SWEN90006: Assignment 1

Name: Yangzhe Xie
Student number: 1029787
Email: yangzhe.xie@student.unimelb.edu.au

September 3, 2019

1 Task 1

1.1 Test template trees

Figure 1 - 4 shows the test template trees for the API addUser, loginUser, updateDetails, and retrieveDetails respectively.

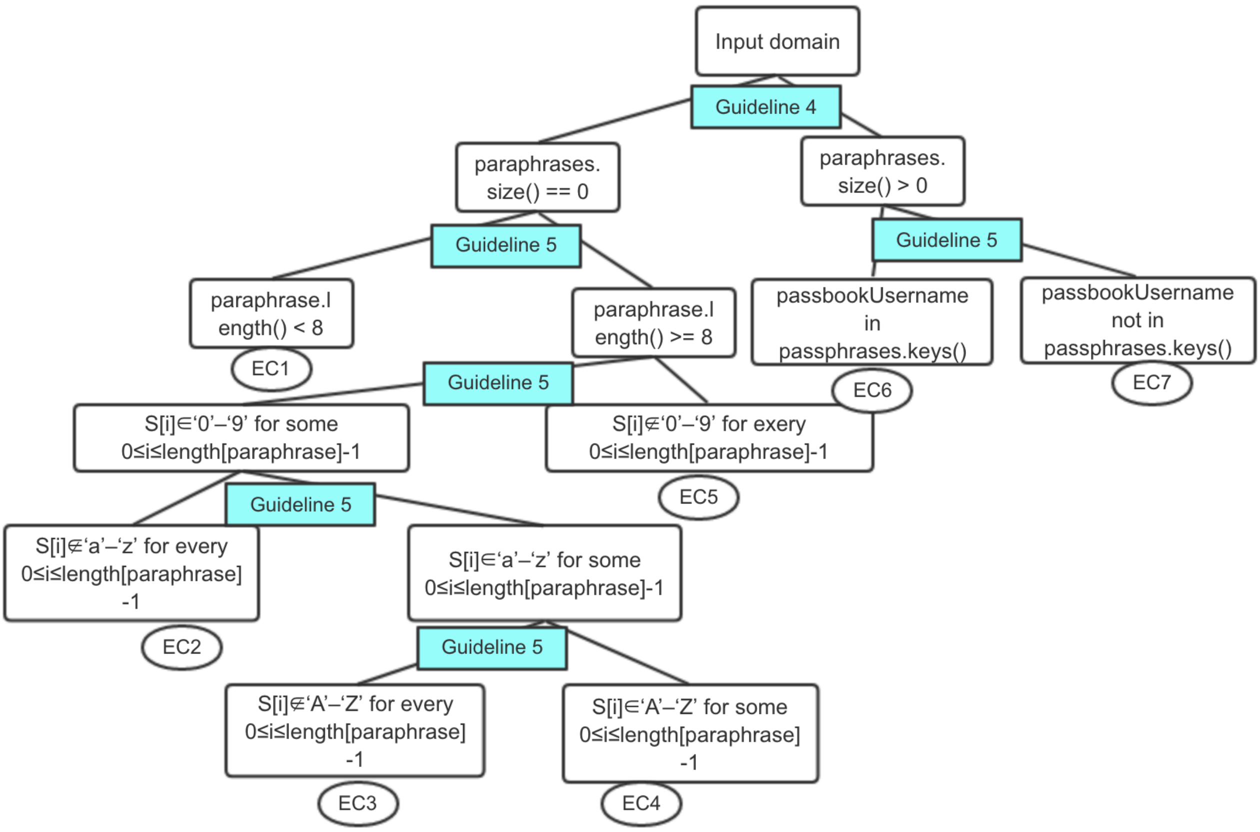

Figure 1: Test template tree for addUser()

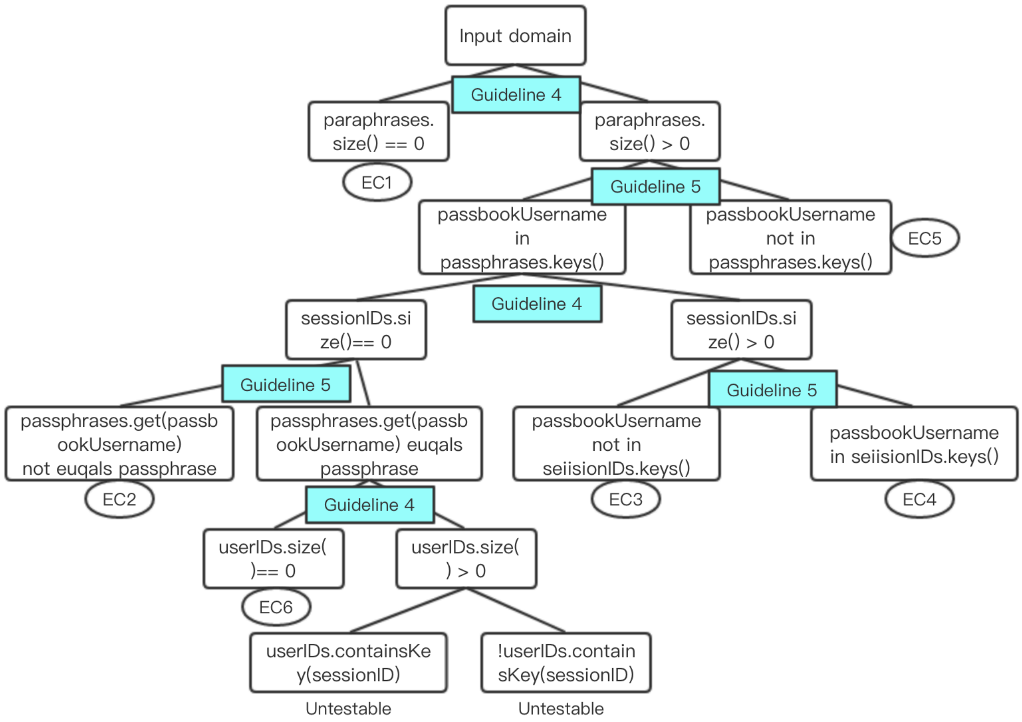

Figure 2: Test template tree for loginUser()

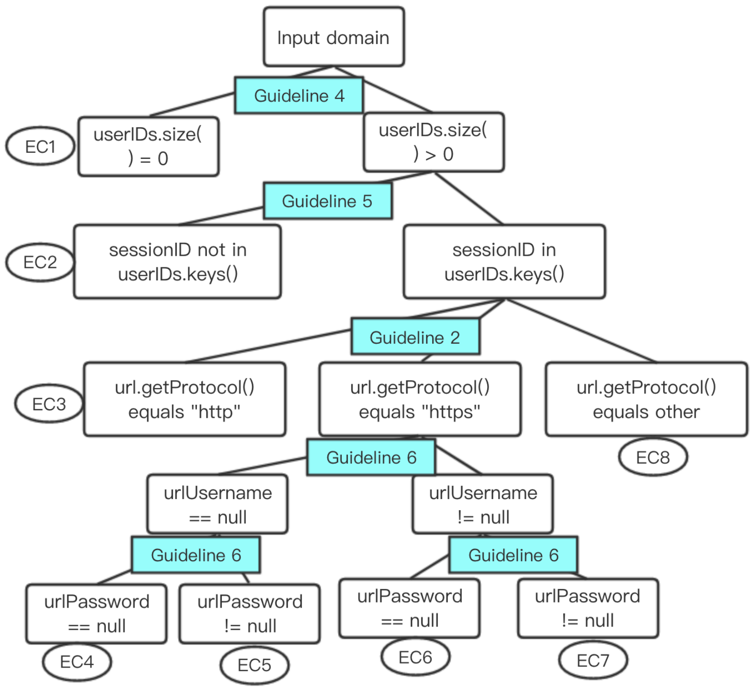

Figure 3: Test template tree for updateDetails()

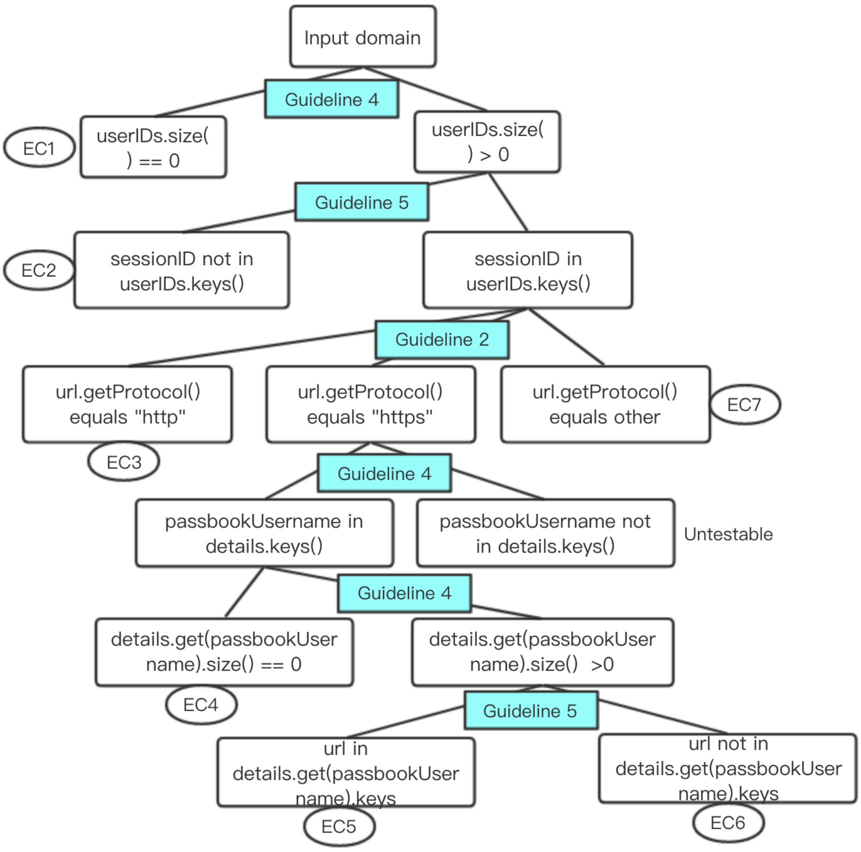

Figure 4: Test template tree for retrieveDetails()

1.2 Do your set of equivalence classes cover the input space?

My set of equivalence classes cover the input space. The reasons are as follows:

1) All leaf nodes are divided strictly and carefully, so that they do not overlap with other leaf.

2) The collection of the set of each sibling node covers all the cases of their parent node.

3) If two variables are independent of each other, then the subtree of one variable can be added to a leaf node of the other variable. In this case, all the nodes add up to cover all situations.

4) As part of your input domain, the instance variables should also be considered. Note that all of these variables are collections, so according to guideline 4, we should follow the zero-one-many rule. But in this particular case, we just care about whether the collection contains some values. So I combined the two cases (number of elements equals 1 and greater than 1) into one (greater than 0), which does not affect the results of the tests.

2 Task 2: Test cases associated with equivalence classes

2.1 addUser

Table 1: Test cases for addUser associated with equivalence classes

ID	Test case	Expected output
EC1	paraphrases = {}, passbookUsername = "abc", paraphrase = "1aA"	WeakPassphraseException
EC2	paraphrases = {}, passbookUsername = "abc", paraphrase = "1234567A"	WeakPassphraseException
EC3	paraphrases = {}, passbookUsername = "abc", paraphrase = "1234567a"	WeakPassphraseException
EC4	paraphrases = {}, passbookUsername = "abc", paraphrase = "123456aA"	-
EC5	paraphrases = {}, passbookUsername = "abc", paraphrase = "abcdABCD"	WeakPassphraseException
EC6	paraphrases = {"abcd":"123456aA"}, passbookUsername = "abcd", paraphrase = "123456aA"	DuplicateUserException
EC7	paraphrases = {"abcd":"123456aA"}, passbookUsername = "abc", paraphrase = "123456aA"	-

2.2 loginUser

Table 2: Test cases for loginUser associated with equivalence classes

ID	Test case	Expected output
EC1	paraphrases = {}, sessionIDs = {}, userIDs = {}, passbookUsername = "abc", paraphrase = "123456aA"	NoSuchUserException
EC2	paraphrases = {"abc":"123456aA"}, sessionIDs = {}, userIDs = {} passbookUsername = "abc", paraphrase = "123456aB"	IncorrectPassphraseException
EC3	paraphrases = {"abc":"123456aA"}, sessionIDs = {"def":123}, userIDs = {} passbookUsername = "abc", paraphrase = "123456aA"	-
EC4	paraphrases = {"abc":"123456aA"}, sessionIDs = {"abc":123}, userIDs = {} passbookUsername = "abc", paraphrase = "123456aA"	AlreadyLoggedInException
EC5	paraphrases = {"abc":"123456aA"}, sessionIDs = {}, userIDs = {} passbookUsername = "abcd", paraphrase = "123456aA"	NoSuchUserException
EC6	paraphrases = {"abc":"123456aA"}, sessionIDs = {}, userIDs = {} passbookUsername = "abc", paraphrase = "123456aA"	-

2.3 updateDetails

Table 3: Test cases for updateUserDetails associated with equivalence classes

ID	Test case	Expected output

EC1	userIDs = {}, sessionID = 123, url = "http://test.com", urlUsername = "123", urlPassword = "456"	InvalidSessionIDException
EC2	userIDs = {123:"abc"}, sessionID = 456, url = "http://test.com", urlUsername = "123", urlPassword = "456"	InvalidSessionIDException
EC3	userIDs = {123:"abc"}, sessionID = 123, url = "http://test.com", urlUsername = "123", urlPassword = "456"	-
EC4	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", urlUsername = null, urlPassword = null	-
EC5	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", urlUsername = null, urlPassword = "123"	-
EC6	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", urlUsername = "123", urlPassword = null	-
EC7	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", urlUsername = "123", urlPassword = "123"	-
EC8	userIDs = {123:"abc"}, sessionID = 123, url = "ftp://test.com", urlUsername = "123", urlPassword = "123"	MalformedURLException

2.4 retrieveDetails

Table 4: Test cases for retrieveDetails associated with equivalence classes

ID	Test case	Expected output
EC1	userIDs = {}, sessionID = 123, url = "http://test.com", details = {"abc":{"http://test.com":{"aaa":"bbb"}}}	InvalidSessionIDException
EC2	userIDs = {123:"abc"}, sessionID = 456, url = "http://test.com", details = {"abc":{"http://test.com":{"aaa":"bbb"}}}	InvalidSessionIDException
EC3	userIDs = {123:"abc"}, sessionID = 123, url = "http://test.com", details = {"abc":{"http://test.com":{"aaa":"bbb"}}}	-
EC4	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", details = {"abc":{}}	NoSuchURLException
EC5	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", details = {"abc":{"https://test.com":{"aaa":"bbb"}}}	-
EC6	userIDs = {123:"abc"}, sessionID = 123, url = "http://test.com", details = {"abc":{"http://java.com":{"aaa":"bbb"}}}	NoSuchURLException
EC7	userIDs = {123:"abc"}, sessionID = 123, url = "ftp://test.com", details = {"abc":{"http://test.com":{"aaa":"bbb"}}}	MalformedURLException

3 Task 3 Boundary-value analysis

3.1 addUser

Table 5: Test cases for addUser associated with boundary-value analysis

Test ID	EC	Test case	Boundary
1	EC1	paraphrases = {}, passbookUsername = "abc", paraphrase = "12345aA"	off point for paraphrase.length() < 8 and on point for paraphrases.size() == 0
2	EC2	paraphrases = {}, passbookUsername = "abc", paraphrase = "123456`A"	off point for S[i] not in ‘a’ - ‘z’, for every 0 <= i <= length[paraphrase]-1
3	EC2	paraphrases = {}, passbookUsername = "abc", paraphrase = "123456{A"	off point for S[i] not in ‘a’ - ‘z’, for every 0 <= i <= length[paraphrase]-1
4	EC3	paraphrases = {}, passbookUsername = "abc", paraphrase = "123456@a"	off point for S[i] not in ‘A’ - ‘Z’, for every 0 <= i <= length[paraphrase]-1
5	EC3	paraphrases = {}, passbookUsername = "abc", paraphrase = "123456[a"	off point for S[i] not in ‘A’ - ‘Z’, for every 0 <= i <= length[paraphrase]-1
6	EC4	paraphrases = {}, passbookUsername = "abc", paraphrase = "234567nA"	on point for S[i] in ‘A’ - ‘Z’, for some 0 <= i <= length[paraphrase]-1
7	EC4	paraphrases = {}, passbookUsername = "abc", paraphrase = "234567nZ"	on point for S[i] in ‘A’ - ‘Z’, for some 0 <= i <= length[paraphrase]-1
8	EC4	paraphrases = {}, passbookUsername = "abc", paraphrase = "234567Na"	on point for S[i] in ‘a’ - ‘z’, for some 0 <= i <= length[paraphrase]-1
9	EC4	paraphrases = {}, passbookUsername = "abc", paraphrase = "234567Nz"	on point for S[i] in ‘a’ - ‘z’, for some 0 <= i <= length[paraphrase]-1
10	EC4	paraphrases = {}, passbookUsername = "abc", paraphrase = "abcdABC0"	on point for S[i] in ‘0’ - ‘9’, for some 0 <= i <= length[paraphrase]-1
11	EC4	paraphrases = {}, passbookUsername = "abc", paraphrase = "abcdABC9"	on point for S[i] in ‘0’ - ‘9’, for some 0 <= i <= length[paraphrase]-1
12	EC5	paraphrases = {}, passbookUsername = "abc", paraphrase = "abcdABC/"	off point for S[i] not in ‘0’ - ‘9’, for every 0 <= i <= length[paraphrase]-1
13	EC5	paraphrases = {}, passbookUsername = "abc", paraphrase = "abcdABC:"	off point for S[i] not in ‘0’ - ‘9’, for every 0 <= i <= length[paraphrase]-1
14	EC6	paraphrases = {"abcd":"123456aA"}, passbookUsername = "abcd", paraphrase = "123456aA"	on point for passbookUsername in passphrases.keys() and off point for paraphrases.size() > 0

15	EC7	paraphrases = {"abcd":"123456aA"}, passbookUsername = "abc", paraphrase = "123456aA"	on point for passbookUsername not in passphrases.keys()

3.2 loginUser

Table 6: Test cases for loginUser associated with boundary-value analysis

Test ID	EC	Test case	Boundary
1	EC1	paraphrases = {}, sessionIDs = {}, userIDs = {}, passbookUsername = "abc", paraphrase = "123456aA"	on point for paraphrases.size() == 0
2	EC2	paraphrases = {"abc":"123456aA"}, sessionIDs = {}, userIDs = {} passbookUsername = "abc", paraphrase = "123456aB"	on point for passphrases.get (passbookUsername) not equals passphrase and on point for sessionIDs.size() == 0
3	EC3	paraphrases = {"abc":"123456aA"}, sessionIDs = {"def":123}, userIDs = {} passbookUsername = "abc", paraphrase = "123456aA"	on point for passbookUsername not in sessionIDs.keys() and off point for sessionIDs.size() > 0
4	EC4	paraphrases = {"abc":"123456aA"}, sessionIDs = {"abc":123}, userIDs = {} passbookUsername = "abc", paraphrase = "123456aA"	on point for passbookUsername in sessionIDs.keys()
5	EC5	paraphrases = {"abc":"123456aA"}, sessionIDs = {}, userIDs = {} passbookUsername = "abcd", paraphrase = "123456aA"	on point for passbookUsername not in passphrases.keys() and off point for paraphrases.size() > 0
6	EC6	paraphrases = {"abc":"123456aA"}, sessionIDs = {}, userIDs = {} passbookUsername = "abc", paraphrase = "123456aA"	on point for passphrases.get (passbookUsername) equals passphrase

3.3 updateDetails

Table 7: Test cases for updateUserDetails associated with boundary-value analysis

Test ID	EC	Test case	Boundary
1	EC1	userIDs = {}, sessionID = 123, url = "http://test.com", urlUsername = "123", urlPassword = "456"	on point for userIDs.size() == 0
2	EC2	userIDs = {123:"abc"}, sessionID = 456, url = "http://test.com", urlUsername = "123", urlPassword = "456"	on point for sessionID not in userIDs.keys() and off point for userIDs.size() > 0
3	EC3	userIDs = {123:"abc"}, sessionID = 123, url = "http://test.com", urlUsername = "123", urlPassword = "456"	on point for url.getProtocol() equals "http" and on point for sessionID in userIDs.keys()

4	EC4	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", urlUsername = null, urlPassword = null	on point for urlUsername == null and urlPassword == null and url.getProtocol() equals "https"
5	EC5	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", urlUsername = null, urlPassword = "456"	on point for urlUsername == null and urlPassword != null
6	EC6	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", urlUsername = "123", urlPassword = null	on point for urlUsername != null and urlPassword == null
7	EC7	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", urlUsername = "123", urlPassword = "456"	on point for urlUsername != null and urlPassword != null
8	EC8	userIDs = {123:"abc"}, sessionID = 123, url = "ftp://test.com", urlUsername = "123", urlPassword = "456"	on point for url.getProtocol() equals other

3.4 retrieveDetails

Table 8: Test cases for retrieveDetails associated with boundary-- value analysis

Test ID	EC	Test case	Boundary
1	EC1	userIDs = {}, sessionID = 123, url = "http://test.com", details = {"abc":{"http://test.com":{"aaa":"bbb"}}}	on point for userIDs.size() == 0
2	EC2	userIDs = {123:"abc"}, sessionID = 456, url = "http://test.com", details = {"abc":{"http://test.com":{"aaa":"bbb"}}}	off point for userIDs.size() > 0 and on point for sessionID not in userIDs.keys()
3	EC3	userIDs = {123:"abc"}, sessionID = 123, url = "http://test.com", details = {"abc":{"http://test.com":{"aaa":"bbb"}}}	on point for url.getProtocol() equals "http" and sessionID in userIDs.keys() and on point for sessionID in userIDs.keys()
4	EC4	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", details = {"abc":{}}	on point for details.get(passbookUsername) .size() == 0 and url.getProtocol() equals "https"
5	EC5	userIDs = {123:"abc"}, sessionID = 123, url = "https://test.com", details = {"abc":{"https://test.com":{"aaa":"bbb"}}}	on point for url in details.get(passbookUsername).keys and off point for details.get(passbookUsername).size() > 0
6	EC6	userIDs = {123:"abc"}, sessionID = 123, url = "http://test.com", details = {"abc":{"http://java.com":{"aaa":"bbb"}}}	on point for url not in details.get(passbookUsername).keys
7	EC7	userIDs = {123:"abc"}, sessionID = 123, url = "ftp://test.com", details = {"abc":{"http://test.com":{"aaa":"bbb"}}}	on point for url.getProtocol() equals other and on point for sessionID in userIDs.keys()

4 Task 5 Multiple-conditions coverage

4.1 addUser

Table 9 shows all test objectives for the API addUser.

Table 9: All multiple-conditions for addUser

Test Objective ID	Condition	Output(s)
1	passphrases.containsKey(passbookUsername)	true
2	passphrases.containsKey(passbookUsername)	false
3	passphrase.length() <MINIMUM_PASSPHRASE_LENGTH	true
4	passphrase.length() <MINIMUM_PASSPHRASE_LENGTH	false
5	i < passphrase.length()	true
6	i < passphrase.length()	false
7	'a' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'z'	false false
8	'a' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'z'	true false
9	'a' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'z'	false true
10	'a' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'z'	true true
11	'A' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'Z'	false false
12	'A' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'Z'	true false
13	'A' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'Z'	false true
14	'A' <= passphrase.charAt(i) && passphrase.charAt(i) <= 'Z'	true true
15	'0' <= passphrase.charAt(i) && passphrase.charAt(i) <= '9'	false false
16	'0' <= passphrase.charAt(i) && passphrase.charAt(i) <= '9'	true false
17	'0' <= passphrase.charAt(i) && passphrase.charAt(i) <= '9'	false true
18	'0' <= passphrase.charAt(i) && passphrase.charAt(i) <= '9'	true true
19	!containsLowerCase || !containsUpperCase || !containsNumber	false false false
20	!containsLowerCase || !containsUpperCase || !containsNumber	false false true
21	!containsLowerCase || !containsUpperCase || !containsNumber	false true false
22	!containsLowerCase || !containsUpperCase || !containsNumber	false true true
23	!containsLowerCase || !containsUpperCase || !containsNumber	true false false
24	!containsLowerCase || !containsUpperCase || !containsNumber	true false true
25	!containsLowerCase || !containsUpperCase || !containsNumber	true true false
26	!containsLowerCase || !containsUpperCase || !containsNumber	true true true

4.1.1 partitioning score for the API addUser

According to table 9 and table 10, there are 26 test objective in total, of which 20 are tested and 6 untested. So the partitioning score for the API addUser of partitioning is:

$$\frac{20}{26} \approx 76.9\%$$

Table 10: Multiple-conditions tested of partitioning

Test Case	CoverTest Objective ID
1	2 3
2	2 4 5 6 7 9 11 13 14 15 16 18 23
3	2 4 5 6 7 9 10 11 12 13 15 16 18 21
4	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19
5	2 4 5 6 7 9 10 11 12 14 15 16 20

6	1
7	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19

4.1.2 boundary score for the API addUser

According to table 9 and table 11, there are 26 test objective in total, of which 22 are tested and 4 untested. So the partitioning score for the API addUser of boundary is:

$$\frac{22}{26} \approx 84.6\%$$

Table 11: Multiple-conditions tested of boundary

Test Case	CoverTest Objective ID
1	2 3
2	2 4 5 6 7 9 11 13 14 15 16 18 23
3	2 4 5 6 7 8 9 11 13 14 15 16 18 23
4	2 4 5 6 7 9 10 11 12 13 15 16 18 21
5	2 4 5 6 7 9 10 11 12 13 15 16 18 21
6	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19
7	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19
8	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19
9	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19
10	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19
11	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19
12	2 4 5 6 7 9 10 11 12 14 15 16 20
13	2 4 5 6 7 9 10 11 12 14 15 16 17 20
14	1
15	2 4 5 6 7 9 10 11 12 13 14 15 16 18 19

4.2 loginUser

Table 12 shows all test objectives for the API loginUser

Table 12: All multiple-conditions for loginUser

Test Objective ID	Condition	Output(s)
1	!passphrases.containsKey(passbookUsername)	true
2	!passphrases.containsKey(passbookUsername)	false
3	sessionIDs.get(passbookUsername) != null	true
4	sessionIDs.get(passbookUsername) != null	false
5	!passphrases.get(passbookUsername).equals(passphrase)	true
6	!passphrases.get(passbookUsername).equals(passphrase)	false
7	userIDs.containsKey(sessionID)	true
8	userIDs.containsKey(sessionID)	false

4.2.1 partitioning score for the API loginUser

According to table 12 and table 13, there are 8 test objective in total, of which 7 are tested and 1 untested. So the partitioning score for the API loginUser of partitioning is:

$$\frac{7}{8} \approx 87.5\%$$

Table 13: Multiple-conditions tested of partitioning

Test Case	CoverTest Objective ID
1	1
2	2 4 5
3	2 4 6 8
4	2 3
5	1
6	2 4 6 8

4.2.2 boundary score for the API loginUser

According to table 12 and table 14, there are 8 test objective in total, of which 7 are tested and 1 untested. So the partitioning score for the API loginUser of boundary is:

$$\frac{7}{8} \approx 87.5\%$$

Table 14: Multiple-conditions tested of boundary

Test Case	CoverTest Objective ID
1	1
2	2 4 5
3	2 4 6 8
4	2 3
5	1
6	2 4 6 8

4.3 updateDetails

Table 15 shows all test objectives for the API updateDetails

Table 15: All multiple-conditions for updateDetails

Test Objective ID	Condition	Output(s)
1	passbookUsername == null	true
2	passbookUsername == null	false
3	!Arrays.asList(VALID_URL_PROTOCOLS).contains(url.getProtocol())	true
4	!Arrays.asList(VALID_URL_PROTOCOLS).contains(url.getProtocol())	false
5	urlUsername == null || urlPassword == null	false false
6	urlUsername == null || urlPassword == null	false true
7	urlUsername == null || urlPassword == null	true false
8	urlUsername == null || urlPassword == null	true true

4.3.1 partitioning score for the API updateDetails

According to table 15 and table 16 there are 8 test objective in total, of which 8 are tested and 0 untested. So the partitioning score for the API updateDetails of partitioning is:

$$\frac{8}{8} \approx 100\%$$

Table 16: Multiple-conditions tested of partitioning

Test Case	CoverTest Objective ID
1	1
2	1
3	2 4 5
4	2 4 8
5	2 4 7
6	2 4 6
7	2 4 5
8	2 3

4.3.2 boundary score for the API updateDetails

According to table 15 and table 17 there are 8 test objective in total, of which 8 are tested and 0 untested. So the partitioning score for the API updateDetails of boundary is:

$$\frac{8}{8} \approx 100\%$$

Table 17: Multiple-conditions tested of boundary

Test Case	CoverTest Objective ID
1	1
2	1
3	2 4 5
4	2 4 8
5	2 4 7
6	2 4 6
7	2 4 5
8	2 3

4.4 retrieveDetails

Table 18 shows all test objectives for the API retrieveDetails

Table 18: All multiple-conditions for retrieveDetails

Test Objective ID	Condition	Output(s)
1	passbookUsername == null	true
2	passbookUsername == null	false
3	!Arrays.asList(VALID_URL_PROTOCOLS).contains(url.getProtocol())	true
4	!Arrays.asList(VALID_URL_PROTOCOLS).contains(url.getProtocol())	false
5	pt == null	false
6	pt == null	true
7	pair == null	false
8	pair == null	true

4.4.1 partitioning score for the API retrieveDetails

According to table 18 and table 19, there are 8 test objective in total, of which 7 are tested and 1 untested. So the partitioning score for the API retrieveDetails of partitioning is:

$$\frac{7}{8} \approx 87.5\%$$

Table 19: Multiple-conditions tested of partitioning

Test Case	CoverTest Objective ID
1	1
2	1
3	2 4 5 7
4	2 4 5 8
5	2 4 5 7
6	2 4 5 8
7	2 3

4.4.2 boundary score for the API retrieveDetails

According to table 18 and table 20, there are 8 test objective in total, of which 7 are tested and 1 untested. So the partitioning score for the API retrieveDetails of boundary is:

$$\frac{7}{8} \approx 87.5\%$$

Table 20: Multiple-conditions tested of boundary

Test Case	CoverTest Objective ID
1	1
2	1
3	2 4 5 7
4	2 4 5 8
5	2 4 5 7
6	2 4 5 8
7	2 3

5 Question 7 Compare the two sets of test cases and their results.

I think in most cases, boundary-value analysis is more effective. The reasons are as followsff

1) Theoretically, the aim of boundary-value analysis is to find better test cases that are more likely to detect errors. In other words, boundary-value analysis is a supplement to equivalence partitioning. So in terms of their nature, the former are more efficient.

2) From the perspective of the coverage of the valid input/output domain, both methods can well cover the input domain. Of course, the premise is that all equivalence class are divided disjointedly and fully cover the input domain.

3) In section 4.1, we can see that the multiple-conditions coverage score (addUser) of boundary-value analysis is higher than that of equivalence partitioning. The reason is that the boundaries are related to flow control constructs. Boundary-value analysis can bring more test cases near the boundary, which are more likely to cover more conditions.

4) There may not be a boundary that makes sense in a equivalence class, this might be the reason why the multiple-conditions coverage score of boundary-value analysis is equal to that of equivalence partitioning in section 4.2, 4.3 and 4.4. In these cases, it is hard or not possible to come up with a better test cases with boundary-value analysis.

5) As for the mutants killed by the two test suites, the test suite of boundary-value analysis kills more mutants than equivalence partitioning (5 vs. 4). This also confirms the point that the errors

are more likely to be at the boundary between equivalence classes[1], which makes boundary-value analysis more effective.

References

[1] The School of Computing and Software Systems, The University of Melbourne, "Software & Security Testing Course Notes for SWEN90006" unpublished, 2019.