CMPE 287 Software Quality Testing HW1

Name: Akshata Deo
SJSU ID: 012565761

Question #1: Basic Concepts

a. What are the challenges in testing AI software?

Ans. AI software quality testing field is comparatively new, hence not been exploit much. Due to which, it faces many challenges. Challenges are as follows:

- AI applications are built with machine learning algorithms and big data. So to finalize the requirements and coverage criteria becomes difficult in this case
- AI software are learning based system. So to develop quality test models using systematic methods become difficult
- Development of automatic solution tools to support AI based software is challenging

b. Why software testing is so hard?

Ans. There are various reasons why software testing is so hard. Some of them are as follows-

- While testing an application, tester not check just for correctness. Performance is also an important parameter, that should be checked while performing testing
- Testing should be performed using negative test cases as well. In negative test cases, application should give proper error message instead of abrupt failure
- Testing has no definite end limits. But at least the application should be tested for all paths possible (path coverage) in the application. And it should be tested using all possible inputs randomly

c. What are three major challenges in testing software components?

Ans. Testing of the software components is very important. But while executing it various challenges come across. Those are as follows-

- **Reusability of component test:** For efficient testing of software components, reuse of component test is very vital. The important factor in reusing component test is to develop systematic methods and tools to prepare test suites to store various test resources such as test cases, test data, test scripts.
- **Construction of testable components:** Software component should be testable with the use of standardized test resources. Constructing the testable component is another challenge. Testable component have below features-
    1. It should be traceable
    2. It must have a set of built-in interfaces to interact with set if testing resources.
    3. It must have well defined test architecture model and built-in test interfaces to support their interactions with component test suits and test beds.
    4. Testable components with built-in tests must use the standard technique to enclose built-in tests.
- **Construction of component test drivers/stubs in a symmetrical way:** In real world practice, engineers use ad-hoc approach. They create product specific test drivers and stubs according to requirements and specification given for that particular component. The disadvantage of this approach is that generated test drivers and stubs are used only for that specific product.

d. Identify the three major differences between white-box testing and black-box testing.
Ans.

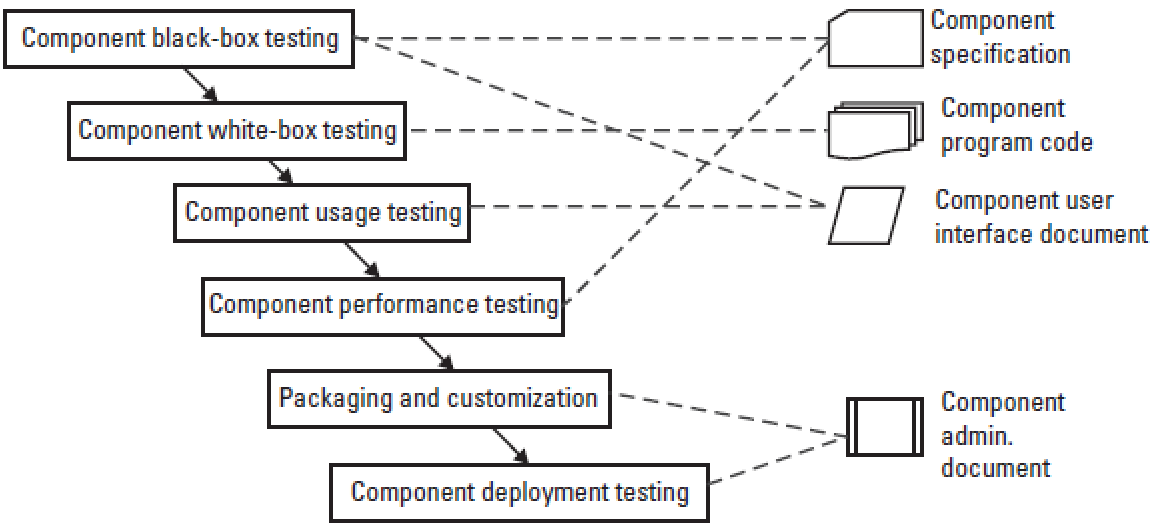
Black-box testing	White-box testing
In Black-box testing, internal structure or the program or the code is hidden and nothing is known about it to the tester.	In White-box testing, the tester has knowledge about the internal structure of the code or the program of the software.
It is a functional/behavioural testing, mostly done by software testers.	It is a structural/logic testing, mostly done by software developers.
No knowledge of implementation and programming is needed.	Knowledge of implementation and programming is required.

e. List vendor-oriented test processes for reusable components.

Ans. Following are the steps-

- **Step 1:** In this step, any traditional black-box testing can be used to check correctness and completeness of an application. It basically checks the functions and behaviours based on the given specifications
- **Step 2:** In this step, developer uses any of the white-box testing techniques to check the internal errors in logic, data objects, data structures
- **Step 3:** In this step, Tester exercises various component usage patterns through component interfaces to final check the correct functions and behaviours
- **Step 4:** In this step, testers and QA engineers all together evaluate the performance
- **Step 5:** In this step, testing is performed on built-in customization features and packaging approaches
- **Step 6:** In this last step, deployment testing is performed to check the design and implementation according to the component model

Following image is taken from a book “Testing and Quality Assurance for Component-Based Software” by Jerry Gao et al.



Question #2: Equivalence Partitioning Testing Questions

(a). List of identified equivalence classes-

<table>
<tr><td>Length of the</td><td rowspan="6">length < 10</td><td colspan="8">10 <= length <= 16</td><td rowspan="6">length > 16</td></tr>
<tr><td>Letter(a-z)</td><td rowspan="5">no letter(a-z)</td><td colspan="7">(a-z)</td></tr>
<tr><td>Digits(0-6)</td><td rowspan="4">no digit(0-6)</td><td colspan="6">(0-6)</td></tr>
<tr><td>Uppercase letter(A-Z)</td><td rowspan="3">no uppercase letter(A-Z)</td><td colspan="5">(A-Z)</td></tr>
<tr><td>Special Char(o,&,.,*,%,$,#)</td><td colspan="2">No special char at all</td><td rowspan="2">(o,&,.,*,%,$,#)</td><td colspan="2">other than (o,&,.,*,%,$,#)</td></tr>
<tr><td>Incorrect/correct password</td><td>Incorrect password</td><td>correct password</td><td>Incorrect password</td><td>correct Password</td></tr>
<tr><td>Partitions</td><td>P0</td><td>P1</td><td>P2</td><td>P3</td><td>P4</td><td>P5</td><td>P6</td><td>P7</td><td>P8</td><td>P9</td></tr>
</table>

(b). Test-cases:

Test case id	Equivalent class partitioning number	Input	Expected Output
00	P0	1. Length of PIN is less than 10 e.g-“america”	“Invalidated Password”
01	P1	1. Length of PIN is from 10 to 16 2. No letter(a-z) e.g- “0123456789”	“Invalidated Password”
02	P2	1. Length of PIN is from 10 to 16 2. Atleast one letter(a-z) 3. No digit e.g- “helloamerica”	“Invalidated Password”
03	P3	1. Length of PIN is from 10 to 16 2. Atleast one letter(a-z) 3. Atleast one digit(0-6) 4. No uppercase letter e.g- “helloamerica1”	“Invalidated Password”
04	P4	1. Length of PIN is from 10 to 16 2. Atleast one letter(a-z) 3. Atleast one digit (0-6) 4. Atleast one uppercase letter(A-Z) 5. No special character 6. Incorrect PIN number e.g- “Helloamerica1”	“Incorrect Password”
05	P5	1. Length of PIN is from 10 to 16 2. Atleast one letter(a-z) 3. Atleast one digit (0-6) 4. Atleast one uppercase letter(A-Z) 5. No special character 6. Correct PIN number e.g- “Helloamerica1”	“Successfully login”

06	P6	1. Length of PIN is from 10 to 16 2. Atleast one letter(a-z) 3. Atleast one digit(0-6) 4. Atleast one special char from (o,&,.,*,%,$,#) e.g- "Helloamerica1*"	"Invalidated Password"
07	P7	1. Length of PIN is from 10 to 16 2. Atleast one letter(a-z) 3. Atleast one digit(0-6) 4. Atleast one special char other than (o,&,.,*,%,$,#) 5. Incorrect PIN number e.g- "Helloamerica@1"	"Incorrect Password"
08	P8	1. Length of PIN is from 10 to 16 2. Atleast one letter(a-z) 3. Atleast one digit(0-6) 4. Atleast one special char other than (o,&,.,*,%,$,#) 5. Correct PIN number e.g- "Helloamerica@1"	"Successfully login"
09	P9	1. Length of PIN is less than 10 e.g- "helloamericahelloamerica"	"Invalidated Password"

Question #3: Boundary Value Testing

(a). Define all boundaries for Z(X, Y) and (Answer shown in the below table)
(b) Define the boundary values for each boundary

Boundary Name	Boundary Range	Boundary values for X	Boundary values for Y
B1	X[1,5] & Y[3,9]	(0,1,2) & (4,5,6)	(2,3,4) & (8,9,10)
B2	X[6] & Y[0]	(5,6,7)	(-1,0,1)
B3	X[7,10] & Y[20,34]	(6,7,8) & (9,10,11)	(19,20,21) & (33,34,35)

(c). Define the test cases for each boundary.

Test cases for B1:

Input value of X (if X has any of these values)	**Input value of Y** (if Y has any of these values)	**Expected Output (Z)** (Put the respective X,Y value in the equation)
(0)	(2,3,4,8,9,10)	-1
(1,2,4,5)	(2,10)	-1
(1,2,4,5)	(3,4,8,9)	2X – 4Y + 5
(6)	(2,3,4,8,9,10)	-1

Test cases for B2:

Input value of X	**Input value of Y**	**Expected Output (Z)**

(if X has any of these values)	(if Y has any of these values)	(Put the respective X,Y value in the equation)
(5)	(-1,0,1)	-1
(6)	(-1,1)	-1
(6)	(0)	Undefined
(7)	(-1,0,1)	-1

Test cases for B3:

Input value of X (if X has any of these values)	**Input value of Y** (if Y has any of these values)	**Expected Output (Z)** (Put the respective X,Y value in the equation)
(6)	(19,20,21,33,34,35)	-1
(7,8,9,10)	(19,35)	-1
(7,8,9,10)	(20,21,33,34)	X + Y + 1
(11)	(19,20,21,33,34,35)	-1

Question #4: Decision table testing question

(a) A decision table (Table A) for sending a reply message without attachments.

		Scenarios			
		1	**2**	**3**	**4**
Conditions	"To" (adding recipient)	T	T	T	F
	Valid "To/CC/BCC"	T	T	F	X
	Filled Subject Line/ Email body	T	F	X	X
Actions	Error				
	Warning				
	Success				

A decision table (Table B) for sending a reply message with attachments.

		Scenarios					
		1	**2**	**3**	**4**	**5**	**6**
Conditions	"To" (adding recipient)	T	T	T	T	T	F
	Valid "To/CC/BCC"	T	T	T	T	F	X
	Filled Subject Line/ Email body	T	T	T	F	X	X
	Suggestive text in email body	T	T	F	X	X	X
	Attachment	T	F	X	X	X	X
Actions	Error						
	Warning						
	Success						

(b). Test cases for Table A:

Test case id	**"To" (recipient)**	**Validation of (To/CC/BCC)**	**Subject line/ Body text**	**Expected output**

00	(Empty)	-	Proper or improper	**Error message:** "Please add recipient"
01	Entered like "akshatadeo"	Invalid recipient	Proper or improper	**Error message:** "Please enter valid email id"
02	Entered like "Akshata.deo@sjsu.edu"	Valid recipients	(Empty)	**Warning message:** "Do you want to send email without subject?"
03	Entered like "Akshata.deo@sjsu.edu"	Valid recipients	Proper subject and body text	**Success message:** "Email has been sent"
04	"To" is empty but CC or BCC is filled	Valid recipients	Proper subject and body text	**Success message:** "Email has been sent"
05	"To" is empty but CC or BCC is filled	Invalid recipient	(Empty)	**Error message:** "Please enter valid email id"

Question #5: AI testing and AI-based software testing questions

(a)

[1] Tosun, Ayse, Ayse Bener, and Resat Kale. "Ai-based software defect predictors: Applications and benefits in a case study." *In Twenty-Second IAAI Conference*. 2010.
[2] D'silva, Vijay, Daniel Kroening, and Georg Weissenbacher. "A survey of automated techniques for formal software verification." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.7 (2008): 1165-1178.
[3] Atif M. Memon, "Using tasks to automate regression testing of GUIs." *In IASTED International Conference on Artificial Intelligence and Applications-AIA*, p. 477-82. 2004.
[4] Nguyen, B.N., Robbins, B., Banerjee, I. and Memon, A., 2014. GUITAR: an innovative tool for automated testing of GUI-driven software. *Automated software engineering,* 21(1), p.65-105.
[5] C. V. Ramamoorthy and S. F. Ho, "Testing large software with automated software evaluation systems," *in IEEE Transactions on Software Engineering*, vol. SE-1, no. 1, pp. 46-58, March 1975.

Paper ID	**Issues and challenges (if there is any)**	**Testing Methods (if there is any)**	**Quality parameters (if there is any)**	**Quality process (if there is any)**	**Supported Testing types and applications**
[1]	Model Calibration requirement Limitation to track predictive performance	Team Software Process (TSP). Naïve Bayes classifier	simplicity accuracy Consistency	Team Software Process (TSP).	General software application
[2]	completeness is only obtainable on very 'shallow' programs	static analysis with abstract domains Model Checking Bounded Model Checking	symbolic analysis Abstraction Concurrency	-	C,C++,Ada
[3]	Regression test selection problem coverage identification problem	Regression	cost of testing	-	GUI

[4]	Requires manually specifying the ignored and terminal widgets in a UI.	GUITAR Framework	Modularity Event Selection oracle specification	-	GUI
[5]	tools not well-structured machine dependent tools	selective testing exhaustive testing formal proofs	correctness Performance	-	large software systems

(b)

[1] Leofante F, Narodytska N, Pulina L, Tacchella A. Automated verification of neural networks: Advances, challenges and perspectives. arXiv preprint arXiv:1805.09938. 25 May 2018.
[2] Van Wesel, Perry, and Alwyn E. Goodloe. "Challenges in the verification of reinforcement learning algorithms." (2017).
[3] Fei Liu and Ming Yang, "Verification and validation of AI simulation systems," *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826), Shanghai, China*, p. 3100-3105 vol.5, 2004.
[4] Seshia SA, Sadigh D, Sastry SS. Towards verified artificial intelligence. arXiv preprint arXiv:1606.08514. 27 June 2016.
[5] Xie, X., Ho, J.W., Murphy, C., Kaiser, G., Xu, B. and Chen, T.Y., 2011. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software*, 84(4), p.544-558.
[6] Murphy, Christian, Gail E. Kaiser, and Marta Arias. "An approach to software testing of machine learning applications." (2007).

Paper ID	Issues and challenges (if there is any)	Testing Methods (if there is any)	Quality parameters (if there is any)	Quality process (if there is any)	Supported Testing types and applications
[1]	Coordinate the efforts of two “separated at birth” AI communities: Machine Learning and Automated Reasoning. Mate precision with scalability	CDCL-style SAT solving algorithm Satisfiability Modulo Theories (SMT) Mixed Integer Linear Programming (MIP)	Local-Invariance, Global-Invariance, Invertibility, Equivalence	State-of-the-art design	BNN, DNN(ReLU), DNN(ReLU+Pooling), DNN, NN
[2]	Assumptions of operating environments Assumptions of Platform (Failure and Fidelity) Assumption of Data	Formal Verification Runtime Verification (RV) Thrun's Validity Interval Analysis	State to action mappings, Action sequences, Algorithm properties independent of data, Validating	-	reinforcement learning algorithms

	(Distribution, samples etc.) Algorithm assumption		assumptions on training data,		
[3]	System representation assumption The efficacy of the inductively learnt model can never be better than the quality of the input examples.	logic formalisms graph theory, empirical, heuristic, formal methods, and optimization Probability methods, gray theory, turing test and expert judgement Runtime verification tools (Java Path Finder)	Completeness, conflicts, redundancies, Nondeterminism	-	Rule-based systems Knowledge-based systems Nondeterministic systems Complex software Adaptive systems
[4]	Environment Modeling Formal Specification Modeling Systems that Learn Computational Engines for Training, Testing, and Verification	Boolean satisfiability (SAT) solvers Binary Decision Diagrams (BDDs) satisfiability modulo theories (SMT) solvers Formal Verification Methods	Effectiveness, Scalability	-	General AI methods
[5]	The oracle problem	Metamorphic testing: - It uses properties of function properties to predict and analyse the changes resulting in the output by implementing the changes in the input test cases. Cross-validation Analysis	Average violation percentage of all MR (Metamorphic Relation) Performance Effectiveness Mutation analysis (for Bio-informatics)	-	supervised classification algorithms: - k-Nearest Neighbors (kNN) Naïve Bayes Classifier (NBC)

[6]	Negative handles could not be addressed. Repeated or missing attribute values are not handled properly. It does not provide how many non-failures should be made each partition.	SVM-Light MartiRank ranking Algorithm	Effectiveness, Scalability		SVM