

The test cases now follow; since we don't use AF→IFC, DF→IFC, AF→SDT or DF→SAT (and didn't define tests based on them), I've not included them in the table.

Test	Arriving flight			Departing flight			Return code	Notes
	OAC	**DAC**	**SAT**	**OAC**	**SDT**	**DAC**		
1	GLA	NRT	13:00	NRT	15:00	JFK	20	Database broken
2	XXX	NRT	13:00	NRT	15:00	JFK	10	Bad AF→OAC
3	GLA	XXX	13:00	NRT	15:00	JFK	10	Bad AF→DAC
4	GLA	NRT	13:00	XXX	15:00	JFK	10	Bad DF→OAC
5	GLA	NRT	13:00	NRT	15:00	XXX	10	Bad DF→DAC
6	GLA	NRT	13:00	JFK	15:00	NRT	16	Flights don't meet
7	GLA	SKL	13:00	SKL	13:25	EDI	0	Shortest possible connect
8	GLA	SKL	13:00	SKL	13:24	EDI	15	Domestic; too short
9	GLA	SKL	13:00	SKL	13:26	EDI	0	Domestic; long enough
10	GLA	SKL	13:00	SKL	15:00	JFK	20	International DF; disallowed
11	GLA	EDI	13:00	EDI	14:34	JFK	15	International DF; too short
12	GLA	EDI	13:00	EDI	14:36	JFK	0	International DF; long enough
13	NRT	SKL	13:00	SKL	15:00	GLA	20	International AF; disallowed
14	NRT	EDI	13:00	EDI	14:34	GLA	15	International AF; too short
15	NRT	EDI	13:00	EDI	14:36	GLA	0	International AF; long enough
16	NRT	SKL	13:00	SKL	15:00	JFK	20	Both international; disallowed
17	NRT	EDI	13:00	EDI	14:34	JFK	15	Both international; too short
18	NRT	EDI	13:00	EDI	14:36	JFK	0	Both international; long enough

There are lots of other things we could think about here — other error conditions, what happens if arrival and departure times are on different days, etc. — but I don't want to overcomplicate this example and think this is a reasonable set of test cases.

Version 1.1, 2018/01/09 13:42:03

Informatics Forum, 10 Crichton Street, Edinburgh, EH8 9AB, Scotland, UK
Tel: +44 131 651 5661, Fax: +44 131 651 1426, E-mail: school-office@inf.ed.ac.uk
Please contact our webadmin with any comments or corrections. **Logging and Cookies**
Unless explicitly stated otherwise, all material is copyright © The University of Edinburgh