

**كتاب**

**دردشة في التيسير**

**إعداد**

**أحمد عثمان**

Ahmed etman

## إهادء

إلى كل من علمنى كيف أكون Quality Control محترف... إلى من كان السبب - بعد مشيئة الله سبحانه وتعالى - فى وضع قدمى فى عالم ال Testing وألتمس أولى خطواتى فيه:

م / أمانى شوشة

م / إيمان عبد الحميد

م / أحمد طه

م / خالد نبيل

م / أحمد عبد السلام

م / سعد ذكى

أتوجه إليكم جميعا بجزيل الشكر والتقدير والعرفان لما قدموه لي... سائل المولى عز وجل أن يجعل هذا العمل في ميزان حسناتكم.

وتقديرا لما قدموه في سبيل الوصول لما أنا عليه الآن، أهدي إليكم هذا الكتاب، راجيا من الله أن ينال إعجابكم واعجاب كل قارئ لهذا الكتاب...

أحمد عثمان

25 يناير 2018

## Index

Items	Page No.
<u>مقدمة</u>	14
<u>؟testing يعني ايه</u>	14
<u>؟ISTQB ايه ال</u>	14
<u>حلوة وظيفة التester يا عمه</u>	15
<u>لية العالم كله بقى مهتم بوظيفة ال software tester</u>	15
<b>Part 1: International Testing Qualifications Board (ISTQB) Syllabus</b>	17
<b>Chapter 1: Fundamentals of Testing</b>	
<b>1.1: Why is Testing Necessary?</b>	
<u>1.1.1: Software Systems Context</u>	18
<u>1.1.2: Causes of Software Defects</u>	18
<u>1.1.3: Role of Testing in Software Development, Maintenance and Operations</u>	20
<u>1.1.4: Testing and Quality</u>	20
<u>1.1.5: How Much Testing is Enough?</u>	21
<b>1.2: What is Testing?</b>	21
<u>Dynamic testing</u>	22
<u>Static testing</u>	22
<u>Objectives of testing</u>	22
<u>Maintenance &amp; Operational testing</u>	23
<u>Debugging and testing</u>	23
<b>1.3: Seven Testing Principles</b>	23

Items	Page No.
<b><u>1.4: Fundamental Test Process</u></b>	<b>26</b>
<b><u>Test process main activities</u></b>	<b>26</b>
<b><u>1.4.1: Test Planning and Control</u></b>	<b>27</b>
<b><u>1.4.2: Test Analysis and Design</u></b>	<b>27</b>
<b><u>1.4.3: Test Implementations and execution</u></b>	<b>28</b>
<b><u>1.4.4: Evaluating Exit Criteria and Reporting</u></b>	<b>29</b>
<b><u>1.4.5: Test Closure Activities</u></b>	<b>30</b>
<b><u>1.5: The psychology of Testing</u></b>	<b>31</b>
<b><u>1.6: Code of Ethics</u></b>	<b>33</b>
<b>Chapter 2: Testing Throughout the Software Life Cycle</b>	
<b><u>2.1: Software Development Models</u></b>	<b>35</b>
<b><u>2.1.1: V-model (Sequential Development Model)</u></b>	<b>35</b>
<b><u>2.1.2: Iterative-incremental Development Models</u></b>	<b>36</b>
<b><u>2.1.3: Testing within a Life Cycle Model</u></b>	<b>37</b>
<b><u>2.2: Test Levels</u></b>	<b>37</b>
<b><u>2.2.1: Component Testing</u></b>	<b>38</b>
<b><u>2.2.2: Integration Testing</u></b>	<b>39</b>
<b><u>2.2.3: System Testing</u></b>	<b>41</b>
<b><u>2.2.4: Acceptance Testing</u></b>	<b>42</b>
<b><u>2.3: Test Types</u></b>	<b>45</b>
<b><u>2.3.1: Testing of function (Functional Testing)</u></b>	<b>45</b>
<b><u>2.3.2: Testing of Non-functional Software Characteristics (Non-functional Testing)</u></b>	<b>46</b>
<b><u>2.3.3: Testing of Software Structure/Architecture (Structural Testing)</u></b>	<b>47</b>
<b><u>2.3.4: Testing Related to changes: Re-testing and Regression Testing</u></b>	<b>47</b>
<b><u>2.4: Maintenance Testing</u></b>	<b>48</b>

Items	Page No.
<b>Chapter 3: Static Techniques</b>	
<b><u>3.1: Static Techniques and the Test Process</u></b>	<b>50</b>
<u>Static testing techniques</u>	51
<u>Types of static testing techniques</u>	51
<b><u>3.2: Review Process</u></b>	<b>52</b>
<u>3.2.1: Activities of Formal Review</u>	52
<u>3.2.2: Roles and Responsibilities</u>	54
<u>3.2.3: Types of Reviews</u>	54
<u>3.2.4: Success Factors for Reviews</u>	57
<b><u>3.3: Static Analysis by Tools</u></b>	<b>58</b>
<b>Chapter 4: Test Design Techniques</b>	
<b><u>4.1: The Test Development Process</u></b>	<b>60</b>
<u>Test conditions</u>	60
<b><u>4.2: Categories of Test Design Techniques</u></b>	<b>62</b>
<b><u>4.3: Specification-based or Black-box Techniques</u></b>	<b>63</b>
<u>4.3.1: Equivalence Partitioning</u>	63
<u>4.3.2: Boundary Value Analysis (BVA)</u>	64
<u>4.3.3: Decision Table (DT) Testing</u>	64
<u>4.3.4: State Transition Testing</u>	65
<u>4.3.5: Use Case Testing</u>	66
<b><u>4.4: Structure-based or White-box Techniques</u></b>	<b>66</b>
<u>4.4.1: Statement Testing and Coverage</u>	67
<u>4.4.2: Decision Testing and Coverage</u>	67
<u>4.4.3: Other Structure-based Techniques</u>	68
<b><u>4.5: Experience-based Techniques</u></b>	<b>69</b>
<u>Error guessing &amp; Fault attack</u>	69
<u>Exploratory testing</u>	69
<b><u>4.6: Choosing Test Techniques</u></b>	<b>70</b>

Items	Page No.
<b>Chapter 5: Test Management</b>	
<b>5.1: Test Organization</b>	
<u>5.1.1: Test Organization and Independence</u>	71
<u>5.1.2: Tasks of the Test Leader and Tester</u>	72
<b>5.2: Test Planning and Estimation</b>	
<u>5.2.1: Test Planning</u>	74
<u>5.2.2: Test Planning Activities</u>	75
<u>5.2.3: Entry Criteria</u>	76
<u>5.2.4: Exit Criteria</u>	76
<u>5.2.5: Test Estimation</u>	76
<u>5.2.6: Test Strategy, Test Approach</u>	77
<b>5.3: Test Progress Monitoring and Control</b>	
<u>5.3.1: Test Progress Monitoring</u>	78
<u>5.3.2: Test Reporting</u>	79
<u>5.3.3: Test Control</u>	79
<b>5.4: Configuration Management</b>	80
<b>5.5: Risk and Testing</b>	80
<u>5.5.1: Project Risks</u>	81
<u>5.5.2: Product Risks</u>	82
<b>5.6: Incident Management</b>	83
<b>Chapter 6: Tool Support for Testing</b>	
<b>6.1: Types of Test Tools</b>	
<u>6.1.1: Tool Support for Testing</u>	85
<u>6.1.2: Test Tool Classification</u>	86
<u>6.1.3: Tool Support for Management of Testing and Tests</u>	87
<u>6.1.4: Tool Support for Static Testing</u>	87
<u>6.1.5: Tool Support for Test Specification</u>	88
<u>6.1.6: Tool Support for Test Execution and Logging</u>	89

Items	Page No.
<u>6.1.7: Tool Support for Performance and Monitoring</u>	90
<u>6.1.8: Tool Support for Specific Testing Needs</u>	90
<b>6.2: Effective Use of Tools: Potential Benefits and Risks</b>	
<u>6.2.1: Potential Benefits and Risks of Tool Support for Testing (for all tools)</u>	91
<u>6.2.2: Special Considerations for some Types of Tools</u>	92
<u>6.3: Introducing a Tool into an Organization</u>	93
<b>Part 2: Testing Practices</b>	95
<b>Day 1</b>	
<b>Software Development models</b>	96
<u>Sequential models</u>	97
<u>Waterfall model</u>	97
<u>V-model</u>	98
<u>Validation &amp; Verification</u>	99
<u>Iterative Model</u>	100
<u>Incremental Model</u>	100
<u>Agile Model</u>	101
<b>Testing within a Life Cycle Model</b>	102
<u>Different stages of SDLC with STLC</u>	102
<b>Definition of testing</b>	104
<u>Testing Misconceptions</u>	104
<b>Day 2</b>	
<b>Testing and Quality</b>	105
<u>Software Quality</u>	105
<u>Testing, QA, and QC</u>	105
<u>Example</u>	106
<b>Testing and Debugging</b>	108

Items	Page No.
<b>Day 3</b>	
<u>Developing vs. Testing.... A way of thinking</u>	109
<u>Developers vs. Testers</u>	109
<u>Test-to-pass and test-to-fail</u>	110
<u>Objectives of a software tester</u>	110
<b>Day 4</b>	
<u>What is a "bug?"</u>	111
<u>Why we call them "Bugs?"</u>	111
<u>Actual Result vs Expected Result</u>	112
<u>Other sources of Expected Results</u>	112
<u>Life experience</u>	112
<u>Example</u>	113
<u>Common sense</u>	113
<u>Communication</u>	113
<u>Standards</u>	113
<u>Example 1</u>	113
<u>Example 2</u>	114
<u>Statistics</u>	114
<u>Valuable Opinion</u>	114
<u>Other sources</u>	114
<b>Day 5</b>	
<u>Software bugs and spec bugs</u>	115
<u>Causes of software defects</u>	115
<u>The human factor</u>	116
<u>Organizational factors</u>	116
<u>Environmental conditions</u>	116
<u>Example</u>	117

Items	Page No.
<b>Day 6</b>	
<u>What are the "right" things to test?</u>	118
<u>Fail is not always a bug</u>	118
<u>Test levels</u>	119
<u>1- Unit testing</u>	119
<u>2- Integration testing</u>	120
<u>Integration testing types</u>	120
<u>1. Big Bang testing</u>	120
<u>2. Top down testing</u>	121
<u>3. Bottom up testing</u>	122
<u>Stups &amp; Drivers</u>	123
<u>Stups</u>	123
<u>Example</u>	124
<u>Drivers</u>	125
<u>Example</u>	125
<u>Summary</u>	126
<u>3- System Testing</u>	127
<u>4- Acceptance testing</u>	128
<u>Acceptance testing types</u>	129
<u>1. User acceptance testing</u>	129
<u>2. Operational (acceptance) testing</u>	129
<u>3. Contract and regulation acceptance testing</u>	129
<u>4. Alpha and beta (or field) testing</u>	130
<u>5- Maintenance testing</u>	130
<u>Re-testing</u>	131
<u>Regression testing</u>	131

Items	Page No.
<b>Day 7</b>	
<u>Test types</u>	132
<u>Test functionality</u>	132
<u>1- Functional testing</u>	132
<u>2- Non-functional testing</u>	133
<u>Testing Methodologies</u>	134
<u>1- Black box testing</u>	134
<u>2- White box testing</u>	134
<u>3- Experience based testing</u>	135
<u>Summary</u>	136
<b>Day 8</b>	
<u>Black box test design techniques</u>	137
<u>1- Equivalence Partitioning (EP)</u>	137
<u>Example</u>	137
<u>2- Boundary Value Analysis (BVA)</u>	138
<u>Example</u>	138
<u>3- Decision Table</u>	139
<u>Example</u>	139
<u>Example 2</u>	143
<u>Example 3</u>	146
<u>4- State Transition</u>	148
<u>Example</u>	149
<u>Example 2</u>	150
<u>5- Use Case Testing</u>	151
<u>Example</u>	151
<u>Example 2</u>	153

Items	Page No.
<u>White box techniques</u>	154
<u>1- Statement Coverage</u>	154
<u>Example</u>	154
<u>2- Decision Coverage</u>	156
<u>Example</u>	156
<u>Choosing the right technique</u>	157
<u>How much testing is enough?</u>	158
<b>Day 9</b>	
<u>Test Case</u>	159
<u>Test Case attributes</u>	159
<u>Example</u>	163
<u>Maintainability of test cases</u>	163
<u>Measures we took to improve the maintainability of test cases</u>	164
<u>Example</u>	165
<u>The number of expected Results inside one test case</u>	167
<u>Example</u>	167
<u>Bad Test Case Practices</u>	168
<u>Example</u>	168
<u>Data-Driven Test Cases</u>	171
<u>Test Case with Assumption</u>	172
<u>Example</u>	172
<u>Test Suites</u>	173
<u>Test Suite Items</u>	173
<u>Marrying Test Suites</u>	174
<u>Checklists</u>	174

Items	Page No.
<b>Day 10</b>	
<a href="#"><u>Bug Reports</u></a>	176
<a href="#"><u>Bug Report components</u></a>	176
<a href="#"><u>Example</u></a>	178
<a href="#"><u>Bug Life Cycle</u></a>	179
<a href="#"><u>Sanity Testing</u></a>	180
<a href="#"><u>Smoke Testing</u></a>	180
<a href="#"><u>Example</u></a>	180
<a href="#"><u>Testware</u></a>	181
<a href="#"><u>Test Harness</u></a>	181
<a href="#"><u>Example</u></a>	181
<b>Part 3: ISTQB Exams</b>	183
<a href="#"><u>ISTQB Official sample exam (Version 2.9)</u></a>	184
<a href="#"><u>ISTQB Official sample exam Answers (Version 2.9)</u></a>	208
<a href="#"><u>ISTQB Exam 2</u></a>	228
<a href="#"><u>ISTQB Exam 2 Answers</u></a>	243
<a href="#"><u>ISTQB Exam 3</u></a>	244
<a href="#"><u>ISTQB Exam 3 Answers</u></a>	259
<a href="#"><u>ISTQB Exam 4</u></a>	260
<a href="#"><u>ISTQB Exam 4 Answers</u></a>	275
<a href="#"><u>خاتمة</u></a>	276
<a href="#"><u>References</u></a>	277

## بسم الله الرحمن الرحيم

### مقدمة:

أهلاً بيكم في عالم الـ **testing**، هنا هنتعلم إن شاء الله إزاي تكون كيو سى (QC) (Quality Control) أو تيستر tester محترف.

في الكتاب ده هنبدأ مع بعض من الصفر لغاية مانوصل لمرحلة إنك تبقى تيستر محترف وقدر تعدى امتحان الـ **ISTQB - Foundation Level** إن شاء الله، كمان هنتعلم إزاي تطبق اللي اتعلمنه ده على الشغل بتاعك، لأنني شايف - من وجهة نظرى - إن تعليم الـ **ISTQB** لوحده مش كفاية، حتى لو اتعلمت المنهج ودخلت الامتحان ونجحت وخدت الشهادة هيفضل برضه عندك **gap** كبيرة بين اللي اتعلمنه واللي هتشتغل بيها، فهنا إن شاء الله هنحاول نعالج النقطة دى.

### طب ف الأول يعني إيه **testing**؟

الـ **testing** أو الـ **QC (Quality Control)** (الاتنين بيعبروا عن نفس الشئ) هى عملية بتمحور حواليين إنك تطلع الأخطاء من أى برنامج سوفتوير software أثناء مراحل الـ developing المختلفة اللي بيتعمل فيها البرنامج ده (هنعرف بعدين المراحل دي).

والشخص اللي بي عمل الكلام ده بيبقى اسمه **Quality Control** أو **tester**، وده فى الفترة الحالية بقى طبعه فى سوق السوفتوير فى العالم كله career path.

وعشان تبقى تيستر محترف وملم بكل تفاصيل الشغلانة بتاعتك لازم تبقى certified من **ISTQB** عشان تبقى متعلم كل حاجة تخص التيسينج وكمان عشان تقدر تلاقى شغل بسهولة إن شاء الله.

### طب إيه الـ **ISTQB**؟

الـ **ISTQB** اختصار **International Software Testing Qualifications Board** ودى عباره عن منظمه بلجيكية تعتبر هى الـ **standard** العالمي لأصول وقواعد التيست، والمنظمه دى بتدى شهادات عالمية في التيسينج testing لأى تيستر محترف طبقاً للمنهج اللي هذرجه في الكتاب إن شاء الله وبعد كده يدخل على الامتحان ولما يعديه يقدر ياخ شهادة الـ **ISTQB - Foundation Level**، طبعاً فيه شهادات تانية كتير بتتلها **ISTQB** level، لكن ما فيش ولا level منهم تقدر تدرسه أو تمتحنه إلا لما تكون واحد الـ **Foundation Level** الأول، واللي حالياً انت تحتاجه كتيستر مبتدئ عشان تقدر تبدأ الـ **career path** بتاعك.

عشان تعرف أكثر عن الـ **ISTQB** ادخل ع السايت بتاعهم: <http://www.istqb.org>



## حلوة وظيفة التيسير يا عمه؟

بص أنا مش هضحك عليك واقولك إن الحياة لونها بمبي والجو ده، لكن فعلياً الـ **Software Testing** بقت من أكثر الوظائف المطلوبة في العالم كله الحمد لله، كل الحكاية انت بس تحتاج تكون إنسان صبور جداً! وتهتم بكل التفاصيل بتاعة الـ **project** اللي بيتعمل، عشان تقدر تحلى أكبر قدر من المشاكل قبل ما تسلم الـ **project** ده للعميل، وبالتالي البرنامج بتاعك تكون الـ **quality** بتاعته عالية (هنجي للنقطة دي كمان شوية، إتقل بس).

## طب ليه العالم كله فجأة بقى مهتم بوظيفة الـ **software tester** للدرجة دي؟

إن سوق السوفتوير في العالم كله اتطور بشكل رهيب، والاهتمام بالـ **quality** بتاعة البرنامج بتاعك بقى شيء أساسى من عوامل نجاحه وانتشاره وتميزه من بين آلاف البرامج الثانية اللي بتتعمل زييه وبتتأدى نفس الشغل بتاعة.

كمان أحياناً الغلطات الصغيرة في البرامج بتخسر الشركة كتير جداً، وبتكون تكاليف الخطأ ده كبيرة جداً.  
إزاي؟



في مايو 1992 م شركة بيبسي عملت عرض في الفلبين، العرض ده إن اللي يلاقى الرقم **349** مطبوع على ضهر غطاً أى إزازة هيشرتها الزبون هيكتب مليون بيسو (العملة المحلية للفلبين) (المبلغ كان يعمل حوالي 40,000 دولار أمريكي وقتها)، المفروض إن البرنامج اللي بيطبع الأرقام على غطيان الأزاييز يطبع الرقم 349 على إزازة واحدة في كل 800,000 إزازة، لكن بسبب خطأ في السوفتوير اللي بيطبع الأرقام **اتعكس الموضوع** فتم طباعة الرقم 349 على الـ 800,000 إزازة إلا إزازة واحدة، طبعاً الزباين مالهمش دعوة بالكلام ده وعايزين يكسبو العرض اللي نزلته الشركة، واترتفعت قضايا ودفعت بيبسي ف الحوار ده **42 مليار دولار**، وده كله بسبب غلطة في برنامج!

رقم زى ده ممكن يخلى إى شركة تشهر إفلاسها فوراً بسبب غلطة برنامج... لكن بيبسي قدراتها المالية والتسويقية في العالم كله قدرت تغطي المبلغ ده.



مثال تانى... فى يوم 2 أغسطس 2016 أعلنت سامسونج عن تليفون **Galaxy Note7** ونزلته للسوق يوم 19 أغسطس 2016... كان من المتوقع إن التليفون يعمل ثورة ضخمة فى عالم الموبايلات لإنه نازل بمواصفات خيالية وقتها... لكن بعد 15 يوم من عرض التليفون ف السوق قررت شركة سامسونج سحب التليفون رسمياً من السوق ووقف بيعه، وده بعد شكاوى المستخدمين المستمرة بإن بطارية التليفون بتتفجر بسبب عيب تصنيع مخل بالبطارية تتفجر، بسبب الموضوع ده خسرت سامسونج حوالي **17 مليار دولار** وانخفضت مبيعات الشركة فى الرابع الثالث من 2016 بنسبة **33%** مقارنة بالربع الثاني من 2016، وده غير سمعتها اللي انضربت ف السوق، واللى لو لا إن سامسونج كانت منزلة أكثر من منتج فى وقتها كانت الشركة حرفياً أفلست أو ع الأقل فقدت ثقة العملاء بتوعها اللي مخلينها من أكبر الشركات مبيعاً لل **smart phones** في العالم.

تخيلوا لو كانت سامسونج عملت تيست زى البنى آدمين ع التليفون قبل ماينزل السوق هل كانت خسرت كل الخساير دى؟ بالتأكيد لأ.



وده أكبر دليل على إن مجال **testing** ده مجال مهم جداً  
مهما كان نوع المنتج اللي بينزل، فلو انت لغاية دلوقتي مش  
مقطوع بالوظيفة او لاقيت حد قدامك بيقل من وظيفة التيسير  
بع الحالتين اللي فوق دول بتوع ببىسى وسامسونج،  
وتخيل لو انت كنت بتعمل تيست في الحالتين دول مش كنت  
هتمنع كارثة بمليارات؟ أكيد طبعاً.

بقى شغلانة التيسير مهمة ولا مش مهمة؟

الأرقام اللي فوق دى أكبر دليل إنها مهمة جداً ومؤثرة جداً وإنها مش وظيفة **level 2<sup>nd</sup>** وغيرها من الكلام  
اللى بيقل من قيمتها.

كده إحنا خلصنا كلامنا عن التيسينج بشكل عام... تعالوا بقى ندخل في التفاصيل...

### بعض ياسيدى... إحنا هنشتغل في الكتاب ده على 3 parts

**:Part 1** وده خاص بشرح المنهج الرسمي لل **ISTQB**.

**:Part 2** وده هنتعلم فيه إزاي نشتغل بالكلام اللي اتعلمناه فعلياً ونطبقه ف شغلنا على حالات عملية وكده.

**:Part 3** ده عباره عن نماذج من امتحان **ISTQB** (منهم الامتحان الرسمي اللي ع الموقعي بتاعهم).

بعد الـ 3 أجزاء دول ه تكون مؤهل ان شاء الله انك تأخذ شهادة ال **ISTQB** وتشتغل ك **tester** محترف.

يلا بينا ندخل في الجد بقى.

Part 1

**International Software Testing  
Qualifications Board (ISTQB) Syllabus**

# Chapter 1

## Fundamentals of Testing

### Why is Testing Necessary? :1.1

#### Software Systems Context :1.1.1

من المعروف إن برامج السوفتوير بقت جزء ضروري من حياتنا اليومية، سواء كانت البرامج دى برامج لبيزنيس معين زى البنوك مثلاً أو برامج لمنتجات بيستعملها الناس زى العربيات مثلاً، معظم الناس واجهت مشاكل مع البرامج إنها مش شغالة زى ما هو متوقع، وأخطاء السوفتوير دى بتنقاوت مشاكلها فممكن تسبب خسائر مالية أو ضياع الوقت أو لسمعة الشركة (زى ما قلنا قبل كده فى المقدمة) أو حتى ممكن تسبب الإصابة أو الوفاة للليوزر (زى برامج الطيارات مثلاً لو حصل فيها أى خلل ممكن الطيارة كلها تقع).

#### Causes of Software Defects :1.1.2

بيتكلم الكتاب فالجزء ده عن أسباب حدوث defects، فيقولك إن الإنسان ممكن يعمل خطأ error معين (زى مثلاً يحط إشارة بالسالب بدل الموجب) ينتج عنه مشكلة Defect (fault or bug) فى كود البرنامج (زى مثلاً انه هي عمل عملية طرح بدل الجمع)، ولو اتنفذ الكود ده السيسنتم ممكن ماي عملش الوظيفة المطلوبة منه أو يعمل حاجة المفروض ماي عملهاش، وده بيسبب فشل failure للعملية كلها (إن البرنامج فشل فى تنفيذ عملية الجمع اللي كان المفروض يطلعها).

ممكن الـ defects اللي فى السوفتوير أو الـ systems أو حتى الـ documents تكون سبب فى الـ failure اللي بيحصل للعملية كلها، لكن مش كل الـ defects بتعمل كده.

الشكل الجاي ده هيووضح الدنيا أكثر



طبعاً لازم ناخذ بالنا من الكلمات دى كويس جداً لأنها بتتجى فى امتحانات الـ ISTQB زى ما هي كده، عشان كده لازم نركز أوى ع الفرق بين الـ **Error** والـ **Bug** والـ **Failure**.

ال Defects زى ما قلنا انها بتحصل لإن الإنسان عرضة للخطأ، لكن فيه أسباب تانية لل Defects، فمن ضمن أسباب حدوث ال Defects:

• **Time Pressure**: ممكن مع ضغط الوقت أعمل حاجة بسرعة ومش واحد بالى من الخطأ اللي عملته ده.

• **Complex Code**: ممكن الكود بتاعي يبقى معقد زى ال spaghetti code مثلا، فنلاقى سطر كود يودينى لحنة معينة من غير مأخلص الحنة الأولانية ويبقى سطر يدخلنى فى حنة تالتة من غير مأخلص execution لباقي الاثنين اللي فاتوا وهكذا... فوارد جدا مع التعقيد اللي ف الكود ده إإن يحصل defect.

• **Complexity of Infrastructure**: ممكن يكون التعقيد فى ال Infrastructure نفسه بتاعة البرنامج معقدة، وممكن جدا يحصل defect فى الدنيا الملحبوطة دى.

• **Changing Technologies**: كمان ممكن ال defect يحصل مع تغيير tools اللي بنشتغل بيها او بنعتمد عليها فى برمجنا او حتى لو حصل update لحاجة معينة وهكذا.

• **Many System Interactions**: ممكن السيستم بتاعي بيتعامل مع systems تانية كتير او ببستدعى functions كتير من اماكن مختلفة، وده ممكن يخلى ال defect تحصل مع زيادة ال Interactions فى السيستم.

### أما ال Failures فممكن تحصل بسبب ظروف بيئية زى مثلا:

• **Radiations**: ممكن الإشعاعات تأثر على أداء برامج معينة فماتشتنغلش بالشكل المطلوب.

• **Magnetism**: ممكن المجال المغناطيسي يؤثر على أداء برنامج معين زى برامج البوصلة مثلا، وبالتالي يحصل failure فى تنفيذ العملية.

• **Electronic Fields**: ممكن الإشارات الإلكترونية تأثر على شغل البرنامج وتعمل failure فى الأداء بتاعه، عشان كده لما تركب أى طياره بيطلب الطيار من الركاب قفل تليفوناتهم عشان المجالات الإلكترونية لإشارات التليفونات ماتأثرش على أجهزة الملاحة والأجهزة الثانية عنده فى الطيارة وتشوش على الإشارات الإلكترونية اللي بيعتها لأبراج المراقبة على الأرض.

• **Pollution**: ممكن تلوث الجو يؤثر على شغل البرنامج زى برامج الأرصاد الجوية مثلا، فلو كان الجو ملوث حوالين ال sensor بتابع جهاز القياس ممكن يطلع نتيجة مختلفة تماما عن الواقع.

كل الحاجات دى ممكن تسبب Failure ف السيستم بسبب انها ممكن تأثر على الهايدوير Hardware بتابع الجهاز فطبعاً السوفتوير بيتأثر شغله.

## Role of Testing in Software Development, Maintenance and :1.1.3 Operations

بيقولك ان التيسننج الاصارم documentation systems وال systems للrigorous testing كمان ممكن يساعد فى تقليل risk المشاكل اللي ممكن تحصل أثناء تنفيذ العملية وبيساهم بالتأكد فى رفع مستوى الجودة بتاعة ال software system فى حالة لو لقينا defects وصلاحناها قبل مانعمل للسيستم ويتسلم للعميل.

كمان ممكن يكون التيسينج مطلوب عشان يحقق شروط معينة فى التعاقد أو ف القانون أو معايير معينة للصناعة اللي السوفتوير معمول عشانها.

## Testing and Quality :1.1.4

إحنا بمساعدة التيسينج نقدر نقىس مدى جودة السوفتوير بتاعى عن طريق ال defects اللي بلاقيها فى functional and non-Functional requirements and characteristics (e.g., .(Reliability, usability, efficiency, maintainability and portability

ماتفتش م الكلام ده كله non-functional testing هنجيله فى شابت 2، ولو حابب تعرف أكثر عن ال software engineering سيرش على - 'Software Product Quality' (ISO 9126)

التسينج كمان بيدينا الثقة فى جودة السوفتوير لو طلعننا defects قليلة أو ماطلعناش خالص، فالتسننج المعموله ديزاين design صح وبنجح من غير defects بيقلى من مستوى risk الريسك الكلى فى السيستم، ولو التيسينج لقى defects ال quality defects بتاعة ال software system بتزيد لما defects دى تحل.

كمان عشان نعلى ال quality بتاعة السيستم لازم نتعلم من أخطائنا فى المشاريع اللي كانت قبل كده، فلازم نفهم الأسباب الأساسية اللي سببت فى ال defects اللي لاقيناها فى المشاريع السابقة، وبالتالي نتجنب الأخطاء دى ف المشاريع الجاية وبالتالي تتحسن جودة البرامج اللي هتتعمل فى المستقبل، ده وجه من وجه ال Quality Assurance (QA) **انتا بنمنع الأخطاء قبل ما تحصل** (فى الجزء الثاني هنشرح الحنة دى بالتفصيل إن شاء الله).

وعشان نزود فى ال Quality بتاعة السيستم كمان لازم نخلى ال Testing خطوة أساسية فى عمليات development standards, training, and defect Quality Assurance .(analysis

### How Much Testing is Enough? : 1.1.5

الجزء ده مهم جداً فلازم تاخذ بالك منه ...

بيقولك إن عشان تقرر إمتي توقف تيسينج لازم تحط ف اعتبارك معايير معينة زى **level of risk** (technical, safety, and business risks) والـ **project constraints** زى **time budget** والميزانية **time** بتاعة المشروع، عامة الكلام ده هيترح أكثر في شابت 5 ان شاء الله.

التيستينج لازم يمد صناع القرار وأصحاب المصلحة **stakeholders** بالمعلومات الكافية عشان ياخدوا قرار بخصوص السيستم اللي بيتعمله تيست ده هيسلموه للعميل ولا لسه فيه حاجات هتتعدل، وده معيار مهم جداً في إنى أوقف تيسينج أو لا.

### What is Testing? : 1.2

#### :Background

المعروف عن عملية التيسينج إنها تنفيذ التيست بس عن طريق عمل **execution** للبرنامجه وتطبيقي **test case** عليه، ف الحقيقة ده جزء من عملية التيسينج مش العملية كالم.

التيستينج موجودة قبل الـ **test execution** وبعده، **الـ activities** دى شاملة:

- Planning and control
- Choosing test conditions
- Designing and executing test cases
- Checking Results
- Evaluating exit criteria
- Reporting on the testing process and system under test
- Finalizing or completing closure activities after a test phase has been completed.

التيستينج كمان شامل مراجعة الـ **Static documents** (including source code) وعمل الـ **Analysis** (هنجيلاهم ف وقتهم ماتفتقش).

بيقولك إن الـ **Static Testing** والـ **Dynamic Testing** ممكن يستخدموا كوساييل لتحقيق أهداف متتشابهة، وكمان يقدروا يقدموا المعلومات اللازمة اللي ممكن نستخدمها لتحسين السيستم اللي بيتعمله تيسينج وكمان تحسين عمليات الـ **development** والـ **testing**.

طب ايه الفرق بين الـ **dynamic testing** والـ **static testing** ؟

### الـ **Dynamic testing**

- هو الـ **execution** الفعلى اللي بعمله على الـ **components** فـ **الـ system**.
- بنتأكـ فيـه إنـنا بـنـتـحـكم فـ الـ **environment** بتـاعـة البرـنـامـج.

### اما الـ **Static testing**

- بـتشـيك عـ الدـنـيـا منـ غـير مـاتـنـفـذـا عـمـلـيـة.
- الـ **reviews** بالـورـقة والـقـلم أـبـرـزـ مـثـالـ.

### لـ **objectives** دـى:

- إـنـى أـلـاقـى أـلـأـخـطـاء **Finding Defects**
- اـكتـسـابـ الثـقـةـ فـي مـسـتـوـيـ الـجـودـةـ **Gaining confidence about the level of quality**
- تـقـدـيمـ الـمـعـلـومـاتـ الـلـازـمـةـ لـصـنـاعـ الـقـرارـ **Providing information for decision-making**
- مـنـعـ الـأـخـطـاءـ **Preventing defects**

عملية التفكير والـ **activities** المشتركة في تصميم الـ **test** بـدرـى فـي الـ **life cycle** (التـأـكـدـ منـ الـ **test basis** أـثنـاءـ الـ **test design**) مـمـكـنـ تسـاعـدـ فـي مـنـعـ الـ **defects** منـ إـنـهاـ تحـصـلـ فـيـ الـ **code**، كـمـانـ مـراـجـعـةـ الـ **requirements documents** (زـىـ الـ **issues** مـثـلاـ) وـالـتـعـرـفـ عـلـىـ الـ **issues** وـلـهـاـ بـتسـاعـدـ لـمـنـعـ الـ **defects** الليـ بـتـظـهـرـ فـيـ الـ **code**.

**ملحوظة:** الـ **test basis** هـىـ أـىـ **sources** للـ **system** الليـ هـشـتـغـلـ عـلـيـهـ (أـىـ حـاجـةـ فـيـهـ مـعـلـومـاتـ عـنـ الـ **system**).

الـ **testing** مـمـكـنـ يـبـقـىـ لـهـ أـكـثـرـ مـنـ **objective** حـسـبـ الطـرـيقـةـ الليـ بـشـتـغـلـ بـيـهـاـ، يـعـنـىـ مـثـلاـ فـيـ الـ **Development testing** (e.g., Component, integration and system testing) الـ **goal** الرـئـيـسـيـ هوـ إـنـىـ أـعـملـ أـكـبـرـ عـدـدـ مـمـكـنـ مـنـ الـ **failures** عـشـانـ أـقـدـرـ أـتـعـرـفـ عـلـىـ الـ **defects** فـيـ الـ **software** وأـحلـهـاـ، أـماـ فـيـ الـ **Acceptance Testing** فالـ **goal** الرـئـيـسـيـ هوـ إـنـىـ أـتـأـكـدـ إـنـ الـ **system** بـيـشـتـغـلـ زـىـ المـتـوـقـعـ لـلـتـأـكـدـ إـنـهـ **يـحـقـقـ الـ requirements**، فـيـ بـعـضـ الـ **cases** بـيـقـىـ الـ **goal** الرـئـيـسـيـ مـنـ الـ **testing** هوـ التـأـكـدـ مـنـ الـ **quality** بـتـاعـةـ الـ **software** (مشـ بنـيـةـ إـصلاحـ الـ **defects** أوـ إـكتـشـافـهـاـ)، وـهـدـهـ عـشـانـ أـدـىـ مـعـلـومـاتـ لـلـ **stakeholders** عنـ الـ **risk** إـنـىـ أـطـلـعـ الـ **system** فـيـ الـ **time** المـحـدـدـ، أـماـ

ال defects عادة بيكون الغرض منه انى أتأكد إن مافيش Maintenance testing جديدة ظهرت وانا بعمل Operational testing changes لل development (بنعمله لما السيستم يشتغل live عند العميل) فالهدف الأساسي منه هو تقييم characteristics معينة في السيستم زى availability وال reliability مثلًا.

**ملحوظة:** التيس من ضمن فوایده إننا بنلاقي ال bugs قبل مايلاقيها العميل، لأنها لو وقعت ف إيد العميل بشكل متكرر وكتير جدا بتخل شكلنا وحش وممكن تأثر على تسويق البرنامج وممكن توصل لنعويضات لو البجات دى خطيرة.

### Debugging and testing

من الأخطاء الشائعة عند معظم الناس إنهم بيعتبروا ال testing هو ال debugging، فى الحقيقة الاتنين مختلفين عن بعض، ال failures أقدر من خلله أطلع ال Dynamic Testing اللي كانت defects سبب فيها، والتيس الللى بيتم بعد كده عن طريق التيسير بيتعمل عشان يتأكد إن الحل فعلا اللي اتعمل عالج ال failure .

أما ال development activity هو debugging بيلاقى أسباب ال failure ويحللها ويعالجها.  
يعنى م الآخر نقدر نقول إن ال testers بيعملوا test وال developers بيعملوا debug، وهتووضح أكثر عملية ال test activities و ال testing فى النقطة 1.4.

### Seven Testing Principles :1.3

فيه 7 قواعد في التيسينج تم إقتراحهم على مدار ال 40 سنة اللي فاتت ويتعرض القواعد العامة المعروفة في التيسينج

#### :Testing Principles

##### **Testing shows presence of defects -1**

ال tiesinj يقدر يثبت إن فيه defects موجودة، لكن مايقدرش يثبت إن مافيش defects، التيسينج بيقلل إحتمال ال defects الغير مكتشفة الباقيه فى السوفتوير، لكن حتى لو مافيش defects تم اكتشافها فده مش دليل كافى إن مافيش defects خالص.

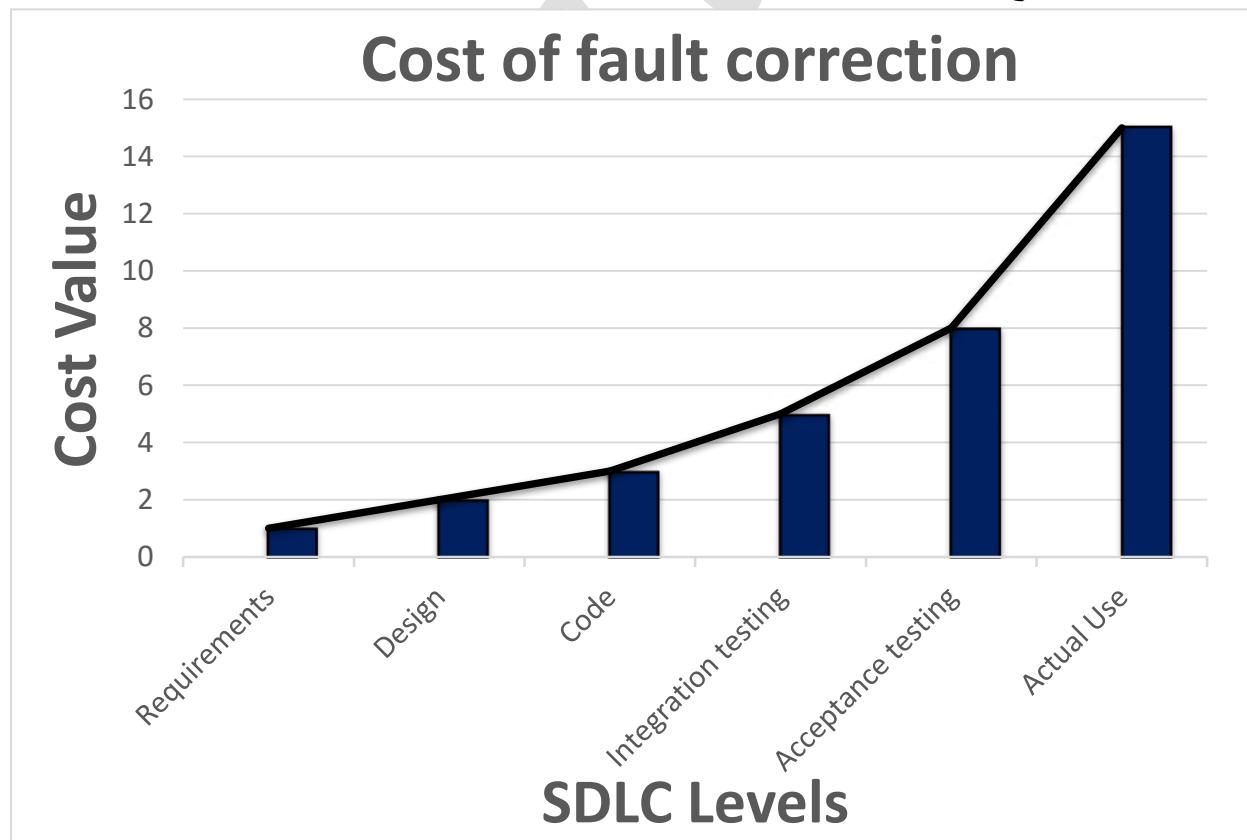
### :Exhaustive testing is impossible -2

بيقولك إن عمل تesting شامل لكل حاجة (ال inputs وال Preconditions ) مش حل عملى إلا فى حالات بسيطة، فبدل ال exhaustive testing ممكن نستخدم ال risk analysis وال priorities لتركيز ال testing efforts فى وجهتها الصح.

مثال: لو قلنا ان عندي text box بيس-تقبل الأرقام من 1 ل 100 بس، فلو بنعمل test case لكل value فنهحتاج 102 test case ( 0 - 1 .... 100 - 101 )، ولو فرضنا ان ال test case الواحدة بتاخذ دقيقة فى تفريزها، فده معناه اننا هننفذ الكلام ده فى 102 دقيقة ( ساعة و 42 دقيقة )، لكن باستخدام 4 test cases ( هنجيلها كمان شوية ) هنختصر الكلام ده كله ف 4 test design techniques بس، وهيتنفذوا فى 4 دقائق بس.

### :Early Testing -3

عشان تلاقي ال defects بدرى لازم تبدأ ال testing activities على قد ماتقدر فى ال Software or System Development Life Cycle ( SDLC )، وتكون مرکزة على أهداف إحنا عايزينها من عملية التesting، وكل ماتأخرنا فى اكتشاف ال bug كل ماتصالحها كلف أكثر، الشكل الجاي هيووضح الدنيا أكثر:



طب ليه التكالفة بتزيد كل مابنتقدم في مراحل الـ **SDLC**؟

لأن وقت تصالحها بيزيد وفي عالم السوق تvier بالذات time is money، فلو طلعننا bug أثناء مرحلة requirements آخرها معايا إنى أسأل على الحل بتاعها وأصلاحها ع الورق، إنما لو bug طلعت بعد تسليم البرنامج للعميل هيعمل report للشركة والشركة هتدور على أسباب الـ bug دى ولو مرتبطة بالـ requirements ممكن أهدم جزء كبير من الـ design والكود المعمولين ع الـ requirements دى عشان أحل الـ bug وطبعاً هنحتاج وقت تانى عشان نعمل عليها الـ re-test والـ regression ... فكل ده هيأخذ وقت وممكن يتقطع فيه شغل العميل وشغل الشركة وممكن العميل يطالب بتعويض عن تعطيل الوقت كمان.

## :Defect Clustering -4



يقولك إن الـ **effort** اللي بنعمله ف التيسينج لازم يـ **defects** إلى حد ما على الـ **modules** اللي فيها **focus** كتير أو متوقعين إن فيها **defects** كتير، عادة بيقى فيه جزء صغير من الـ **modules** فيها أغلب الـ **defects** اللي بنكتشفها أثناء مابنعمل **pre-release testing** أو ممكن تبقى مسئولة عن أغلب الـ **operational failures**، فمثلاً ممكن 20% من الـ **modules** أو الأجزاء اللي معهولة ف السـystem فيها حوالي 80% من الـ **defects** بعمل **prioritization** للتيست، وساعة مايشتغل التيسـتر يطلع بـ جـ ف الثالثة بيكون فـرـحـة سـلـيمـان عـيدـاـ

## **:Pesticide paradox -5**



لو نفس الـ **tests** بتتكرر مرات كتيرة ورا بعض فعادة  
الـ **defects** اللي عاملينها مش هتلافق أى **test cases**  
جديدة (لاحظ إن أكيد بيقى فيه **defects** مش مكتشفة)،  
وعشان نتغلب على الـ **pesticide paradox** ده لازم  
نراجع كل فترة على الـ **test cases** اللي عاملينها قبل  
ده ونكتب جديدة ومختلفة بـ **tests techniques** جديدة

عنوان نتیست على أجزاء ونواحي مختلفة من البرنامج عشان نلاقي defects جديدة، فمثلا لو طلعنا defects من برنامج معين وظهرت defects معانا حليناها في version 2 وعملنا نفس ال changes في version 1

على الـ **version** الجديدة دى، فممكن جداً ما تطلعش أى bugs جديدة لأننا حلينا الـ bugs اللي طلعت قبل كده، وبالتالي لازم أفكّر في طرق جديدة لـ **test cases** جديدة عشان أقدر أطلع bugs جديدة.

### **:Testing is context dependent -6**

التيست اللي إحنا بنعمله بـ **يختلف حسب السیستم** والـ **context** اللي بنعمل عليه التيسينج بتاعنا، فمثلاً عشان نتيست على **safety-critical software** هنحتاج طريقة نتيسته بيهَا مختلفة تماماً عن **e-commerce site**.

### **:Absence-of-errors fallacy -7**

مهما عملنا نتيست ع السیستم بتاعنا أو طلعنا **defects** حلوة جداً فكل ده **مالوش قيمة في هاتين:**

**1- لو السیستم نفسه مابيعملش الحاجة اللي طالبها العميل** أو متوقعها العميل، زي مثلاً لو العميل طالب عملية جمع فنعمله إحنا عملية ضرب مثلاً.

**2- لو السیستم اللي اتعمل مش هيستخدم**، زي مثلاً لو عامل برنامج حلو جداً لكن بيشتغل على **windows xp** بس، وبالتالي ماحدش هيشتغل عليه وبكده مايكونش ليه أي لازمة.

## **Fundamental Test Process :1.4**

### **Background**

أكثر جزء متضاف في عملية التيسينج هو الـ **test execution**، لكن عشان يبقى الـ **execution** ده فعال ومؤثر لازم الـ **test plans** تشمل الوقت المطلوب على التخطيط للتنيست وعمل ديزاين للـ **cases** والإعداد لتنفيذ التنيست وتقييم النتائج اللي طالعة.

### **Test process main activities**

**Test planning and control -1**

**Test analysis and design -2**

**Test implementation and execution -3**

**Evaluating exit criteria and reporting -4**

**Test closure activities -5**

**ملحوظة:** خد بالك من عدد الخطوات دى وترتيبها، لأنه بيسأل عليها في امتحان الـ **ISTQB**.

بالرغم من التسلسل المنطقي للactivities دى إلا إنها ممكن تتدخل overlap مع بعض أو تحصل مع بعض في نفس الوقت بالتوافق، وطبعاً لازم نربط الـ main activities دى بالـ context بناءً على السياق والـ project اللي بنعمله، والخطوات دى بتتنفذ حسب طبيعة المشروع ونظام الشركة.

### Test Planning and Control :1.4.1

الـ test planning هو activity اللي عايزينها من التيسينج اللي هنعمله ونحط الـ test activities اللي المفروض تتم عشان نوصل للـ objective أو الـ mission اللي حدناه ف الأول.

أما الـ test control بقى فهي عملية مستمرة بـ **بنقارن فيها الـ actual progress** اللي تم بالـ **status reports** موجود في الـ plan ونعمل بالـ plan الحالياً، ده طبعاً بيشمل الاختلافات اللي في الواقع عن الموجود في الـ plan، كمان الـ test control بتشمل الخطوات اللي المفروض تتعملاً عشان نضبط الاختلافات دى عشان نوصل للهدف اللي حدناه قبل كده في الـ plan، وعشان نتحكم في عملية التيسينج زى ما حنا عايزين لازم نراقب الـ activities اللي بتتم خلال الـ project، لأن الـ Test Planning بتأخذ في الاعتبار الـ monitoring and control activities اللي بييجي من الـ feedback هنتكلم أكثر عن الـ control activities بتفاصيل أكثر في شابتر 5 إن شاء الله.

### Test Analysis and Design :1.4.2

دى بقى الـ activity اللي فيها بتحول الأهداف العامة للتيسينج اللي حدناها في الـ test planning لـ **test cases** وـ **test conditions** ملموسة.

ـ items to be :test conditions هي أي item أقدر أعمله تبنته test case أو أكثر (test

### الـ major tasks ليها الـ test analysis and design activity دى:

- مراجعة الـ risk (زى الـ requirements) integrity - مستوى السلامة test basis (الـ test basis للـ testability القابلية للـ التيسىت).
- تقارير الـ architecture – risk analysis level (الـ risk analysis - الديزاين - مواصفات الـ interface).
- تقييم الـ test objects (الـ test basis للـ testability) والـ testability (الـ test basis للـ testability).
- التعرف على الـ test conditions وترتيب الأولوية prioritizing ليها بناء على تحليـل كل الـ test items والـ behavior والـ specifications والـ structure للـ test items للـ test cases للـ test conditions للـ test data للـ test cases.
- عمل الـ design والـ test cases (مش high level test cases) prioritizing للـ test cases فعلاً لكنها high level يعني حاجة عامة كده من غير ماندخل فى تفاصيل).
- تحديد الـ test data الـ ضرورة اللي بنحتاجها وأحنا بنشتغل على الـ test conditions والـ test cases.
- عمل الـ tools infrastructure وتحديد أى setup design للـ test environment للـ test cases.
- عمل تتبع ثنائى bi-directional traceability بين الـ test basis والـ test cases (information documents) (بنحط الـ test cases والـ test basis مع بعض).

### Test Implementation and execution :1.4.3

دى بقى المرحلـة أو الـ activity اللي فيها بنحوـل الـ test procedures or scripts وبنرتـبها وبندخل معانا أى معلومات تانية محتاجـينها وأحنا بنعمل تنفيـذ execution للـ التيسـت بتاعـنا، فبنـسطـب الـ environment وبنـrun التـيسـت بتـاعـنا.

### الـ major tasks ليـه الـ test implementation and execution دـى:

- عمل الـ test cases بتفاصيلها ونعملـها prioritizing بحيث نشتغل عـ المهم فيـهم الأول (طبعـاً) (test data) وبـنـحدـدـ فيها الـ test data.
- بنـ develop الـ test procedures الـ test data اللي هـتنـتفـدـ، ونحضرـ الـ test data وأحيـنا نعملـ الـ test harness (الـ documentation الحاجـات اللي بـجهـزـها قبلـ ما أعملـ تـيسـت زـى الـ documentation مثلـاً).
- ونـكتـبـ الـ test automation لو هـنـعملـ automated test scripts.
- هـنـعملـ الـ test suites اللي فيها بنـكتـبـ الـ test cases بالـ procedures بتـاعـتها عـشـان نـعملـ الـ test execution مؤـثرـ ولـيه قيمةـ.

- نتأكد إن الـ **test environment** اتسطبت وجاهزة بالشكل المطلوب عشان نـ **run** الـ **test cases** بتابعنا.
- نتأكد من المراقبة الثنائية **bi-directional traceability** بين الـ **test basis** والـ **test execution tools** ونعملها **update** لو هحتاج لكده.
- ننفذ الـ **test procedures** **manually** سواء أو باستخدام **test execution tools** حسب الـ **sequence** اللي مخططين ليه.
- تسجيل الـ **outcome** اللي طالع من الـ **test execution** وكمان الـ **identities** والـ **versions** للسوافتويير اللي بنعمله تيست والـ **testware** والـ **test tools** (أى حاجة تخص عملية التيسـ **test cases** والـ **tools documentation** والـ **test cases documentation**) عشان أعملها **archiving** عشان أستعملها بعد كده.
- هنقارن بين الـ **actual result** اللي طلعت معانا بالفعل فى الـ **expected result** والـ **test case** اللي كنا متوقعينها واحدنا بنعمل الـ **result**
- هنعمل **Incidents report** بالـ **expected result** (الاختلافات بين الـ **actual result** والـ **expected results**) وتحليلها عشان نعرف سببها (زى مثلا **defect** ف الكود أو الـ **test data** اللي استخدمناها أو الـ **test document** ماكانتش مطبوبة أو أى خطأ فى طريقة تنفيذ التيسـ **test activities** اللي دى بعد مايحلها الديفلوبـ **incidents** الخاصة بالـ **test activities** بتابعها عشان نتأكد انها اتحلت ( النوع ده اسمه **confirmation testing** ، وكمان **case** execution)، بنعمل **test cases** المرتبطة بيها حتى لو كانت **passed** قبل كده عشان نتأكد إن الـ **fix** اللي تم للـ **incident** مااثرش ع الـ **modules** المرتبطة بيها (بنسمى النوع ده **(Regression testing)**).

#### Evaluating Exit Criteria and Reporting : 1.4.4

ف المرحلة دى بقى بنقيم الـ **test execution** اللي عملناه ونقارنه بالـ **objectives** اللي عملناها ف الأول أثناء الـ **Planning**، وده لازم يتعمل لكل **test level** (هنشوفها ف النقطة 2.2).

طب إيه هـ **exit criteria** أصلـ؟

**exit criteria** هـ **exit criteria** هي النقطـ أو المرحلة اللي بتحدد إمتـ أوقف التـ **testing levels**، وبتتطـ على كل **exit criteria**، فمـ **exit criteria** بتاعـ يكون:

- الـ coverage باستخدم طرق قياس زى: Decision coverage o بمعنى إنى مثلاً ممكن أوقف تىست لو unit testing فى الـ decisions اللـى ف الكود (هنعرفها أكثر فى 100% coverage .(Part 2)
- User requirements o بمعنى إنى ممكن أقول هوقف تىست لو غطيت 100% من الـ requirements اللـى طلبها اليوزر user .
- Most frequently used transactions o غطيت أكثر العمليات المستخدمة بشكل متكرر ف السيسـتم.
- faults المكتشفـة (يعنى ممكن أوقف تىست لما أوصل لمشكلـة عكس المتوقع).
- التكلفة أو الوقت (ممكن أوقف تىست لو لقيت إن الاستمرار ف التىست هيكـلـفـ عـلـيـاـ أوـ لوـ هـسـلـمـ فـىـ معـادـ مـعـيـنـ فـلـازـمـ أـوـقـفـ التـىـسـتـ عـنـ الدـادـهـ).

#### تقييم الـ exit criteria بـيـشـمـلـ tasks دـى:

- بيـشـيكـ علىـ الـ test logs (الـى سـجـلـناـهـاـ) وـيـقارـنـ بـيـنـهـاـ وـبـيـنـ الـ exit criteria (الـأسـاسـ اللـىـ هـوـقـفـ عـنـهـ التـىـسـتـ) اللـىـ كـانـ حـاطـئـهـ فـىـ الـ test planning .
- بنـقـيمـ الـوضعـ وـنـشـوفـ لـوـ مـحـاجـينـ عـمـلـيـاتـ test criteria أـكـثـرـ أـوـ نـغـيرـ الـ .exit criteria
- بنـكتـبـ stakeholders test summary report لـاتـخـادـ القرـارـ المناسبـ زـىـ ماـقـلـنـاـ قـبـلـ كـدـهـ .

### Test Closure Activities :1.4.5

فـ المرـحلةـ دـىـ بـنـجـمـعـ الدـاتـاـ مـنـ الـ test activities المـكـتمـلةـ وـنـلـخـصـهـاـ عـشـانـ نـخـزـنـهـاـ فـيـ شـكـلـ خـبـراتـ وـحـقـائـيقـ وـأـرـقـامـ (زـىـ مـثـلاـ تـقـرـيرـ عنـ الـ bugs اللـىـ لـاقـيـنـاهـاـ)، الـ test closure وـtestware بـتـحـصـلـ فـىـ الـ project milestones activities زـىـ مـثـلاـ لـماـ نـعـملـ لـلـ software release أوـ لـماـ الـ test project يـكـمـلـ أـوـ يـتـكـسـلـ - كـدـهـ يـبـقـىـ اـتـحـقـقـتـ milestone - أـوـ لـماـ يـكـمـلـ maintenance release للـسـيـسـتمـ .

### الـ tasks بيشمل test closure activities دى:

- بي Shawf إيه اللي وصلنا له من الـ planned deliverables اللي كنا محددينها قبل كده وإيه اللي لسه ماوصلنا لهوش.
- بيغفل الـ remain open changing records أو بيرفع الـ incident reports لأى Documenting the acceptance of the system للسيستم acceptance.
- بيكتب الـ test infrastructure والـ test environment والـ testware لإعادة الاستخدام بعد كده.
- بيسلم الـ maintenance organization testware Handing over للـ testware.
- بيحلل الدروس المستفادة / اللي اتعلمناها من الـ project ده عشان يحدد التغييرات المطلوبة فى الـ future releases and projects.
- بنستخدم المعلومات اللي عرفناها دى لتحسين الـ test maturity.

## The psychology of Testing :1.5

### Background

بيقولك ان الـ mindset المستخدمة أثناء الـ testing والـ reviewing مختلفة عن الـ mindset المستخدمة أثناء الـ developing software، فمع طريقة التفكير الصح الـ developers يقدروا يعملوا تيست لل코드 بتاعهم، لكن فصل المسؤولية دى للتنيستر لازم يحصل عشان ده بيـ focus effort الـ focus اللي بيتم فى التيسينج وكمان ليه مزايا إضافية، زي مثلاً النظرة المستقلة للتنيستر المحترف والـ اللي معاه معاـ independent testing، professional testing resources فى التيسينج level.

بيقولك ان الـ independence لما بيكون فى أعلى درجاته – بدون تحيز مؤلف الـ syllabus – عادة بيخلـى التنيستر effective أكثر انه يطلع الـ failures والـ defects، وبالرغم من كده فهو مش بديل للـ familiarity، فالديفلوبـرـز ممكن برضه يلاقوا defects كتير فى الكود بتاعهم.

فيه مستويات كثير للIndependence، المستويات دى هي:

- إن الشخص اللي عامل **السوفتوير** (الديفلوبير) هو نفسه اللي يعمل الديزاين للتيست (ده أقل مستوى في independence).
- حد تانى من نفس **الجروب** (من تيم ال development مثلاً) هو اللي يعمل الديزاين للتيست.
- حد من تيم تانى **جوه الشركة** هو اللي يعمل الديزاين للتيست (زى independent test usability or performance test team أو test specialist (team specialists).
- حد من شركة تانية **خالص** غير شركتنا هو اللي يعمل الديزاين للتيست (حد outsourcing or independence level)، وده أعلى level certification.

كمان من المهم جداً انك تحط objectives وأهداف لعملية التيسينج اللي بتعملها، ده لأن objectives هي اللي بتحرك الناس وال projects، ده لأن الناس غالباً بترتبط بال plans بتاعتتها بال stakeholders اللي بتحطها الإداره وال objectives، فعشان تحدد الاتجاه اللي هتمشى فيه في التيسينج وتحدد نوع ال defects اللي بتدور عليها وبتعمل تيست على أساسها أو تتأكد إن سوفتوير بيحقد objectives اللي اتعمل عشانها.

عملية اكتشاف failures أثناء التيسينج ممكن يتصلها بعض النقد في ال product وال author، كمان، ونتيجة لكده عادة بيعتبروا التيسينج عملية هدامه بالرغم من أنها عملية بناءة جداً في إدارة product risks، البحث عن failures في السيستم بيطلب حب الاستطلاع وتشاؤم محترف professional pessimism وعين فاحصة منتبهة لتفاصيل وتوافق جيد مع الزملاء في error guessing والخبرة اللي بناء عليها هتبقى الأساس لتوقع الإيرور development.

لو ال errors أو ال defects أو ال failures أو ال constructives، ممكن تتجنب ال bad feelings بين analysts, designers and developers testers، ده بالإضافة لل defects اللي لاقيناها أثناء ال reviewing كذلك أثناء التيسينج.

عشان كده لازم ال tester وال test leader يبقى عندهم مهارات تواصل كويسيّة عشان يصلوا المعلومات الخاصة بال defects وكذلك ال progress وال risks بشكل واقعي، أما بالنسبة لل document author بتابع سوفتوير أو ال document ممكن معلومات ال defect تساعده في تحسين مهاراته، كمان ال defects اللي بنلاقيهما وبنتصالح أثناء ال testing بتتوفر الوقت والفلوس بعد كده وبقلل risks.

ممكن تحصل مشاكل في ال communications، بالذات لو ال testers تم النظر لهم على انهم هما اللي دايما بيجبوا الأخبار الوحشة، ومع ذلك فيه طرق كتير لتحسين ال communication والعلاقات بين ال testers وباقى التيم.

#### من ضمن الطرق دي:

- خليك متعاون بدل مانقعد تحارب في خلق الله – فكر كل واحد بالهدف المشترك لجودة سيسنتم أفضل.
- وانت بتتواصل مع الناس اعرض الحاجة اللي لاقيتها بشكل موضوعي وركز ع الحقيقة من غير ماتجرح ف الشخص اللي عاملها، يعني مثلا اكتب ال objective وال incident اللي بتواجهك في ال report وراجع ع اللي انت لاقيته تانى.
- حاول تحس ياحساس الشخص اللي جاي تقوله الكلام ده وليه هو بيتصرف كده (الديفلوبر بيحس ان شقى عمره ضاع لما بيطلع له bug).
- اتأكد إن الشخص اللي بتقوله ع المشكلة بتاعتاك **فهم تماما** اللي انت عايز تقوله والعكس صحيح طبعا.

### Code of Ethics :1.6

بيقولك إن الشغل في ال software testing بيتيح للأشخاص انهم يعرفوا معلومات مهمة ومش بتطلع لكل الناس، عشان كده ال code of ethics ضروري وعشان فيه كمان أسباب تانية تحمي علينا إن المعلومات ماتستخدمش استخدام غير صحيح، وبعد التعرف على ال ACM code of ethics الخاص ب IEEE، ال ISTQB حطت مجموعة من ال code of ethics.

#### :ISTQB Code of Ethics

- **PUBLIC**: ال certified software testers لازم يتصرفوا في شغلهم في إطار الاهتمام العام (سواء للمشروع أو للشركة أو حتى للمجتمع).
- **CLIENT AND EMPLOYER**: ال certified software tester لازم يتصرف بطريقة بحيث انه يحقق أحسن طلبات ال client وال employer ومتكملا مع الاهتمام العام.
- **PRODUCT**: ال certified software tester لازم يتتأكد إن الشغل اللي بيعمله (على products وال systems اللي بيعمل عليهم التيسن) بتحقق أعلى معايير بروفشنال ممكنة.
- **JUDGMENT**: ال certified software tester لازم يتصرف بحيادية واستقلال لما ييجي يحكم على حاجة بشكل بروفشنال.

- certified software test managers and leaders :**MANEGEMENT** • لازم يلتزمو ويشجعوا النواحي الأخلاقية أثناء إدارتهم لعملية ال software testing .
- ال certified software tester :**PROFESSION** • لازم يراعي التزاهة وسمعة المؤسسة والمهنة مع الشأن العام.
- ال supportive fair certified software tester :**COLLEAGUES** • لزمائيله ويتعاون مع ال software developers .
- ال certified software tester :**SELF** • لازم يشارك فى التعليم طويل الأجل طبقاً لل practice lifelong learning (لازم يشغل على نفسه ويطورها).

## Chapter 2

### Testing Throughout the Software Life Cycle

#### :Software Development Models :2.1

##### **Background**

التيستينج مش مرحلة منعزلة عن الـ (SDLC)، بالعكس **Software Development Life Cycle (SDLC)** مرتبطة بالـ **test activities** مرتبطة بالـ **software development activities**، وكل **development life cycle model** يحتاج طريقة في التيسينج تتناسب به في التعامل معاه، في **syllabus** بتاعنا انكلم الكتاب عن **3 SDLCs** مختلفين، كل واحد فيهم شغال بطريقة مختلفة عن الباقيين، وكل واحد منهم يحتاج طريقة تعامل مختلفة في التيسينج... تعالوا نشوفهم يلا.

#### :Software Development Life Cycle (SDLC) models

##### **V-model (Sequential Development Model) :2.1.1**

برغم الاختلافات في الـ **V-models** الموجودة، إلا إن النموذج المتعارف عليه في الـ **V-model** بيستخدم **4 development levels** طبقاً لـ **4 test levels**

##### :V-model levels

- Component (unit) testing
- Integration testing
- System testing
- Acceptance testing

ممكن الـ **V-model** يكون فيه **levels** أكثر أو أقل أو حتى مختلفين عن دول سواء في الـ **development** أو الـ **testing** حسب طبيعة الـ **project** والـ **software product** اللي بيتعمل، يعني مثلاً ممكن يبقى فيه **component integration testing** بعد الـ **system testing** أو يبقى فيه **system integration testing** بعد الـ **component integration testing**.

الـ **use cases** أو **business scenarios** (زى مثلاً الـ **software work products**) والـ **design documents** والـ **requirements specifications** والـ **code** اللي بيطلعوا أثناء الـ **development** بيكونوا عادة الـ **test basis** أو أكثر، **basis of testing** (test basis).

كمان الـ **Capability Maturity** include work products للـ references Model Integration (CMMI) or 'Software life cycle processes' (IEEE/IEC 12207)، كمان الـ **early test design** والـ **validation** والـ **verification** ممکن يحصلوا أثناء الـ **software work products** فى الـ **development**.

لو الدنيا مش واضحة معاك ماقلقش... هننشرح النقطة دى بتفاصيل أكثر فى part 2 ان شاء الله... احنا هنا بس بنركز ع الموجود ف الـ **.syllabus**.

### **Iterative-incremental Development Models :2.1.2**

الـ **Iterative-incremental development** هي عملية عمل الـ **requirements** والـ **design** والـ **building** والتيسينج للسيستم فى صورة **development cycles** صغيرة ومتسلسلة ورا بعضها، بمعنى إنى بقسم البرنامج على أجزاء وكل جزء بعمله الشغل الخاص بي.

#### **:Iterative-incremental models**

- Prototyping
- Rapid Application Development (RAD)
- Rational Unified Process (RUP)
- Agile development models

السيستم اللي بيعمل بالـ **models** دى ممکن يتعمل عليه تيست بأكتر من **test level** أثناء كل **iteration** يتعمل، الـ **increment** – بينضاف للحاجات اللي اتعملت قبل كده – بيشكل سيستم بيكبر حته حنة، ولازم كل ده يتعمله تيست طبعا، فالـ **regression testing** ف الحالة دى بتزيد قيمته على كل الـ **iterations** بعد أول واحد، كمان الـ **validation** والـ **verification** ممکن يحصلوا على كل **increment** بتعمل.

برضه النقطة دى هنجيلها فى part 2 ان شاء الله.

### Testing within a Life Cycle Model :2.1.3

بيقولك إن في أي life cycle model good testing فيه خصائص كثير للزى:

- لكل **development activity** فيه **testing activity** مقابل ليه.
- كل **test objectives** ليه **test level** مخصصة للـ **level**.
- الـ **test analysis and design** للـ **level** الذى احنا واقفين عنده لازم يبدأ بالتوافق مع **development activity**.
- الـ **testers** لازم يشتراكوا فى مراجعة الـ **documents** بمجرد ماتجهز الـ **drafts** المتاحة فى **development life cycle**.

ممكن الـ **test levels** تتجمع أو يتبعad تنظيمها على حسب طبيعة الـ **project** أو الـ **system**, يعني مثلا لو عملت **Commercial Off-The-Shelf integration architecture** فى **integration testing (COTS)** (زى السيستم بتاعى), ممكن العميل يطلب **functional and/or non-functional Acceptance testing** (system deployment .(functional, and user and/or operational testing

### :Test Levels :2.2

#### Background

لكل **test level** لازم نبقى عارفين الحاجات دى: الأهداف العامة للتيسىت اللي احنا بنعمله – الـ **work product(s)** اللي ه تكون مرجعنا واحدنا لنعمل الـ **test cases** (زى مثلا الـ **test basis**) – الـ **test object** (احنا هنتيسىت إيه بالضبط) – الـ **failures** والـ **defects** اللي دور عليها – الـ **test harness** والـ **tool support** – والطريقة اللي هستخدمها والمسؤوليات بتاعتى، ولازم واحدنا بنعمل **test planning** .**system's configuration data** نحط ف الاعتبار الـ **test planning**

و دلوقتى هنتكلم على الـ **test levels** بالتفصيل.

### Component Testing :2.2.1

#### الlevel of test basis

- Component requirements
- Detailed design
- Code

#### أما ال typical test objects

- Components
- Programs
- Data conversion / migration programs
- Database modules

ال (unit, module or program testing) يدور على ال functioning defects ويتتأكد من أنها شغالة تمام في ال software modules على ال objects وال classes وال programs وال الحاجات التي يمكن عملها تيست بشكل منفصل، فممكن عمل التيست هنا بشكل منعزل عن باقى السيسن، وده بيعتمد على محتوى drivers وال stubs وال development life cycle وال simulators (هشرحها أكثر في part 2).

ال non-functional component testing ممكن يشتمل على تesting لاحتاجات واحتاجات functional معينة زي ال (white box) أو اختبار المثانة/الصلابة robustness testing، بالإضافة إلى (لو شغالين work test cases)، ال structural testing e.g., decision coverage زي مثلا ال specification products (test basis) وال component بقاعة ال software design أو ال data model.

#### ملاحظات:

- ال momory leaks: معناها إن السيسن بيكون حاجز أماكن في الميموري ليه لوحده وماحدش يأخذ الأماكن دي غيره.
- ال Robustness testing: هي طريقة للتtesting لاكتشاف نقط الضعف في component .stressful environment معينة سواء عن طريق إدخال inputs غير متوقعة أو الشغل في

الـ **component testing** فيه بنـ **access** الكود اللي بنعمل عليه التيست وبيبقى فيه من **support** عادة **debugging tool** أو **unit test framework** زى **development environment** بيشترك فيه المبرمج اللي كتب الكود، والـ **defects** بتتحل بمجرد ما بيتم اكتشافها من غير مايتعلما report بشكل رسمي.

فيه طريقة تانية لـ **component testing** وهى عبارة عن اننا نجهز الـ **test cases** ونعملها **test-driven coding** قبل الـ **automate**، الطريقة دى إسمها **test-first approach** أو **test-driven development**، فـ **الطريقة دى** بدئ للديفلوبر الـ **test cases** الأول بحيث يتتجنب الـ **defects** قبل مايعلم الكود)، الطريقة دى تكرارية جداً ومعتمدة على الـ **cycles of test cases**، وبعد كده عمل حتـ صغيرة من الكود ونعملها **integration** مع الباقيين، ونـ **execute component tests** ونصح أى **issues** ونكرر الكلام ده تانى لغاية ما يبقى **pass**.

### Integration Testing :2.2.2

#### الـ level فـ test basis ده :

- Software and system design
- Architecture
- (business Workflows
- Use cases

#### والـ typical test objects هي:

- Subsystems (يعنى هعمل تيست على جزء صغير من السيسن)
- Database implementation
- Infrastructure (التعامل مع الـ hardware والـ software)
- Interfaces
- System configuration and configuration data (بيـ support الـ hardware)
- والـ software ولا لأ).

ال integration testing ي العمل بين ال components interfaces لـ testing المختلفة وكذاك ال interactions مع أجزاء مختلفة من السيستم زى:

- ال Operating System (OS)
- ال hardware والهاردوير file system
- ال systems interfaces بين ال

كمان من الممكن يبقى فيه أكثر من level لـ testing وممكن يحصل على test كمان من الممكن يبقى فيه أكثر من level لـ testing وممكن يحصل على objects بأحجام مختلفة زى مثلا:

-1 ال Component integration testing بيتبيست ال interactions بين ال software ويتعمل بعد ال component testing components.

-2 ال System integration testing بيتبيست ال interactions بين ال systems مختلفة أو بين ال hardware وال software وممكن يتعمل بعد ال System testing ، وف الحالة دى ال developing organization ممكن تتحكم فى جانب واحد بس من ال interface وده ممكن يعتبر risk لأن ال business processes بتتنفذ ب workflows ممكن يشمل سلسلة من ال systems cross-platform issues ، وبالتالي ال systems ممكن تبقى كتيرة.

وكل ماكبـر ال scope بتاعـنـا كل مايصعب حصرـال defects فى أو سـيـسـتـمـ معـيـنـ ، وـده مـمـكـنـ يـزـوـدـ الـ riskـ والـوقـتـ الـلىـ هـيـتـاخـدـ عـشـانـ نـكـشـفـهاـ وـنـحـلـهاـ.

ال integration strategies مـمـكـنـ تـبـقـىـ علىـ basedـ علىـ:

- ال (bottom-up و top-down) system architecture
- أو ال functional tasks
- أو ال transaction processing sequences
- أو أى other aspect of the system or components

وعشـانـ نـسـهـلـ عـلـىـ نـفـسـنـاـ الـ integrationـ fault isolationـ وـنـحدـدـ الـ defectsـ بـدـرـىـ ، الـ integrationـ لـ اـلـمـ لـ اـلـ

تلـقـائـيـاـ يـبـقـىـ "big bang" بـدـلـ الـ "incremental" (هـنـشـرـحـ الـ كـلـامـ دـهـ أـكـثـرـ فـىـ 2ـ partـ 2ـ).

ممـكـنـ بـرـضـهـ نـعـملـ تـيـسـتـينـجـ عـلـىـ non-functional characteristicsـ معـيـنـةـ (زـىـ الـ functionalـ performanceـ) أـثـنـاءـ الـ integrationـ testingـ جـنـبـ مـعـ الـ functionalـ testingـ .

في كل مرحلة في الـ integration testers يركزوا بس على نفسـه، يعني integration testing مثلـاً لو فيه module A و module B وبنعملـهم integration لازم نركـز علىـ integration communication بينـهم بـس مشـ functionality كلـ مودـيـولـ فيـهمـ لـوحـدهـ، لأنـ دـهـ اـتـعـمـلـ قـبـلـ كـدـهـ فـ الـ component testing، مـمـكـنـ أـسـتـخـدـمـ الـ functional and structural approaches.

وعـشـانـ الشـغلـ يـبـقـىـ مـثـالـىـ أـوـيـ لـازـمـ الـ testersـ يـفـهـمـواـ الـ architectureـ وـ تـأـثـيرـ الـ integration planningـ فـلـوـ عـمـلـيـةـ التـيـسـنـجـ تمـ تـخـطـيـطـهاـ قـبـلـ مـاتـعـمـلـ الـ componentsـ أوـ الـ most efficientـ فالـ componentsـ systemsـ دـىـ مـمـكـنـ تـعـمـلـ بـالـ تـرـتـيـبـ الـ المـطـلـوبـ لـلـ .testingـ

### System Testing :2.2.3

الـ levelـ فـ الـ test basisـ دـهـ:

- System and software requirement specification
- Use cases
- Functional specification
- Risk analysis reports

والـ typical test objectsـ هـىـ:

- System, user and operation manuals
- System configuration and configuration data

الـ testing scopeـ بـيـهـتـمـ بـالـ behaviorـ للـ sysـtemـ أوـ الـ productـ كـلـهـ، والـ system testingـ لـازـمـ يـتـلـخـصـ بـشـكـلـ وـاضـحـ بـشـكـلـ عـامـ أوـ /ـ والـ Level Test Planـ لـلـ test levelـ دـهـ.

فيـ الـ final targetـ لـازـمـ الـ test environmentـ تـتوـافـقـ معـ الـ production environmentـ بـقـدـرـ الإـمـكـانـ عـشـانـ نـقـلـ الـ riskـ environmentـ specific failuresـ الليـ مـاتـمـشـ اـكتـشـافـهاـ فـ التـيـسـنـجـ.

### الـ system testing تكون tests based على ممكـن يبقى من ضمنه:

- requirements specifications و/أو الـ risks
- Business processes
- system models أو high level text descriptions أو Use cases
- behavior
- Operating System interactions مع الـ
- system resources

الـ system testing لازم يدقق فى الـ functional and non-functional requirements فى السيـستـم وكذلك فى الـ testers، كمان محتاجين يتعاملوا مع الـ requirements اللي لسه ماكملتش أو لسه ماتكتبش، الـ system testing مع الـ functional requirements specification-based technique مناسب فى الـ decision techniques عشان نتنيـسـتـ الجـزـءـ اللي عـاـيـزـينـهـ منـ السـيـسـتـمـ، يعني مثلاـ الـ black box (business rules) table ممكن نـسـتـخـدمـهـ لوـ فـيـهـ مـجـمـوعـةـ منـ الـ effectsـ مـوـصـوـفـةـ فـ الـ white box (structure-based techniques) مـمـكـنـ نـسـتـخـدمـهـ لـ تـقـيـيـمـ دـقـةـ التـيـسـتـينـجـ معـ الـ menu structureـ زـىـ الـ web pageـ مراعـاةـ وـتقـدـيرـ الـ structural elementـ أوـ الـ navigationـ مـثـلاـ (هـنـتـكـلـمـ اـكـثـرـ فـ النـقـطـةـ دـىـ فـ شـاـبـتـرـ 4ـ).

بيكون فيهـ independent test teamـ هوـ الـ system testingـ

### **Acceptance Testing :2.2.4**

#### الـ test basisـ فـ الـ levelـ دـهـ:

- User requirements
- System requirements
- Use cases
- Business processes
- Risk analysis reports

### والـ typical test objects هي:

- Business processes on fully integrated system
- Operational and maintenance processes
- User procedures
- Forms
- Reports
- Configuration data

عادة الـ **Acceptance testing** يكون مسؤولية الـ **customers** أو الـ **users of a system**، وممكن تأثيرهم يدخلوا فـ **الحوار**.

الهدف فـ **acceptance testing** هو اكتساب الثقة في السيستم أو أجزاء منه أو **specific non-functional characteristics** في السيستم، هنا البحث عن الـ **defects** مثل هو الهدف الرئيسي في الـ **acceptance testing**، ممكن كمان الـ **acceptance testing** يقيم قابلية السيستم للـ **deployment** والاستخدام، وبالرغم من كده مش ضروري يكون الـ **acceptance testing** هو آخر level فـ **التيستينج**، يعني مثلا الـ **large-scale system integration test** ممكن ييجي بعد الـ **acceptance test** للسيستم.

### الـ Acceptance testing ممكن يحصل في أوقات كتير في الـ life cycle زى مثلا:

- الـ **COTS software product** ممكن يتعمله acceptance test لما يتعمله **install**
- أو لما يتعمله **.integrate** ممكن أعمل **Acceptance testing** على الـ **component** معين أثناء **usability**
- ممكن أعمل **Acceptance testing** على الـ **function enhancement** جديدة قبل **system testing**

## الـ Acceptance Testing ليه أكثر من زى Form

### User acceptance testing -1

هنا بنتأكيد إن السيستم مناسب لل business users عشان يستخدموه.

### Operational (acceptance) testing -2

هنا بن Shawf مدى قبول الـ system administrators للسيستم، وده بيشمل:

- Testing of backup/restore
- Disaster recovery
- User management
- Maintenance tasks
- Data load and migration tasks
- Periodic checks of security vulnerabilities (weakness points)

### Contract and regulation acceptance testing -3

الـ Contract acceptance testing بيعمل فى مقابل معايير acceptance اللي ف العقد المعمول لإنتاج custom-developed software، معايير القبول ممكن تعرف لما كل الأطراف يوافقوا ع العقود، الـ Regulation acceptance testing بيعمل فى مقابل أى لوايح لازم الالتزام بيها، زى اللواوح الحكومية والقانونية ولوائح الأمان regulations.

### Alpha and beta (or field) testing -4

بيقولك إن الـ COTS software – أو developers of the market software – عادة بيعوزوا ياخدوا feedback من العملاء الموجودين أو المهمين فى السوق بتاعهم قبل ما يعرضوا product للبيع بشكل تجاري، الـ Alpha testing بيعمل فى شركة البرمجة لكن اللي بيعمله مش developing team – أو Beta testing، إنما Field Testing – اللي بي عمله customers أو العملاء المهمين فى أماكنهم الخاصة عندهم فى الشركات.

ممكن الشركات تستخدم معايير تانية برضه زى الـ site factory acceptance testing وال factory acceptance testing لل systems.

**ملحوظة:** لازم يكون عندك الصبر على كل ده، لو ما عندكش الصبر على إنك تخبر السيستم، السيستم هو اللي هيختبر صبرك.

## :Test Types :2.3

### :Background

فيه مجموعة من الـ **test activities** هدفها هو التأكد إن السيستم (أو جزء منه) بيرتكز على سبب محدد أو **target** معين عشان نعمله تيست.

#### الـ **target** بيرتكز على **test objective** محدد، ممكن يكون واحد من دول:

- الـ **function** اللي بيأدتها السوفتوير.
- زى مثلا الـ **reliability** أو **non-functional quality characteristic**.
- الـ **usability** أو **architecture** أو **structure** للسوفتوير أو السيستم.
- زى مثلا التأكد إن الـ **defects** اتحلت (الـ **Change related (confirmation testing)**) أو التأكد إن الـ **regression testing**.

من الممكن إن يكون فيه **model** فى السوفتوير يتعمله **develop** و/أو يستخدم فى الـ **structural** (زى مثلا **menu structure model** أو **control flow model**) أو **non-model** (زى الـ **functional testing** أو **security model** أو **performance model** أو **usability model**) والـ **process flow model** (زى مثلا الـ **functional testing** أو **threat modeling** أو **state transition model** أو **plain language specification**).

### Testing of function (Functional Testing) :2.3.1

الـ **functions** اللي بيعملها السيستم أو **component** أو حتى **subsystem** معين ممكن تتوصف فى **requirements specification** أو **use cases** أو **work products** أو **functional specification**، أو ممكن تكون **undocumented functions**، الـ **functional specification** هى الشئ اللي بيعمله السيستم .  
the functions are "what" the system does

الـ **Functional tests** (الموصوفة فى **features**) بترتكز على الـ **functions** والـ **specific systems documents** أو مفهومة بالنسبة للـ **testers** وتوافقها مع **test levels** على كل الـ **components** (زى مثلا لما أعمل تيست على **component specification** ممكن تكون مرتكزة على **component specification**).

ال **specification-based techniques** ممكن نستخدمها عشان نستخلص ال **test cases** من ال **functionality** بتاعة السوفتوير أو السيسنتم (هنجيلها ف شابتر 4)، ال **functional testing** (ده اسمه **Black-box testing**) بياخد بعين الاعتبار ال **external behavior** للسوفتوير.

ال **Functional testing** (بيعتبروه في منهج ال ISTQB نوع من أنواع ال **Security testing**) بيشتغل على تحديد ال **threats** زى الفيروسات مثلًا من ال **malicious outsiders**.

فيه نوع تاني من ال **functional testing** اسمه **Interoperability testing**، وده بيقيس قدرة ال **software product** على التفاعل والتعامل مع **component** معين أو سيسنتم معين أو أكثر.

### Testing of Non-functional Software Characteristics (Non- functional Testing) : 2.3.2

ال **Non-functional testing** بيـ— include (لكن مش مقتصر على) ال **performance testing** ، **usability testing** ، **stress testing** ، **load testing** ، **testing** ، **portability testing** ، **reliability testing** ، **maintainability testing** ، فهو عبارة عن تيسينج لكيفية شغل السيسنتم .it's the testing of "how" the system works

ال **Non-functional testing** ممكن يتعمل على كل ال **test levels** ، وده مصطلح بيعبر عن عمليات التيسنن المطلوبة لقياس خصائص معينة ف السيسنتم والسوفتوير ممكن تتقاس كميتها أو قيمتها على نطاقات مختلفة، زى مثلا ال **response times** فى ال **performance testing** ، **Software Engineering - Software Product Quality** (ISO 9126) دى ممكن نلاقيه مشار إليها فى أى **quality model** زى اللي متعرف فى **black-box test design techniques** بياخد بعين الاعتبار ال **external behavior** للسوفتوير وفيأغلب الحالات بيسخدم عشان يحقق الكلام ده.

### Testing of Software Structure/Architecture (Structural :2.3.3 Testing)

ال Structural (white box) testing ممكن يتعمل فى كل ال test levels، وده بيقى أفضـل لو استعملناه بعد ال specification-based techniques، وده عشان يساعدنا فى قياس مدى دقة التesting أثناء تقييم ال coverage لنوع معين من ال structure.

ال coverage هى المدى اللي بنقىس بيه عملية تنفيذ ال structure عن طريق ال test suite، وبitem التعبير عنها بالنسبة المؤوية لل items اللي المفروض بنعملها coverage، ولو ال coverage دى ماوصلتش ل 100% معنى كده اننا محتاجين نعمل design لعمليات تبـتـ أكـتر عـشـان تـنـتـيـسـتـ missed items الـ coverage techniques اللي وقعت منها عـشـان نـزـوـدـ نسبةـ الـ coverageـ techniquesـ موجودـةـ بـتفـاصـيلـ أـكـترـ فـيـ شـابـتـرـ 4ـ وكـذـلـكـ فـيـ 2ـ partـ.

فى كل ال test levels (لكن بشـكـلـ خـاصـ فـيـ الـ component~testingـ والـ elementsـ code~coverageـ toolsـ) ممكن نـسـتـخـدمـ عـشـانـ نقـىـسـ الـ integration~testingـ معـيـنةـ زـىـ الـ statementsـ أوـ الـ decisionsـ الـ structural~testingـ، الـ architectureـ بتـاعـةـ السـيـسـتـمـ، زـىـ الـ calling~hierarchyـ.

ممكن بـرضـهـ نـفـذـ طـرـقـ الـ structural~testingـ عـلـىـ سـيـسـتـمـ أوـ الـ system~integrationـ اوـ الـ business~modelsـ (زـىـ الـ menu~structuresـ) acceptـingـ testing~levelsـ.

### Testing Related to changes: Re-testing and Regression Testing :2.3.4

بعد ما بنطلع ال defect ونحلـهـ، لازم نـعـملـ re-testـ عـلـىـ السـوـفـتـوـيرـ عـشـانـ نـتـأـكـدـ إنـ الـ defectـ اـتـحـلـتـ، دـهـ إـسـمـهـ الـ confirmation~testingـ، أـمـاـ الـ debuggingـ (تحـديـدـ سـبـبـ وـحـلـ الـ defectـ) دـهـ testing~activityـ مشـ development~activityـ.

ال regression testing هو إعادة التesting على برنامج اتعمل عليه تبـتـ لكن حـصـلـ عـلـيـهـ تعـديـلـ، وـدهـ عـشـانـ نـغـطـىـ أـىـ defectsـ جـديـدةـ طـلـعـتـ أوـ مـاـكـنـاشـ واـخـدـينـ بـالـنـاـ مـنـهـاـ كـنـتـيـجـةـ لـلـتـغـيـرـاتـ الـ related~or~environmentـ، الـ software~componentـ، وـدهـ بـيـتـعـملـ لـمـاـ يـتـغـيـرـ الـ softwareـ أوـ الـ unrelated~software~componentـ بتـاعـتهـ، والمـدىـ بتـاعـ الـ regression~testingـ بـيـقـىـ عـلـىـ أـسـاسـ الـ def~ectsـ فـىـ السـوـفـتـوـيرـ الـ كـنـاـ شـغـالـينـ عـلـيـهـ قـبـلـ كـدـهـ.

عمليات التيسننج لازم تتكرر لو تم استخدامها في ال confirmation testing وعشان نساعد ال regression testing .

ال non-functional test levels ممكن يتعمل على كل ال regression testing وبيشمل ال functional regression test suites، وال structural testing بيعملها run كذا مرة وبشكل عام بتتطور ببطء، عشان كده ال automation مرشح قوى لل regression testing .

## Maintenance Testing :2.4

### Background

بمجرد ما نعمل deploy للسيستم بيبقى عادة في الخدمة لسنين أو حتى عقود، في أثناء الوقت ده السيستم أو data configuration بحتاجه أو ال environment عادة بتتصفح أو بتتغير أو بتتمدد، عشان كده， فيه successful maintenance testing مهم جدا عشان نعمل planning of releases فرق بين ال planned releases والحلول الفورية، ال maintenance testing يتعمل على operational system أو migration أو modifications أو ال migration أو retirement للسوافتوير أو للسيستم.

التعديلات دى شاملة ال planned enhancement changes (e.g., release-based) والتغييرات ال corrective emergency وكذلك التغييرات على ال environment زى planned upgrade of database upgrades أو planned operating system أو ال batashat عشان نصح نقط الضعف المكتشفة فى ال operating system Commercial-Off-The-Shelf software .

ال maintenance testing لل platform migration (من changed software للثانية) لازم يشمل migration testing لل environment الجديدة وكذلك ال tests (Conversion testing) كمان بنحتاجه لما نقل داتا من أبليلكيشن تانى للأبليلكيشن بحتاجنا.

كمان بنحتاج ال maintenance testing لو بنعمل retirement للسيستم، وده بيشمل عمل تيسننج على ال data migration or archiving لو مطلوب الاحتفاظ بالداتا دى لوقت طويل.

بالإضافة للتيسننج ع اللي اتغير، ال maintenance testing بيشمل ال risk of maintenance testing بتاع ال scope مرتبط بال changes وحجم السيستم نفسه وحجم التغيير اللي حصل، واعتمادا على ال test levels أو كل ال maintenance testing ممكن يتعمل على أي نوع impact analysis من التيسيننج أو كلهم، وتحديد إزاي السيستم بيتأثر بالتغيير ده اسمه، وبيستخدم

للمساعدة في اتخاذ قرار نعمل regression testing قد إيه، ال impact analysis ممكن يستخدم لتحديد ال regression test suite.

من الممكن ان ال maintenance testing يبقى صعب لو ال specifications قديمة out of أو missing date أو testers خالص أو مافيش فكرة عامة عن السيستم.

## Chapter 3

### Static Techniques

#### :Static Techniques and the Test Process :3.1

##### Background

على عكس ال static testing الذى لا يلزم نعمل execution للسوالفتير، ال automated analysis بيعتمد على ال manual examination (reviews) وال techniques execution لل코드 أو أى project documentation (static analysis) تانى من غير مانعمل لل코드.

ال دى طريقة فى عمل التيسينج ع ال reviews (من ضمنهم الكود) ونقدر نعملها كويس قبل ال dynamic test execution (هنشرح النقطة دى أكثر فى part 2)، ونقدر نطلع ال reviews بدرى فى ال life cycle defects (زى مثلا ال defects بنلاقيها ف ال requirements)، ودى عادة أرخص فى حلها من إننا نطلعها واحنا بنعمل على الكود (هنشرح النقطة دى برضه بالتفصيل ف part 2).

ال review ممكن يتعمل كـ manual activity تماماً، لكن فيه برضه tool support activity الرئيسية هى إننا نشييك على ال work products ونكتب ال comments عليها، وأى requirements ممكن يتعمل عليه review وده شامل ال software product test plans وال الكود وال design specifications وال specifications user test scripts وال test cases وال specifications web pages وال guides.

##### :reviews

1. تحديد ال defect بدرى وتصحىحه.
2. تحسين إنتاجية ال rework development أقل.
3. بيقل الجداول الزمنية timescales فى عملية ال development.
4. تقليل وقت وتكلفة عملية التيسينج.
5. بالتالى تقليل تكلفة ال lifetime maintenance testing (عن طريق تقليل ال defects).
6. بيقل ال communication بين أفراد التيم كله.
7. بيحسن ال communication

كمان ال reviews يقدر يحدد الحاجات الساقطة منا سهوا omissions زى مثلا فى ال requirements اللي عادة ما بنلاقبهاش موجودة أثناء ال dynamic testing.

ال reviews وال static analysis dynamic testing ليهم نفس الهدف وهو التعرف على defects واكتشافها وبيكملوا بعض، وال techniques المختلفة تقدر تلاقى أنوع مختلفة من (defects) failures بتأقى أسباب ال defects بشكل فعال، فال static techniques بدلًا من dynamic testing نفسها اللي بيلاقفها ال failures.

ال defects فعلا اللي بتبقى أسهله اننا نلاقفها فى ال reviews عن اننا نلاقفها فى ال requirement defects standards المعروفة وال design defects وال incorrect interface وال insufficient maintainability وال specifications.

### What are static testing techniques and types?

دى ال techniques اللي ببها بنتيست السوفتوير من غير execution لل코드، ده بيشمل:

- عمل تيسينج على ال work-products الثانية غير الكود (زى ال requirements documents).
- عمل تيست على الكود لكن من غير execution فعلى ليه (زى مثلا بنراجع الكود وال standards وبنشوفه رايح فين وجاي منين وكده).

### :Types of static testing techniques

بنستخدم ال review عشان نلاقى ونعالج ال errors وال حاجات الغامضة فى development documents قبل ما تتسخدم فى عملية ال development، ده بيقلل واحد من مصادر defects فى الكود، ال reviews دى عادة بتتم manually.

هنا بنحلل الكود لاكتشاف ال structural defects أو ال programming weaknesses defects عادة static analysis tools automatically باستخدام automatically.

## :Review Process :3.2

### **Background**

الأنواع المختلفة من الـ reviews يختلف من **the informal** إلى **ما فيه تعليمات مكتوبة** **review** **لل systematic reviewers** **الى** **يتميز بمشاركة التيم وال results المكتوبة لل review** **والإجراءات المكتوبة لعمل the review**, **وال formality** **الى** **مرتبطة بعوامل معينة زى النضج maturity** **لعملية the development** **وأى requirements قانونية أو تنظيمية أو مطلوبة لعملية التدقير والمراجعة.**

الطريقة التي بيتفذ فيها the review بتعتمد على الأهداف المتفق عليها لل review (زى مثلا إيجاد defects أو اكتساب المعرفة أو تعليم team members وال testers الجداد أو النقاش واتخاذ القرار بالإجماع by consensus).

### **Activities of Formal Review :3.2.1**

#### اللى ليه ال formal review دى: main activities

##### **Planning -1** وده بيشمل:

- تعريف معايير المراجعة (المعايير اللي براجع على أساسها) .  
**criteria**
- اختيار الأشخاص (اللى هراجع معاه).
- تحديد الأدوار (للأشخاص) .  
**Allocating roles**
- تعريف **entry criteria** (هبدأ إمتي) وال **exit criteria** (توقف إمتي) لأنواع أكثر من **formal review** (زى مثلا **inspections** (هنجيله كمان شوية)).
- اختيار الأجزاء اللي هيتم **review** عليها فى **documents**.
- بنشيك على الـ **entry criteria** (أنواع أكثر من الـ **formal review**).

##### **Kick - off -2** وبنشمل:

- توزيع **documents** ع الناس اللي مختارهم.
- شرح **process** وال **participants** وال **documents** وال **objectives** اللي معايا ف الليلة دى.

### Individual preparation -3

- الإعداد للـ review meeting وأحضرها للـ documents بمراجعة الـ meeting الجاى.
- الإشارة للـ defects والأسئلة والـ comments المهمة وتسجيلها ف شكل notes.

### Examination/evaluation/recording of results (review meeting) -4

وبتشمل:

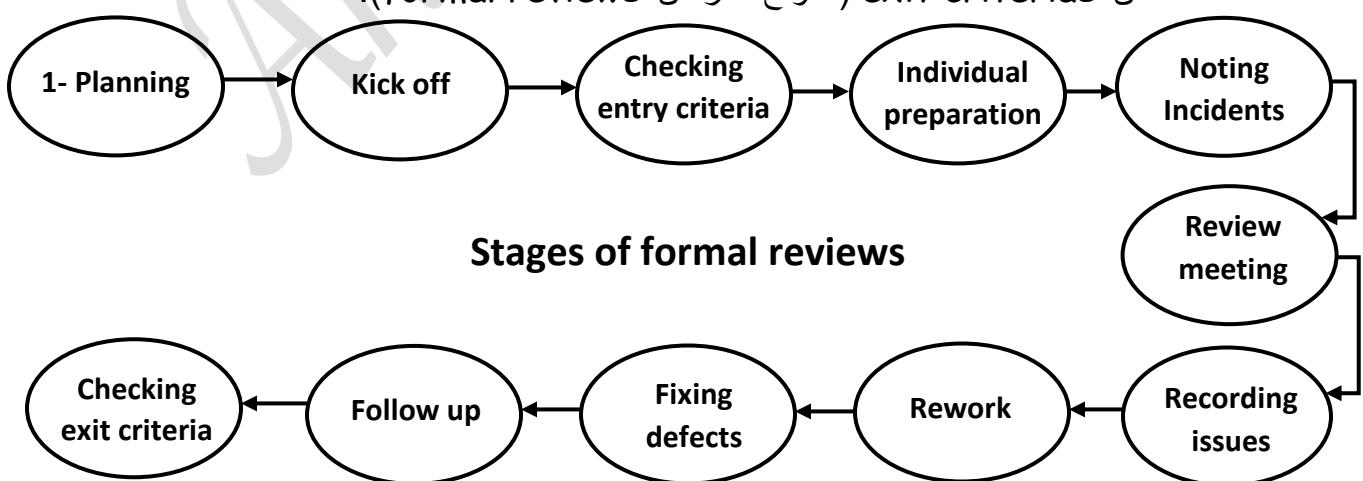
- المناقشة أو التسجيل (بأخذ feed back من الناس كلها)، مع تسجيل النتائج أو الدقائق (لأنواع أكثر من الـ formal review).
- تسجيل الـ defects اللي طلعتها وعمل الـ recommendations بخصوص معالجة أو حل الـ defects واتخاذ القرارات بخصوصها.
- دراسة / تقييم الـ issues وتسجيلها أثناء أى group physical meetings أو عمل أى electronic communications.

### Rework -5

- حل الـ defects اللي لاقيناها ف الميتيج (بيتم عن طريق الـ author اللي عاملها).
- تسجيل آخر حالة للـ defects (ف الـ formal reviews).

### Follow-up -6

- التأكيد إن الـ defects اتحلت.
- تجميع المقاييس gathering metrics (رزي عدد الـ defects والإحصائيات وكده ... الملخص يعني).
- التأكيد من الـ exit criteria (لأنواع أكثر من الـ formal review).



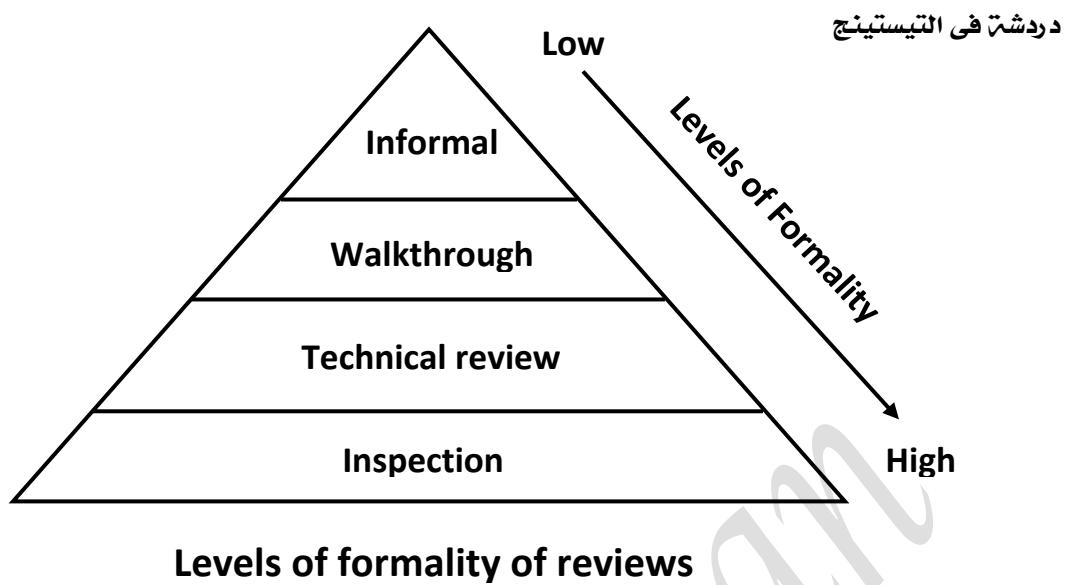
### Roles and Responsibilities :3.2.2

#### الـ roles بيشمل الـ formal review دى:

- **الـ manager :** بيأخذ قرار بخصوص تنفيذ الـ reviews وبيحدد الجدول الزمني للـ project .  
وبيحدد لو الهدف من الـ review اتحدد ولا لا .
  - **الـ moderator :** ده الشخص اللي بيقود الـ review لـ document أو لمجموعة documents ، ده بيشمل تخطيط لـ review وتشغيل الميتنج running the meeting .  
والمتابعة بعد الميتنج ، ولو ضروري ممكن الـ moderator يبقى وسيط بين وجهات النظر الكبير .  
وعادة بيكون الشخص اللي بيعتمد عليه نجاح الـ review .
  - **الـ Author :** ده اللي عمل الـ document(s) اللي بتتراجع أو المسئول مسئولية رئيسية عنها .
  - **الـ reviewers :** دول أشخاص ليهم مواصفات technical معينة أو ليهم business background ع الموضوع (ممكن يتسموا inspectors أو checkers ) .  
بعد الإعداد الضروري - بيتعرفوا على الـ findings فى الـ product اللي بيترافق (زى defects مثلًا) ويوصفوا ، والـ reviewers دول لازم يتم اختيارهم عشان يعبروا عن وجهات نظر و roles مختلفة فى عملية الـ review ، ولازم يشاركون فى أي meetings .
  - **الـ recorder (أو الـ scribe) :** ده بيـ كل الـ document issues والـ problems والنقط المفتوحة اللي اتحددت خلال الميتنج .
- النظر لـ software products أو لـ work product المرتبط بالموضوع من وجهات نظر مختلفة واستخدام الـ reviews ممكن يخلى الـ checklist بكافأة وفاعلية أكثر ، زى مثلاً معمولة على أساس وجهات نظر مختلفة زى اليوزر user أو maintainer أو التيسير أو operations او الـ requirements لمشاكل الـ checklist ممكن تساعده فى كشف المشكلات اللي ماتمش اكتشفها قبل كده .

### Types of Reviews :3.2.3

الـ related work product أو الـ single software product ممكن يبقى موضوع للمناقشة فى أكثر من review واحد ، فلو تم استخدام أكثر من نوع واحد فـ review فممكن يختلف الترتيب ، فمثلاً الـ informal review ممكن يحصل قبل الـ technical review ، أو الـ inspection ممكن يحصل على الـ requirements specification قبل الـ walkthrough مع العملاء .



الخصائص الرئيسية والـ options والأغراض من أنواع الـ review المعروفة هي:

### Informal Review

- مافيش عمليات بتنم بشكل رسمي (كله بشكل غير رسمي).
- ممكن ياخد شكل الـ technical pair programming أو حد reviewing بيقود عملية الـ designs والكود.
- النتائج ممكن تبقى documented.
- بختلف الاستفادة على حسب الـ reviewers.
- الهدف الرئيسي: طريقة رخيصة لاكتساب الفائدة.

### Walkthrough

- الـ author هو اللي بيقود الميتنج.
- ممكن ياخد شكل الـ scenarios, dry runs, peer group participation
- Open-ended sessions
- ممكن يتعمل الـ pre-meeting preparation للـ reviewers
- ممكن إعداد الـ review report فيه list of findings
- ممكن يكون فيه كاتب (غير الـ author).
- ممكن يختلف فـ informal practice إلى حد ما فالـ very formal
- الهدف الرئيسي: التعلم – اكتساب الفهم – إيجاد الـ defects

## Technical Review

- بيبقى documented وبيتعرف فى المرحله دى على عملية defect-detection اللي
- بتشمل peers أو technical experts مع إمكانية مشاركة الإداره.
- ممكن يتعملى peer review من غير مشاركة الإداره.
- غالبا اللي بيقود ال review (مش author) trained moderator حد
- ال pre-meeting preparation يعملوا reviewers
- ممكن يتم استخدام ال checklists
- إعداد ال review report اللي بيشمل ال list of findings والحكم إذا كان ال software
- بتحقق ال requirements ولا لأ، وكمان - منين مايبيقى مناسب - findings recommendations
- ممكن يختلف ف ال practice ما بين ال quite informal لغاية ال very formal
- الأهداف الرئيسية: المناقشه - اتخاذ القرارات - تقدير البديل - إيجاد defects - حل المشاكل
- ال technical - والتأكد من التوافق بين ال specifications اللي طالبها العميل وال standards واللوائح وال

## Inspection

- ال inspection هو اللي بيقود ال trained moderator (مش author)
- عادة بتعملى peer examination
- ال roles معروفة ومحددة.
- بتشمل استخلاص المقاييس metrics gathering
- عملية formal تعتمد على ال rules وال checklists
- بيبقى فيه software product entry and exit criteria محددة لقبول ال
- بيبقى فيها pre-meeting preparation
- بيبقى فيه list of findings inspection report يحتوى على
- Formal follow-up process (with optional process improvement components)
- ممكن يكون فيه reader
- الهدف الرئيسي: إيجاد defects

**peer** ممكن يتعلموا خلال **inspections** وال**technical reviews** وال**walkthroughs** مجموعة من ال**colleagues** في نفس ال**organizational level** (مجموعه من ال**group**)، النوع ده من **“peer review”** اسمه **review**.

### **:Success Factors for Reviews :3.2.4**

#### عوامل نجاح ال reviews بتشمل:

- كل review ليه **predefined objectives** واضحة.
- اشتراك الناس المناسبين لأهداف ال**review**.
- ال**reviewers** مهمين وده لأنهم بيساهموا في عملية ال**review** وكمان **testers** يعتبروا مهمين (زى مثلا ان الكلام ده يتحول لخبرة إضافية بيتعلموا أكثر عن الـ **product** اللي بيخليهم يحضروا الـ **tests** بتاعتهم بدري).
- المكتشفة مرحباً بيها وبتعرض بموضوعية.
- مشاكل الناس والنواحي النفسية بيتم التعامل معها (زى مثلا ان الكلام ده يتتحول لخبرة إضافية للـ **author**).
- الـ **review** بيتم في جو من الثقة، والـ **outcome** مش هيستخدم في تقييم المشاركون في الـ **review**.
- الـ **review techniques** اللي بتنطلب لازم تكون مناسبة لتحقيق الأهداف اللي عايزينها وكمان تكون مناسبة لنوع ومستوى الـ **reviewers** والـ **software work products**.
- الـ **roles** أو الـ **checklists** أو الـ **defect** أو الـ **formal techniques** عمل تدريب في الـ **review techniques** خصوصاً أكثر أنواع الـ **review techniques** زي الـ **inspection**.
- تدعيم الإدارة لعملية الـ **review** بشكل جيد (زي مثلا الدمج وقت كافى للـ **review activities** في الـ **project schedules** فيه تركيز على التعلم وتحسين العملية).

## :Static Analysis by Tools :3.3

### **Background**

الهدف من الـ static analysis هو إيجاد الـ defects في الـ software source code، والـ static analysis يتعمل فعلياً من غير مايشتغل السوفتوير اللي عليه الكلام عن طريق الـ tool اللي بتشغله، على عكس الـ dynamic testing اللي بي execute static analysis على عكس الـ defects اللي صعب نلاقيها في الـ software code، الـ static analysis يقدر يحدد الـ defects اللي صعب نلاقيها في الـ static analysis reviews، وأنشاء الـ static analysis dynamic testing بتلقي الـ defects بدلًا من الـ control flow tools، الـ static analysis failures بتحليل كود البرنامج (زى مثلاً الـ XML failures والـ XML generated output) وكذلك الـ HTML failures والـ data flow failures.

### قيمة الـ static analysis تكمن فى:

- تحديد الـ defects بدري قبل الـ test execution.
- بيدينا إنذار مبكر بخصوص الأوجه المريبة للكود أو الـ metrics زى high complexity measure.
- التعرف على الـ defects اللي مش سهل نلاقيها فى الـ dynamic testing.
- تحديد التبعيات dependencies والـ inconsistencies.
- رجوع الـ links زى الـ models.
- تحسين الـ maintainability للكود والـ dizain.
- منع الـ defects لو تم تعلم الدروس فى الـ development.

### الـ static analysis tools اللي بيكتشفها الـ defects بتشمل:

- رجوع الـ undefined value بـ variable.
- التعارض فى الـ interfaces بين الـ components والـ modules.
- الـ variables الغير مستخدمة أو اتعلملها declare بشكل غير صحيح.
- الـ Unreachable (dead) code.
- خطأ أو missing فى الـ logic (زى الـ infinite loops).
- الـ constructs المعقدة جداً.
- الاختلاف عن الـ programming standards.

- نقاط الضعف الأمنية .**security vulnerabilities**
- الاختلافات ف ال syntax سواء ف الكود أو فى ال software models

ال predefined rules أو بيشيكوا على ال static analysis tools (بخدمها ال ديفلوبرز) قبل وأثناء ال component testing وال integration standards (programming standards) أو لما تشيك ع الكود عن طريق ال configuration management tools، وكذلك testing tools أو static analysis tools، ال designers ممكن static analysis tools، ال modeling software designers، ال warning messages اللي بنحتاجها عشان ندىرها كويس عشان نستفيد أقصى استفادة من ال tool.

metrics ممكن يعرض نوع من الدعم لل static analysis compilers، وده بيشمل حساب ال static analysis tool لل compiler أشهر أشهر (ال النقطة دي مهمة جدااا).

## Chapter 4

### Test Design Techniques

#### :The Test Development Process :4.1

##### **Background**

عملية الـ **test development** الموضحة فـ **الجزء ده ممكن تتعمل بطريق مختلفة من الـ very formal documentation** اللي فيه **informal** قليلة أو ما فيهاش خالص للـ **very formal** (زى ما هنوضح بعد كده)، ومستوى الـ **formality** ده بيعتمد على محتوى التيسينج، وده بيشمل الـ **maturity** لعملية **safety or regulatory** والـ **time constraints** والـ **development** والـ **test conditions**، والناس المشتركين فـ **الحوار ده requirements**.

أثناء الـ **test analysis** ببitem تحليل الـ **test basis documentation** عشان نحدد نعمل تيست على إيه بالظبط، زى مثلا تحديد الـ **test conditions**.

ده الـ **item** أو الـ **event** اللي ممكن تتأكد من صحته بـ **test case** أو أكثر زى مثلا الـ **function, transaction, quality characteristic or structural element**.

**يعنى م الأخر الـ **test condition** هو الـ **item to be test****

عمل تتابع **traceability** من الـ **test conditions** لورا لـ **specifications** والـ **requirements**، بيمكننا من عمل **impact analysis** مؤثر لما تتغير الـ **requirements** وكذلك تحديد الـ **tests** لمجموعة من الـ **requirements coverage**، أثناء الـ **test analysis** بتنفذ عشان اختيار الـ **detailed test approach** اللي **test design techniques** هنستخدمها على أساس الـ **identified risks** (من بين الاعتبارات الثانية)، وهنعرف أكثر عن الـ **risk analysis** فى شابت 5.

أثناء الـ **test design** يتم عمل وتحديد الـ **test cases** والـ **test data**، **الـ test case** يتكون من مجموعة من **expected execution preconditions** والـ **input values** والـ **execution postconditions results** دى بنعرفها عشان نقطى **test case**، والـ **test condition** معينة أو **objective(s)** 'Standard for Software Test Documentation' (IEEE STD 829-1998) **test design** بيوصف محتوى الـ **Test Documentation** (IEEE STD 829-1998) . **test case specifications** والـ **test conditions specifications**

الـ **expected results** لازم ت العمل كجزء من مواصفات **test case** وبتشمل الـ **outputs** والتغييرات سواء للبيانات أو الحالات **states**، وأى آثار تانية للتيست، لو الـ **expected result** مش **defined** يبقى كده النتائج المعقوله لكنها خطأ ممكن تتقدم على إنها النتائج الصح، الـ **test execution results** لازم تعرف طبقا للـ **test case**.

أثناء الـ **implementation** **develop** **test cases** **test implementation** **test procedure specification** (IEEE STD 829-1998) **organize** **prioritize** **execution sequence of actions** **test procedure** عشان نعمل **sequence of actions** **tests** **test execution tool** run ب باستخدام **test execution tool** ببقي الـ **test script** (automated test procedure) ده الـ **test script** متعدد فى **actions**.

الـ **test procedures** الكثيرة والـ **automated test scripts** بتحول بعد كده لـ **test execution schedule** اللي بحدد الترتيب اللي على أساسه بيعمل **test execution** لـ **test execution** **scripts** الكثيرة والـ **automated test scripts** المحتملة، والـ **prioritization** **regression tests** ده بياخد ف الاعتبار شوية عوامل زى الـ **schedule** والـ **telluric factors** كانت **technical** أو **logical**.

## :Categories of Test Design Techniques :4.2

### **Background**

الهدف من الـ **test cases** هو التعرف على الـ **test design technique** والـ **test conditions** والـ **test data**.

من الطرق الكلاسيكية التي يندرج تحتها **black-box test techniques** (بما يشمل **black-box test design technique**، **black-box specification**، **black-box test conditions** أو **black-box based techniques**) طريقة لاستخلاص و اختيار الـ **test cases** (based on **functional test basis documentation** أو **test data** على أساس تحليل الـ **functional test basis documentation**، وده بيشمل الـ **test data** على أساس تحليل الـ **functional test basis documentation**). طبقاً للكلام ده **black-box testing** مابيستخدمش أي معلومات مرتبطة بالـ **internal structure** أو **system** التي بنعمل عليه **white-box test design techniques** (المعروف كمان باسم **white-box specification** أو **white-box structure**) فمرتكز على أساس تحليل الـ **internal structure** أو **system** للـ **component** أو **system** أو **component** white-box testing black-box testing ممكن كمان white-box testing black-box testing للـ **component** أو **system** أو **component** experience-based techniques يتجمعوا مع الـ **developers** للتأثير بالإيجاب على خبرة الـ **developers** والـ **testers** والـ **users** لتحديد إيه اللي لازم يتعلمه تيست.

بعض الـ **techniques** يندرجون في **single category**، وبعض الثاني عنده عناصر من أكثر من **category**.

المنهج يتبع عن الـ **specification-based test design techniques** باسم **specification-based test design techniques** والـ **black-box techniques** بالإضافة إلى إن المنهج كمان بيغطي الـ **white-box techniques** .**test design techniques**

### الخصائص المعروفة للـ **specification-based (black box) test design techniques** :

- الـ **models** - سواء **formal** أو **informal** - يستخدم كـ **specification** للمشكلة اللي عايزة نحلها أو للسوفتوير أو مكوناته.
- الـ **models** ممكن ناخدها بشكل منهجي من الـ **test cases** دى.

### أما الخصائص المعروفة للstructure-based (white box) test design techniques بتشمل:

- المعلومات الخاصة بالبنية بتاعة السوفتوير اللي بتسخدم لاستخراج ال test cases (زى مثلا الكود وال detailed design information).
- ممكن قياس مدى تغطية ال test cases للسوفتوير، وكذلك ممكن نكتر ال test cases بشكل منظم لزيادة التغطية دى.

### أما الخصائص المعروفة للexperience-based test design techniques بتشمل:

- المعرفة وخبرة الناس بيتم استخدامهم فى عمل ال test cases .
- المعرفة الخاصة بال stakeholders وال users وال developers وال testers عن environment بتعاتهم دول كلهم مصدر للمعلومات.
- معرفة ال defects المحتملة وتوزيعها ده مصدر تانى للمعلومات.

## :Specification-based or Black-box Techniques :4.3

الجزء ده هنتكلم عنه دلوقتى زى ما هو موجود ف ال syllabus بالظبط، و هننشرحه باستفاضة أكثر و ازاي نشغل بيه وكده فى part 2 إن شاء الله.

### **:Equivalence Partitioning :4.3.1**

فى ال equivalence partitioning بتنقسم ال inputs فى السوفتوير أو السيستم لمجموعات متوقع انها يبقى ليها behavior مشابه، عشان كده بيتم التعامل معاهם بنفس الطريقة.

ال values (زى ال valid data) نقدر نعملها فى ال equivalence partitions (or classes) المقبولة (زى ال invalid data) ، كمان ممكن ال partitions تتعملى event وال time-related values وال internal values وال outputs معين) وال integrated components (زى ال interface parameters اللي بنعملها تيسىت أثناء مرحلة ال integration testing)، ال tests ممكن يتعملها ديزاين عشان تغطى كل ال levels of equivalence partitioning، وال invalid partitions testing.

ال input and output ممكن يستخدم لتحقيق أهداف ال equivalence partitioning، وممكن يتعمل على ال inputs أو ال human input عن طريق ال interfaces لسيستم معين أو ال interface parameters .integration testing في ال interface parameters.

### **:Boundary Value Analysis (BVA) :4.3.2**

ال behavior في ال edge لكل equivalence partition بيقى احتمال الخطأ فيه أكبر من behavior خلال ال partition من جهة، عشان كده boundaries هى المنطقه اللي عادة التيسينج بيلقى فيها ال defects، ال minimum values وال maximum values معين هى دى ال valid partition لل boundary value، فال invalid partition لل boundary value هى invalid boundary value، وال invalid boundary partition لل boundary value هى invalid boundary values وال valid design tests ممكن يتعملها عشان تغطى ال valid design test cases نعمل design لل boundary value .boundary value

ال boundary value analysis ممكن يتعمل على كل ال test levels، ده لأنه سهل في عمله وإحتمال اكتشاف ال defect فيه كبير، ال detailed specifications هتساعدنا في تحديد ال boundaries اللي عليها الكلام.

ال black-box test ده عادة بيعتبر امتداد لل equivalence partitioning أو technique لوحده، وممكن يتعمل على classes متساوية لل user input على transactional speed زى مثلا على ال time ranges (زى مثلا time out) وال screen (زى مثلا table ranges requirements أو table ranges (requirements

### **:Decision Table (DT) Testing :4.3.3**

ال decision tables هي طريقة كويسة للشغل على ال system requirements اللي بتحتوى على logical conditions، وكمان عشان نكتب ال internal system design، وممكن كمان تستخدم لتسجيل ال business rules المعقدة للسيستم اللي عليه الكلام، فلما نعمل ال decision table specification tables بتتطلب وبتعرف ال actions conditions وال conditions بتاعة السيستم، ال actions input conditions وال input conditions غالبا بتعرض بطريقة معينة بحيث انها لازم تكون يا true يا false (يعنى)، وال decision table يحتوى على ال conditions اللي قلنا عليها دى وعادة كمان بتحتوى على مجموعة من ال true وال false لكل ال input conditions وال resulting actions لكل مجموعة من ال conditions، وكل column فى الجدول بيتوافق مع

business rule معينة بتعرف conditions من الـ unique combination وأيا كانت نتيجتها فى تنفيذ الـ actions فهى متفقة مع الـ rule دى، الـ coverage standard المستخدم عادة مع الـ decision table testing هو إنه ع الأقل فيه test واحد لكل column ف الجدول، واللى فعليا بيغطى كل الـ conditions بقاعة الـ combinations.

قوة الـ decision table testing تكمن فى إنه بيعمل combination من الـ conditions اللي من غيره يمكن مانقرش نفكر فيها أثناء عملية التيستينج، وممكن نعمله على كل الـ situations اللي فيها السوفتوير بيعتمد على logical decisions كتيرة.

#### State Transition Testing :4.3.4

الـ system ممكن يعرض اعتمادا على الـ conditions الحالية أو السابقة (الحالات بتاعتة)، ففى الحالة دى الـ system ممكن نعبر عنه بالـ state transition diagram، لإنه بيسمح للـ tester إنه يعرض السوفتوير فى كل حالاته والتنقلات ما بين الحالات states دى والـ inputs أو transitions اللي بتغير الحالة events (transitions) والـ actions اللي ممكن تنتج عن الـ transitions، الحالات بتاعة الـ system أو الـ object under test دى بتبقى منفصلة ومعروفة ومتحددة برقم نهائى.

الـ state table بيظهر العلاقة بين الـ states والـ transitions، ويقدر يجيب الـ المكنة invalid.

الـ tests ممكن نعملها ديزاين عشان نقطى sequence معين من الـ states أو عشان نقطى كل state أو عشان ننفذ كل transition أو عشان ننفذ sequences معينة من الـ transitions أو عشان نتيست الـ invalid transitions.

الـ embedded software industry State transition testing بيستخدم بكثرة فى الـ technique بشكل عام، مع ذلك الـ technical automation ده كمان مناسب لعمل تصميم business object ليه حالات محددة أو عمل تيستينج على الـ screen-dialogue flows (زى (business scenarios أو الـ internet applications).

### :Use Case Testing :4.3.5

نقدر نعمل الـ tests اللي احنا عايزينها من الـ use cases، والـ use cases دى بتوصف الـ interactions ما بين الـ actors (سواء users أو systems) اللي بطلع نتيجة ليها قيمة abstract level use cases، customer أو للـ system user (business use case, technology-free, business process level) أو فى الـ system (system use case on the system functionality level) use case، وكل ليها level (system use case on the system functionality level) المطلوبة عشان الـ use case تشتعل بنجاح، وكل use case بتتنهى بـ postconditions اللي هى النتائج اللي هتحصل والحالة النهائية للسيستم بعد ما تكمل الـ use case، عادة الـ use case ليها سيناريو رئيسى mainstream scenario وسيناريوهات بديلة alternative scenarios.

**use cases** بـ "Process flows" فى السيستم على أساس الاستخدام الفعلى المعاد عليه، عشان كده الـ test cases نقدر نطلعها من الـ use cases ودى بتبقى مفيدة جدا فى اكتشاف defects الغير مكتشفة أثناء الاستخدام الفعلى للسيستم فى الواقع الحقيقى، كمان الـ use cases مفيدة جدا فى عمل الديزاين للـ acceptance tests اللي بيكون بمشاركة الـ customer أو الـ user بتاع السيستم، كمان بتساعدنا فى اكتشاف الـ integration defects الغير مكتشفة عن طريق الـ interaction و التداخل interference ما بين الـ components المختلفة، وف الحاله دى بنبع individual component interaction بين الـ components بس بغض النظر عن كل use cases design من الـ test cases ممكن يتجمع مع الـ specification لوحده فىهم، والـ white-box testing من الـ test cases design ممكن يتجتمع مع الـ based test techniques.

### :Structure-based or White-box Techniques :4.4

#### Background

الـ structure-based white-box testing أو الـ structure-based white-box testing بيرنكر على الـ identified structure للـ sofwtware أو السيستم، **زى ما هو موضح فى الأمثلة دى:**

- الـ component level: ده الـ software component بتاع الـ structure زى مثلا .distinct paths والـ branches decisions والـ statements أو حتى الـ branches decisions statements
- الـ integration level: هنا الـ structure call tree ممكن بيقى diagram ده فيه modules بتتادى على الـ modules تانية.
- الـ system level: الـ menu structure أو business structure ممكن تبقى system level .web page structure أو process

في الجزء ده الكتاب هيناقش 3 code-related structural test design techniques على أساس الـ statements والـ branches والـ decisions للـ code coverage، وممكن نستخدم الـ control flow diagram عشان نوضح البدائل لكل قرار.

#### **:Statement Testing and Coverage :4.4.1**

في الـ statement coverage، الـ component testing هى تقييم النسبة المئوية للـ executable statements اللي اتنفذت عن طريق الـ test case suite، الـ statement testing technique عشان ننفذ statements معينة، عادة عشان نزود الـ statement coverage.

نقدر نحسب الـ statement coverage عن طريق اننا نقسم عدد الـ statements اللي بتغطيها الـ test case (أو اتعملها ديزاين) على عدد كل الـ executable statements فى الكود، يعني نقدر نعبر عنها بالمعادلة دي:

$$\text{Statement coverage} = \frac{\text{no. of executed statements}}{\text{no. of all statements in the code under test}}$$

#### **:Decision Testing and Coverage :4.4.2**

الـ decision coverage – هو تقييم النسبة المئوية للـ decision coverage – مرتبط بالـ branch testing (زى مثلا الـ IF لجملة True & False options) اللي اتنفذت عن طريق الـ test cases execute decision testing technique، الـ case suite decision points اللي طلعين من الـ decision outcomes بيتفرع من الـ branches معين، الـ decision outcomes ف الكود و بتبيّن النقلة فى الـ control flow different parts of the code.

الـ decision coverage بنحسبيه بعدد الـ decision outcomes اللي بتغطيها الـ test case (سواء بالديزاين أو execute) مقسوم على عدد كل الـ decision outcomes فى الكود اللي بنعمل عليه التيس، يعني نقدر نعبر عنه بالمعادلة دي:

$$\text{Decision coverage} = \frac{\text{no. of executed Decisions}}{\text{no. of all Decisions in the code under test}}$$

الـ control flow testing هو نوع من الـ decision testing لأنّه بيتابع flow معين للـ control flow testing، Statement coverage، الـ Decision coverage، decision points خلال الـ

ف 100% statement coverage 100% decision coverage، لكن العكس غير صحيح.

الكلام ده هيترسح بتفاصيل أكثر في part 2 إن شاء الله.

#### :Other Structure-based Techniques :4.4.3

فيه أقوى من structural coverage levels، زي مثلاً decision coverage .multiple condition coverage وال condition coverage

التابع coverage concept نقدر كمان نطبقه على test levels تانين، يعني مثلاً في integration level النسبة المئوية لل classes أو components أو اللى module, component, or class ممكن نعبر عنها ب test case suite .coverage

ال structural testing of code tool support بيبقى مفيد في

:Experience-based Techniques :4.5

## Background

ال**tests** هو ال**Experience-based testing** المستخلصة من مهارة وحدس التيسير وخبرته بال**techniques** المشابهة، لما يتم استخدام ال**techniques** دى بشكل منهجه بتبقى مفيدة فى تحديد **tests** خاصة مش بسهولة نقدر نعملها بال**formal techniques**، خصوصا لما نعملها بعد **formal approaches** أكثر، ومع ذلك ال**technique** ده ممكن يتعملى على نطاق واسع بدرجات مختلفة من التأثير، وده على حسب خبرة التيسير.



ال**experience-based technique** هو **error guessing**، بشكل عام فكرته قائمة على إن التيسير **defects** بيتوقيع الـ **defects** بناء على خبرته، والطريقة المثالية فى الـ **error guessing technique** هو إن التيسير يحضر ليسته من الـ **defects** المحتملة أو اللي بيتوقيعها ويعمل ديزاين للـ **tests** الـ **attack** دى، والطريقة دى اسمها **fault attack**، وهنا بتكون الـ **defect and failure lists** معهولة بناء على الخبرة والـ **failure data** ومن برضه المعرفة العامة عن أسباب الـ **fail** فالسوفتوير.

ال**test design** هو تكنيك متزامن concurrent فى الـ **Exploratory testing** والـ **test execution** والـ **test logging** والـ **learning**، بناء على الـ **test charter** اللي بتحتوى على الـ **test objectives**، وبيحصل الكلام ده خلال **time-boxes**، فدى طريقة مفيدة جدا لما يبقى عندنا **specifications** قليلة أو مش كافية أو **few or inadequate specifications** وفيه ضغط فى الوقت، أو عشان نزود أو نكمel الـ **formal testing** أكثر، فممكن نعتبر اننا بنشيك على عملية التيسير عشان نتأكد إن احنا لاقينا الـ **defects** الخطيرة جدا.

## :Choosing Test Techniques :4.6

### **Background**

اختيار أي تكنيك من الـ **test techniques** عشان نستعمله بيتوقف على عدد من العوامل من ضمنهم نوع السيستم والمعايير التنظيمية **regulatory standards** والـ **requirements** سواء اللي طالبها العميل أو موجودة ف العقد ومستوى الـ **risk** ونوعه والهدف من التيسير والـ **documentation** المتاحة ومعرفة الـ **testers** والوقت والميزانية **development life cycle budget** والـ **use case models** اللي ماشى عليها والـ **defects** والخبرة السابقة بأنواع الـ **defects** اللي لاقيناها.

بعض الـ **techniques** مناسبة أكثر لحالات **test levels** معينة، وبعضهم ممكن يكون مناسب لكل الـ **test levels**.

فلما بنعمل الـ **test cases** بيتستخدم الـ **testers** بشكل عام مجموعة من الـ **test techniques** شاملة الـ **process, rule, and data-driven techniques** للتأكد إنهم عملوا تغطية كافية للـ **object** اللي بيتعمل عليه التيسير.

## Chapter 5

### Test Management

#### :Test Organization :5.1

##### :Test Organization and Independence :5.1.1

بيقولك إن تأثير إيجاد ال defects في التيسينج وال reviews ممكن تحسن عن طريق استخدام independence options، وال independent testers يتضمن الآتي:

- ما فيش independent testers خالص... الديفلوبرز بيعملوا تيست ع الكود بتاعهم.
- Independent testers موجودين ف ال development teams
- reports فى الشركة، وبيعملوا ال Independent test team or group
- executive management أو ال project management لل
- Independent testers من ال business organization اللي طالبة السوق توير ده أو من ال user community اللي بيستعمله.
- usability Independent test specialists
- certification testers أو ال security testers (اللي بي certify السوق توير اللي بيتعمل طبقاً لقواعد و standards معينة).
- Independent testers outsourced or external to the organization

بيقولك إنه في المشاريع الكبيرة أو المعقّدة أو ال safety critical بيبقى عادة من الأفضل استخدام أكثر من level ف التيسينج، معظم أو كل ال levels دي بيعملها ال independent testers، ولكن يمكن ال development staff يشتراكوا كمان ف عملية التيسينج خصوصاً في ال lower levels، لكن قلة الموضوعية عندهم بتخلّى تأثيرهم محدود (الإنسان عادة ما يعرّف نفس يشوف أخطاؤه وما يحبش يطلعها لنفسه)، ال independent testers ممكن يبقى لهم صلاحيات طلب وتعريف عمليات التيست وقواعدها، لكن لازم ال testers ياخدوا ال process-related roles بس في حالة وجود تصريح واضح من الإدارة بده.

### من ضمن مميزات الـ independence :

- الـ independent testers يبيشووا defects تانية و مختلفة و بدون تحيز (زى الـ diفلوبرز).
- الـ independent tester يقدر يتأكد من الـ assumptions اللي افترضتها الناس أثناء الـ implementation specification للسيستم.

### أما عيوب الـ independence فبتشمل:

- الانعزال عن الـ development team (لو هما totally independent)، وده بيخلط التواصل مع الـ diفلوبرز أصعب.
- الـ diفلوبرز ممكن يفقدوا الإحساس بالمسؤولية عن الـ quality (لإن كل حاجة هي Shirleyها التيسيرز).
- ممكن يتصل للـ independent testers على إنهم عنق الزجاجة أو إنهم السبب فى تأجيل الـ release اللي هيطلع.

الـ testing tasks ممكن يعملها الناس فى testing role معين، أو ممكن يعملها حد فى الـ business and project manager أو الـ quality manager أو الـ IT operations أو الـ infrastructure أو الـ domain expert.

### **:Tasks of the Test Leader and Tester :5.1.2**

المنهج عندنا بيغطي 2 positions، التيسير والـ test leader والـ activities والـ tasks اللي بيعملها الناس فى الـ 2 roles دول بيعتمدوا على الـ project والـ product context والناس اللي ف الـ organization دى والـ roles.

أحيانا الـ test leader بيبقى إسمه test coordinator أو الـ test manager، الـ test leader بتاع الـ test leader ممكن يعمله الـ project manager أو الـ development manager أو الـ manager of a test group أو الـ quality assurance manager، أما فى المشاريع الكبيرة فيه 2 positions ممكن يبقوا موجودين: الـ test leader والـ test manager، الـ test leader عادة بيخطط للـ test activities and tasks وبيراقبهم وبيعملهم الـ control اللازم زى ما قلنا فى النقطة 1.4.

الـ **test leader tasks** المثلية تتضمن:

- التنسيق مابين الـ **test strategy** وباقي الناس فى الـ **project managers** والـ **test strategy** .  
**plan** .كتابة أو مراجعة الـ **test policy** للمشروع والـ **test strategy** للـ **organization** •  
المساهمة ببصمة التيسينج فى الـ **project activities** زى الـ **integration** •  
**planning** .  
عمل **tests plan** (مع الأخذ ف الاعتبار الـ **context** وفهم الـ **test objectives**)، ده شامل اختيار الـ **test approaches** وتقدير الوقت والـ **effort** والتكلفة  
بتاعتة التيسينج وطلب الـ **resources** وتعريف الـ **test levels** والـ **cycles** وتحطيط  
الـ **incident management** .  
البدء فى عمل الـ **specification** والـ **preparation** والـ **implementation** والـ **execution** للـ **tests** للـ **check** على الـ **exit criteria** •  
التكيف مع الـ **planning** بناء على نتائج التيسىت والـ **progress** اللي بيتم (أحياناً بيقى  
فى شكل **status reports** واتخاذ أى **action** ضروري عشان نعوض  
المشاكل دي. •  
تسطيب المقدار الكافى من الـ **testware configuration management** للـ **testware** عشان نقدر  
**traceability** .  
تقديم المقاييس المناسبة لقياس الـ **test progress** وتقييم جودة التيسينج والـ **product** •  
اتخاذ القرار بخصوص إيه اللي هييقى **automated** ولاي درجة وإزاي هيتم. •  
اختيار الـ **tools** لدعم التيسينج وتنظيم أى **training** للتيسترز فى استخدام الـ **tools** دى. •  
اتخاذ القرار بخصوص الـ **test environment implementation** للـ **test environment** •  
كتابة الـ **test summary reports** بناء على المعلومات المستنيرة أثناء التيسينج. •

### أما الـ **tester tasks** المثالية ممكناً تشمل:

- **review** والمساهمة في **test plans**.
- تحليل ومراجعة وتقدير **user requirements** وال**models specifications**.
- **testability**.
- عمل **test specifications**.
- تسطيب **test environment** (عادة يتم بالتنسيق مع **system administration**).
- **(network management)** وال**الـ test data**.
- إعداد والحصول على **expected results document**.
- تنفيذ التيسير على كل **test levels**، وتسجيل **tests** التي تتضمن وتقدير النتائج وعمل **test monitoring** **test administration or management tools**.
- استخدام **tools** زى ما هو مطلوب.
- **Automate tests** (ممكناً يكون فيه دعم من **الـ developer** أو من **expert**).
- قياس **systems performance** لل**components** وال**systems** (إذا فيه إمكانية).
- مراجعة **tests** التي عملها ناس تانية.

الناس التي بتشغل على **test analysis** وال**test design** وأنواع معينة من **test** أو **test automation** ممكناً يبقوا متخصصين في **roles** ذى، واعتماداً على **test level** وال**risks** وال**test automation** المرتبطة بال**product** وال**project** فممكناً ناس مختلفة تقوم بدور التيسير، وده بيحافظ على درجة ما من **independence**، فعادة التيسير في **component** وال**integration level** ممكناً يبقوا **business experts and acceptance test level** ممكناً يبقوا **acceptance test level**، والتيسير في **operators** وال**users**، والتيسير في **operational acceptance testing**.

## :Test Planning and Estimation :5.2

### :Test Planning :5.2.1

هنتكلم في الجزء ده عن الغرض من **test planning** خلال **development** وال**implementation** وال**maintenance activities** وكذلك لل**projects**، فممكناً **test planning** يكون **documented** سواء في **master test plan** أو في **separate test plans** التي بنعملها لكل **test level** زى مثلاً **system testing** وال**acceptance testing**، والشكل العام

للى 'Standard for Software Test Planning' موجود فى ال Documentation' (IEEE Std 829-1998)

scope of testing organization وال planning test policy فى ال testability وال criticality وال constraints وال risks وال objectives وال ممكن test planning progress project، وكمادة أى availability of resources وال يكون فيه معلومات وتفاصيل أكثر تبقى متاحة فممكن ندخلها معانا في ال plan.

activity test planning هى مستمرة ويتعمل فى كل ال life cycle processes and changing activities من ال test activities عشان نتعرف على ال feedback activities، وبنستخدم ال risks planning عشان نقدر نعدل ال

## **:Test Planning Activities :5.2.2**

**الـ test planning activities للـsystem كلـه أو جـزء مـنـه مـمـكـن يـشـمل:**

- تحديد ال risks وال scope والتعرف على أهداف عملية التيسينج.
  - تعريف منهجية التيسينج بشكل كلى، ده يشمل تعريف ال entry and exit levels وال test levels .criteria
  - دمج ال testing activities والتنسيق بينهم فى ال software life cycle activities .(acquisition, supply, development, operation and maintenance)
  - اتخاذ القرارات بخصوص ايه اللي هيتعمله تيست وال roles اللي هتعمل ال test activities وإزاي ال test results هتعمل وإزاي هتقيم ال test analysis and design activities عمل جداول لل evaluation وال execution وال test implementation .
  - تحصيص ال resources لل activities المختلفة اللي عرفناها قبل كده.
  - تعريف ال amount ومستوى التفاصيل وال structure وال templates لل test documentation .
  - اختيار المقاييس metrics للمراقبة والتحكم فى إعداد التيست وتنفيذ وحل ال defect .risk issues
  - تحديد مستوى التفاصيل لل test procedures لتقديم معلومات كافية لدعم ال preparation and execution المتكرر.

### :Entry Criteria :5.2.3

الـ **entry criteria** يحدد إمتى نبدأ التيسينج زى مثلا بداية **test level** معين أو إمتى تكون مجموعة من الـ **tests** جاهزة للـ **execution**.  
الـ entry criteria ممكن تشمل:

- Test environment availability and readiness
- Test tool readiness in the test environment
- Testable code availability
- Test data availability

### :Exit Criteria :5.2.4

الـ **exit criteria** يعرف إمتى أوقف تيسينج زى مثلا فى نهاية **test level** معين أو إمتى مجموعة من الـ **tests** تحقق هدف معين.

الـ exit criteria ممكن تشمل:

- القياسات الدقيقة زى مثلا **coverage** على الكود أو **functionality** معينة أو **risk** معين.
- تقدير **defect density estimate** أو **reliability measures**.
- **cost**.
- الـ **risks** الباقيه زى مثلا **defects** ماتحلتش لسه أو ضعف الـ **test coverage** فى مناطق معينة.
- الـ **schedules** زى مثلا اللي بنعملها للـ **market** (أناحتاج أنزل السوق توير بتابعى ف وقت معين).

### :Test Estimation :5.2.5

فيه طريقتين لتقدير الـ test effort

- الـ **metrics-based approach**: وده بيقدر الـ **testing effort** بناء على الـ **metrics**.
  - اللي اتعلمت على مشاريع مشابهة أو سابقة أو **based** على نتائج فعلية.
  - الـ **expert-based approach**: وده بيقدر التاسكات بناء على التقديرات اللي حطها الـ **owner** بتاع التاسك دى أو الخبراء ف الليلة دى.
- وب مجرد ما بيقدر الـ **test effort** ده (هياخد وقت قد إيه وكده) نقدر بعد كده نحدد الـ **resources** ونعمل الـ **schedule** للكلام ده كله.

### ال factors شاملة: testing effort ممكن يعتمد على مجموعة من

- خصائص ال product: جودة ال specification وأى معلومات تانية بستخدمها ف ال test (زى ال requirements) وحجم ال product وتعقيد مجال المشكلة requirements security وال reliability المطلوبة لـ documentation الخاصة بال organization.
- خصائص عملية ال development: زى ال organization tools stability وال المستخدمة وال test process ومهارات الناس المشتركة ف الليلة دى وضغط الوقت.
- اللى طالع من التesting: عدد ال defects وكمية ال rework المطلوبة.

### :Test Strategy, Test Approach :5.2.6

ال project test strategy implementation هو test approach معين، ال approach بتعرف وبتشتت من ال test designs وال test plans، وكمان بتشتمل القرارات اللي اتعلمت بناء على الهدف من المشروع ده (فى التيست) وتقدير ال risk، وده نقطة البداية فى ال planning لعملية التيست ولا اختيار ال test types وال test design techniques المطلوبة وكذلك تعريف ال entry criteria وال exit criteria.

ال approach اللي بنختاره بيعتمد على المحتوى وممكن يأخذ ف اعتباره شوية حاجات زى ال risks وال safety وال regulations وال test objectives (COTS مثلا) وال built resources وال technology وال skills وال custom.

### ال approaches المثلية بتشتمل:

- زى ال risk-based testing: Analytical approaches على المناطق اللي فيها أعلى risk.
- زى ال stochastic testing: Model-based approaches إحصائية عن معدلات ال failure growth models (failure growth models) أو الاستخدام (operational profiles).
- زى ال failure-based: Methodical approaches شامل ال error guessing (failure-based) وال quality attacks وال checklist-based experience-based (fault attacks) وال characteristic-based.

## :Test Progress Monitoring and Control :5.3

## :Test Progress Monitoring :5.3.1

الغرض من الـ **test monitoring** هو تقديم الـ **feed back** والرؤية الخاصة بالـ **test activities** والمعلومات اللي هنراقبها دى ممكن تجتمع **manually** أو بشكل أوتوماتيك وممكن تستخدم لقياس الـ **exit criteria** زى الـ **coverage** مثلا، الـ **metrics** ممكن كمان تستخدم لنقييم الـ **progress** اللي تم مقارنة بالـ **planned schedule** والميزانية **budget**.

المعروفة **test metrics** تشمل:

- نسبة الشغل اللي تم فى إعداد الـ **test environment**.
  - (زى مثلا عدد الـ **test cases** اللي اتعملها **run** واللى لسه،  
والـ **passed** والـ **failed** **test cases**).
  - المكتشفة واللى  
الـ **defect information** (زى مثلا الـ **defect density** والـ **defects**).  
تم حلها ومعدل الـ **failure** ونتائج الـ **(re-test)**.
  - الـ **requirements** أو الـ **risks** أو الكود.  
مدى ثقة الـ **testers** ف الـ **product**.
  - تواريخ الـ **milestones**.
  - تكلفة التيسينج، وده شامل التكلفة مقارنة بالـ **benefit** من إنى ألاقي الـ **defect** الجاية أو إنى  
أعمل **next test run** للـ **test cases**.

## :Test Reporting :5.3.2

الـ **test reporting** بيهتم بتلخيص المعلومات عن عملية التesting.

## دہ پیشمنل:

- ايه اللى حصل أثناء فترة التيسينج، زى التواريخ اللى اتحققت فيها ال exit criteria  
 • تحليل المعلومات والمقاييس recommendations metrics لدعم ال decisions والقرارات بخصوص  
 • actions المستقبلية، زى تقييم ال defects الباقيه و الفايدة الاقتصادية من التيسينج المستمر  
 • وال tested software ومستوى الثقة فى ال outstanding risks.

'Standard for Software Test' موجود فی test summary report outline .Documentation' (IEEE Std 829- 1998)

الـ metrics لازم تجمع أشياء الـ test level وف آخره لتقيم:

- مدى كفاءة الـ **test objective** للـ **test level**.
  - كفاءة طرق التيسير **اللى** ماشيين **عليها**.
  - تأثير التيسير **الى** **نسبة** **للـ objectives**.

### **:Test Control :5.3.3**

الـ **test control** بيوصـف أى guiding or corrective actions بنـاخـداـها كـنتـيـجـة المـعـلـومـات والـ **test activity metrics** المسـتـنـجـة والـلى اـتـعـلـمـها report، الخطـوـات دـى مـمـكـن تـغـطـى أـى وـمـمـكـن تـأـثـر عـلـى أـى task أو software life cycle activity.

أمثلة على الـ **test control actions**

- صناعة القرارات بناء على المعلومات اللي جاية من ال test monitoring .
  - عمل identified risk tests re-prioritizing لما يحصل أى (زى التأخير فى ال software delivery )
  - التغيير فى ال availability بسبب ال unavailability أو ال test schedule لـ test environment .
  - تحديد ال entry criterion اللي تحتاج fix عشان يتعمل عليه re-test .
  - عن طريق الـ confirmation test ( build ) قبل ما قبله فى

## :Configuration Management :5.4

### :Background

الغرض من ال configuration management هو إنشاء والحفظ على ال integrity فى documentation (البيانات والبيانات) components (البرمجيات أو الأجهزة) products خلال project and product life cycle.

بالنسبة للتيسينج، ال configuration management ممكن تشمل التأكيد من الحاجات دى:

- كل ال items مترقبة testware معرفة وال development items (test objects) changes مرتبطة ببعضها ومرتبطة بال عشان نحافظ على traceability خلال عملية التيسينج.
- كل ال referenced software items identified documents بشكل واضح في test documentation.

بالنسبة للتيسينج، ال configuration management بيساعد لتعريف بشكل موحد (وإعادة إنتاج) .test harness(es) وال tests وال test documents tested item أثناء ال configuration management procedures and test planning .implemented documented infrastructure (tools) لازم يتم اختيارها وتكون.

## :Risk and Testing :5.5

### :Background

ال risk ببitem تعريفه على على إنه فرصة حدوث event أو خطر أو تهديد أو موقف معين ويبينج عنه عواقب غير مرغوب فيها أو مشكلة محتملة، ومستوى ال risk بيتحدد باحتمال حدوث event عكسى وال impact (مدى الضرر اللي نتج عن ال event).

### :Project Risks :5.5.1

الـ **project risks** هى الـ **risks** المحيطة بقدرة الـ **project objectives** على تحقيق الـ **project** بذاته،  
زى مثلا:

#### **:Organizational factors •**

- نقص المهارات والـ **staff training** والـ **problems**.
- مشاكل شخصية.
- مشاكل Political issues زى مثلا:
  - مشاكل مع التيسيرز فى توصيل احتياجاتهم والـ **test results**.
  - فشل التيم كله فى متابعة المعلومات الموجودة فى التيسينج والـ **reviews** (زى مثلا عدم تحسين الـ **development and testing practices**).
  - تصرف غير لائق تجاه التيسينج والتوقعات الغير لائقة بالتسينج (زى مثلا عدم تقدير قيمة اكتشاف الـ **defects** أثناء التيسينج).

#### **:Technical issues •**

- مشاكل فى تعريف الـ **requirements** الصحيحة.
- المدى اللي فيه الـ **requirements** مش هنقدر تحقق الـ **constraints** الحالية اللي عندنا.
- Test environment مش جاهزة فى الوقت المحدد.
- التأخير فى الـ **migration planning and data conversion** والـ **data tools** الخاصة بعمل التيسىت على الـ **development conversion/migration**.
- قلة الجودة فى الـ **designs** والـ **code** والـ **data configuration** والـ **tests**.

#### **:Supplier issues •**

- Failure of a third party
- Contractual issues

أثناء ما بنعمل **test manager** وـ **analyzing** وـ **mitigating** **risks** دى، الـ **outline well-established project management principles** بتاتع الـ 'Standard for Software Test Documentation' (IEEE Std 829- 1998) للـ **contingencies** **test plans** بيذكر الـ **risks** والـ  **الطوارئ**

### :Product Risks :5.5.2

يقولك إن **product risks** هي عبارة عن المناطق التي محتمل حدوث **failure** فيها (**events**) أو مخاطر مستقبلية عكسية) في السوق أو السيستم، وإنها تعتبر **risk** لجودة الـ **product** فممكن تشمل:

- الـ **software delivered** المعرضة للفشل.
- احتمال إن السوق/الهاردوير ممكن يسبب ضرر لشخص أو شركة.
- الخصائص الضعيفة للسوق (على مثلا الـ **functionality** والـ **reliability** والـ **usability**).
- والـ **performance**.
- الـ **data integrity and quality** الضعيفة (على مثلا مشاكل الـ **data migration** - مشاكل نقل البيانات - انتهاك معيير البيانات **violation of data conversion standards**).
- السوق الذي لم يحصل على المعدة قبل كده **functions**.

الـ **risks** بتستخدم لتقرير نبدأ فين التيسينج وننتهي أكثر فين؛ والتيسينج بيسخدم في تقليل **risk** الذي بيحصل، أو لتقليل تأثير الـ **effect** العكسي.

الـ **product risks** هو نوع خاص من مخاطر نجاح المشروع، فالتيسينج كـ **risk-control** يقدم الـ **feedback activity** عن المخاطر الباقيه عن طريق قياس تأثير إزالة الـ **critical defect** وكمان خطط الطوارئ **contingency plans**.

الـ **risk-based approach** في التيسينج يقدم فرص استباقية لتقليل مستويات الـ **product risk** وببدأ في المراحل الأولية للـ **project**، ده بيشمل تعريف الـ **product risks** واستخداماتها في إرشاد الـ **test planning and control** وإعداد وتنفيذ الـ **tests** وإعداد **specification**.

في الـ **risk-based approach** ممكن تستخدم الحاجات دي:

- تحديد الـ **test techniques** الذي هنسخدمها.
- تحديد مدى التيسينج الذي هيتفذ.
- عمل **prioritize** للتيسينج في محاولة لإيجاد الـ **critical defects** بدري على قد مانقدر.
- تحديد أي **non-testing activities** المفروض تتعمل عشان نقل الـ **risk** (على مثلا عمل تدريب للـ **inexperienced designers**).

الـ **risk-based testing** يعتمد على الـ **collective knowledge** ورؤيه الـ **stakeholders** لتحديد الـ **risks** ومستويات التيسينج المطلوبه لمعالجة الـ **risks** دي.

عشان نتأكد إن فرصة ال product failure فى أقل مستوى، ال risk management activities بتقدم طريقة منهجية عشان:

- تقييم (وإعادة تقييم ال basis العادلة) إيه اللي ممكن يكون غلط (risks).
- تحديد ال risks المهمة اللي نتعامل معها.
- تنفيذ actions معينة للتعامل مع ال risks.

بالإضافة لـكده، التيسينج ممكن يدعم التعرف على risks جديدة، وممكن يساعد فى تحديد ال risks اللي لازم نظلها، وممكن يقلل الشك فى risks معينة.

## :Incident Management :5.6

### :Background

بيقولك إن من ضمن أهداف التيسينج هو إيجاد ال defects، فالفارق بين النتائج الفعلية والمتوقعة بنحتاج نسجلها على أنها incidents، ال incident لازم نركز معاه لأنه ممكن يتحول لـ defect، ولازم incidents نعرف خطوات مناسبة عشان نخلص على ال defects وال incidents، ولازم incidents defects دى يتسجلوا من أول اكتشافهم وتصنيفهم لغاية تصحيحهم وتأكيد حلهم، وعشان ندير كل incidents لغاية ماتكمل لازم الشركة تعمل an incident management process وقواعد incidents لتصنيف incidents دى.

ال incidents ممكن تتربع أثناء ال development أو ال review أو التيسينج أو أثناء استخدام software product، كمان ممكن تزيد وتكون issues فى الكود أو السيستم اللي شغال أو أى نوع من ال documentation requirements وال development documents وال test documents ومعلومات اليوزر زى ال "Help" مثلاً أو ال installation guides.

### ال goals لـ incident reports :

- بيمد الديفوبرز وال feedback بال other parties عن المشكلة عشان نقدر نتعرف عليها ونعزلها ونصححها لو ضروري.
- بيمد ال test leaders بوسائل تسجيل جودة السيستم اللي بيتعمل عليه التيسير والمجهود المبذول فى عملية التيسينج.
- بيقدم أفكار لتحسين عملية التيسير.

تفاصيل الـ **incident report** ممكن تشمل:

- تاريخ الـ issue organization والـ author اللي عاملها ريبورت.
  - الـ actual results والـ expected results.
  - Identification of the test item (configuration item) and .environment
  - الـ software or system life cycle process اللي حصلت فيها الـ incident دى.
  - الـ logs بتاع الـ incident description عشان نقدر نعملها تانى ونحلها، ده بيشمل الـ screenshots أو الـ database dumps.
  - الـ stakeholders أو درجة التأثير على اهتمامات الـ stakeholders.
  - الـ severity (درجة التأثير على السيستم).
  - الـ priority (أولوية حلها).
  - حالة الـ incident (open, deferred, duplicate, waiting to be fixed, fixed) .(awaiting re-test, closed)
  - Conclusions, recommendations and approvals
  - الـ Global issues زى المناطق الثانية اللي ممكن تتأثر بتغيير النتيجة بسبب الـ incident.
  - الـ project team members زى مثلا تسلسل الخطوات اللي عملها الـ project team members.
  - بخصوص الـ incident دى لعزلها وتصلحها والتأكد إنها اتحلت.
  - References الـ test case specification ودى بتشمل التعرف على الـ References اللي أظهرت المشكلة.
- الـ 'Standard for incident report' كمان موجودة فى الـ 'Software Test Documentation' (IEEE Std 829-1998) هنجي للنقطة دى أكثر إن شاء الله في 2 part.

## Chapter 6

### Tool Support for Testing

#### :Types of Test Tools :6.1

#### :Tool Support for Testing :6.1.1

بيكولك إن ال tools ممكن تستخدم فى activity أو أكثر من testing بتدعم ال tools دوبل بيشملوا:

1. ال tools اللي بتستخدم مباشرة في التيسينج زى ال test execution tools وال result comparison tools وال data generation tools
2. ال tools اللي بتساعد فى إدارة عملية التيسينج زى المستخدمة لإدارة ال tests وال test tools وال الداتا وال incidents وال requirements وال defects وكمان لمراقبة incidents reporting عمل test execution عليها.
3. ال tools المستخدمة للاكتشاف (زى مثلا ال tools اللي بتراقب ال file activity application معين).
4. أى tool بتساعد فى التيسينج (ال spreadsheet تعتبر كمان test tool ف الحالة دي).

#### دعم ال tool للتيسينج ممكن يبقى ليه أكثر من غرض حسب المحتوى زى:

- تحسين كفاءة ال repetitive tasks عن طريق automating test activities
- دعم ال test design test planning manual test activities .reporting and monitoring
- عمل manually resources activities لل automation كبيرة لما بتعمل (زى ال static testing)
- عمل manually activities لل automation large scale (زى ال client-server applications) performance testing
- زيادة الثقة فى التيسينج (عن طريق automation data لمقارنات ال large data .simulating behavior)

### المصطلح "test frameworks" بستخدم كمان بشكل متكرر ف الصناعة في 3 معانى ع الأقل:

- الـ **testing libraries** القابلة لإعادة الاستخدام والتتوسع اللي ممكن نستخدمها عشان نعمل **test tools building** (اسمها كمان الـ **test harnesses**).
- نوع من الـ **diyazain** لـ **test automation** (زى مثلا **keyword-data-driven** أو **driven**).
- عملية الـ **execution of testing** كلها بشكل عام.

وعشان هدف المنهج فمصطلاح الـ "test frameworks" بستخدم فى أول معنيين ليه زى ما هيوضح أكثر في النقطة 6.1.6.

### :Test Tool Classification :6.1.2

فيه عدد من الـ **tools** اللي بتدعم أوجه مختلفة من التيسينج، والـ **tools** ممكن تتصنف بناء على معايير كتير زى الـ **commercial / free / open-source / shareware - purpose** - **testing activities** المستخدمه وهكذا، الـ **tools** مصنفة ف المنهج ده طبقا لـ **technology** اللي بتدعمها.

بعض الـ **tools** بتوضح **activity** واحدة، وفيه تانيين ممكن يدعموا أكثر من **one activity**، لكن بتصنف تحت الـ **activity** اللي هما مرتبطين بي بعض أكثر، الـ **tools** اللي بتكون من **single provider package** - خاصة اللي بتكون مصممة عشان تشتعل مع بعض - ممكن تندمج في واحد.

بعض الأنواع من الـ **test tools** ممكن تبقى طفلية **intrusive**، وده معناه إنهم ممكن يؤثروا على النتيجة الفعلية للتيسينج، يعني مثلا الـ **actual timing** ممكن يختلف بسبب الـ **extra instructions** اللي اتفقدت عن طريق الـ **tool**، أو ممكن تديينا قياس مختلف للـ **code coverage**، النتيجة بتاعة الـ **intrusive tools** دى بنسميها الـ **.probe effect**.

بعض الـ **tools** بتعرض دعم مناسب أكثر للديفلوبرز (زى مثلا الـ **component integration testing** والـ **component testing**)، زى الـ **tools** اللي جنب اسمها (D) في الأنواع اللي هتتجى بعد كده في كلامنا.

### :Tool Support for Management of Testing and Tests :6.1.3

ال software life مطلوبة في كل ال test activities على مدار ال management tools life cycle كلها.

#### **:Test Management Tools**

ال tools دى بتقدم interfaces لتنفيذ ال tests ومتابعة ال defects وإدارة ال requirements ده جنبا إلى جنب مع دعم التحليل الكمي quantitative analysis وعمل reporting لل requirement specifications لل test objects، كمان ال tools دى بتدعم تتبع ال objects وممكن يبقى ليها إمكانية مستقلة للتحكم في ال interface أو ال version لوحدة خارجية.

#### **:Requirements Management Tools**

ال tools دى بتخزن ال requirements statements وخصائص ال requirements (شاملة individual requirements) وبتقديم unique identifiers (priority) وبتقديم tools دى ممكن كمان تساعد في التعرف على ال requirements الناقصة أو الغير متنسقة مع بعضها.

#### **:Incident Management Tools (Defect Tracking Tools)**

ال tools دى بتخزن وبتدير ال faiures أو ال defects أو incidents أو change requests أو المشاكل الملمسة، وكمان بتساعد في إدارة ال life cycle باتباع statistical analysis.

#### **:Configuration Management Tools**

بالرغم من إن ال configuration management tools مش test tools دقيقة، إلا إنها ضرورية لتخزين وعمل testware لل version management والسوفتوير اللي عليه الكلام خاصة operating system من حيث ال hardware/software environment أكتر من ...browsers – compilers – system versions ... الخ.

### :Tool Support for Static Testing :6.1.4

ال Static testing tools بتقدم طريقة فعالة ماديا لإيجاد defects أكتر في مرحلة بدري في عملية development.

## Review Tools

ال tools دى بتساعد مع عمليات reviews وال checklists وال review guidelines و بتستخدم لتخزين ولتوصيل ال defects عن ال report review comments و عمل effort وال defects عن ال report التي يتم، كمان ممكن يساعدوا بشكل أكبر عن طريق تقديم المساعدة لل teams online reviews لـ teams الكبيرة أو المشتقة بشكل جغرافي (زى مثلا team مكون من حد م مصر على حد م الهند على حد من امريكا... كده يعني).

## Static Analysis Tools (D) (6.1.2)

ال tools دى بتساعد الديفلوبرز والنيسترز إنهم يلاقوا defects قبل dynamic testing عن طريق تقديم الدعم لفرض ال coding standards (شاملة ال secure coding standards) وتحليل risk analysis أو ال dependencies planning أو ال structures عن طريق تقديم مقاييس metrics لل코드 (زى ال complexity مثل).

## Modeling Tools (D)

ال tools دى بتسخدم عشان تتأكد من صحة ال software models (زى مثلا ال data models) عن طريق تسجيل التناقضات وإيجاد ال defects . ال model tools دى تقدر عادة تساعد فى عمل بعض ال test cases بناء على ال model.

## Tool Support for Test Specification :6.1.5

### Test Design Tools

ال tools دى بتسخدم لإنتاج ال executable tests أو test inputs وأو توقعات التيسىت design models من ال graphical user interfaces أو ال requirements oracles أو ال (state, data, or object) الكود.

### Test Data Preparation Tools

ال tools دى بتعالج ال databases أو ال files تتعلق البيانات عشان تحضر ال test data لاستخدامها أثناء تنفيذ ال tests للتأكد من ال security فى حالة عدم التعرف على البيانات.

## :Tool Support for Test Execution and Logging :6.1.6

### **:Test Execution Tools**

ال tools دى بتخلی ال tests تتنفذ semi-automatically أو automatically عن طريق استخدام inputs متخرنة ونتائج متوقعة، وده من خلال استخدام scripting language وعادة بيقدم tools دى ممكن كمان تستخدم فى تسجيل ال test log لكل test run ، وال tools دى ممكن عشان ال parameterization GUI-based configuration أو scripting languages عشان ال customization of data وال tests الثانية فى ال .tests

### **:Test Harness/Unit Framework Tools (D)**

ال system عن طريق محاكاة ال environment الذى فيها ال test object——run ، خلال تقديم components بيسهل عملية التesting لل unit test harness or framework السيسystem المقلدة دى ك drivers أو stubs objects (هنعرفهم أكثر فى 2 part).

### **:Test Comparators**

ال test comparators بيحدد الفروق مابين ال databases أو ال files أو ال test results post-execution tools dynamic comparators ، لكن ال test execution tools ممكن تتعمل ب comparison tool منفصلة ، ال comparison tool ممكن يستخدم توقعات التesting automated test oracle خصوصاً لو .automated

### **:Coverage Measurement Tools (D)**

ال tools دى – سواء بوسائلها ال non-intrusive أو ال intrusive – بتقيس النسبة المئوية لأنواع معينة من ال statements exercise (زى مثلا code structures) عن طريق مجموعة من module or function calls وال branches or decisions .tests

### **:Security Testing Tools**

ال tools دى بتسخدم لتقدير قدرة security characteristics ف السوفتوير ، ده بيشمل تقدير data confidentiality, integrity, authentication ، حماية ال authorization, availability, and non-repudiation غالباً بتركز على particular technology, platform, and purpose .

### :Tool Support for Performance and Monitoring :6.1.7

#### **:Dynamic Analysis Tools (D)**

ال dynamic analysis tools اللى بتلاقي defects لما السوفتوير نعمله execution time dependencies أو ال memory leaks tools ، وال component integration testing ولما بنـتـيـسـتـ ال component testing middleware.

#### **:Performance Testing/Load Testing/Stress Testing Tools**

ال performance testing tools على إزاي السيسـتمـ بيـتـصـرـفـ تحتـمحاـكـاهـ ظـرـوـفـ الاـسـتـخـدـامـ منـ حيثـ عـدـدـ الـ c~oncurrent~ u~s~e~r~sـ معـ بـعـضـ فـ نـفـسـ الـ w~o~r~c~t~ وـ الـ r~a~m~p~ pattern up pattern بـتـاعـهـمـ والـ n~o~t~o~r~yـ والـ f~r~e~q~u~e~n~c~yـ والـ t~r~a~n~s~a~c~t~o~r~sـ المرـتـبـطـةـ بـبعـضـهـاـ ،ـ مـحاـكـاهـ الـ l~o~a~d~ بـيـتـمـ بـوـسـائـلـ عـلـمـ u~s~e~r~sـ وـ هـمـيـنـ بـيـنـذـنـوـاـ مـجـمـوعـةـ مـعـيـنـةـ مـنـ الـ t~r~a~n~s~a~c~t~o~r~sـ ،ـ الـ اـنـتـشـارـ خـلـالـ الـ l~o~a~d~ g~e~n~e~r~a~t~o~r~sـ الـ k~e~n~t~i~e~r~ ع~اد~ه~ بـيـع~ر~ف~ بـاسـم~ t~e~s~t~ m~a~c~h~i~n~e~s~ .ـ

#### **:Monitoring Tools**

ال monitoring tools بـتـحـلـ وـبـتـ verifyـ وـبـتـ systemـ بـشـكـلـ مـسـتـمـرـ عـلـىـ اـسـتـخـدـامـ الـ r~e~p~o~r~t~ .ـ مـعـيـنـةـ وـبـتـديـ لـمشـاكـلـ الـ s~e~r~v~i~c~e~sـ resourcesـ وـبـتـديـ لـمشـاكـلـ الـ w~a~r~n~i~n~g~sـ .ـ

### :Tool Support for Specific Testing Needs :6.1.8

#### **:Data Quality Assessment**

الـ d~a~t~a~ هـيـ الـ c~e~n~t~e~r~ لـبعـضـ الـ p~r~o~j~e~c~t~sـ زـىـ الـ a~p~p~l~i~c~a~t~o~n~sـ والـ d~a~t~a~ w~a~r~e~h~o~u~s~e~sـ زـىـ الـ a~t~t~r~i~b~u~t~e~sـ بـتـاعـتـهـاـ مـمـكـنـ يـخـتـلـفـواـ مـنـ نـاحـيـةـ الـ v~o~l~u~m~e~ وـ الـ c~r~i~t~i~c~a~l~i~t~y~ وـ الـ c~o~n~t~e~x~t~s~ ،ـ فـىـ بـعـضـ الـ t~o~o~l~s~ بـنـقـىـ مـحـاجـيـنـ تـسـتـخـدـمـهـاـ لـنـقـيـيـمـ جـوـدـةـ الـ b~i~o~d~a~t~a~ لـمـراـجـعـةـ وـالتـأـكـدـ مـنـ الـ d~a~t~a~ c~o~n~v~e~r~s~a~t~i~o~n~ وـ قـوـادـ الـ m~i~g~r~a~t~i~o~n~ لـلـتـأـكـدـ إـنـ الـ p~r~o~c~e~s~e~d~ d~a~t~a~ صـحـيـحةـ وـكـامـلـةـ وـبـتـكـامـلـ مـعـ الـ p~r~e~d~e~f~i~n~e~d~ c~o~n~t~e~x~t~s~p~e~c~i~f~i~c~a~l~ s~t~a~r~t~a~r~d~ .ـ

فـيـهـ t~e~s~t~i~n~g~ t~o~o~l~s~ تـانـيـةـ لـلـ u~s~a~b~i~l~i~t~y~ .ـ

## Effective Use of Tools: Potential Benefits and Risks :6.2

### Potential Benefits and Risks of Tool Support for Testing (for all tools) :6.2.1

بيقولك إن استخدام أو تأجير tool معينة مش ضمان للنجاح مع الـ tool دى، كل نوع من الـ tool ممكن يتطلب جهد إضافي لتحقيق الفائدة بشكل حقيق و دائم، أه فيه منافع و فرص محتملة لما استخدم الـ tool ف التيسينج، لكن كمان فيه risks.

#### المنافع المحتملة لاستخدام الـ tool تشمل:

- تقليل الشغل المتكرر (زى مثلا running regression tests وإعادة إدخال نفس ال test .(checking against coding standards data
- تناسق وتكرار أكبر (زى مثلا tests الذى اتنفذت بالـ tool بنفس الترتيب وبنفس requirements frequency والـ tests المستمدة من الـ requirements (coverage – static measures
- تقدير الهدف (زى مثلا tests أو التيسينج (زى مثلا الإحصائيات والرسومات البيانية سهولة الوصول للمعلومات عن الـ tests عن الـ test progress ومعدلات الـ incident .(performance

#### أما مخاطر استخدام الـ tools بتشمل:

- التوقعات المبالغ فيها لـ tool (شامل الـ functionality وسهولة الاستخدام).
- التقليل من الوقت والتكلفة والجهود للتقديم المبدئى لـ tool (شامل التدريب والخبرات الخارجية).
- التقليل من الوقت والجهود المطلوبين لتحقيق منافع كبيرة ومستمرة من الـ tool (شامل الاحتياج للتغييرات فى عملية التيسينج والتحسين المستمر للطريقة الذى بتستخدم بيهـا الـ tool).
- الاعتماد الزايد على الـ tools (استبدال للـ test design أو استخدام الـ automated tools فى حاجة هىكون الـ manual testing فيها أفضل).
- إهمال الـ tools فى أصول التيسـت خلال الـ tool.
- إهمال العلاقات ومشاكل العمل المشتركة بين الـ critical tools requirements، زى الـ incident management tools والـ version control tools والـ management tools tools والـ vendor tools tools .defect tracking tools والـ vendor tools والـ vendor tools
- بمنسبة المورد، فيه risk بخصوص الـ tool vendor إنه يصفى شغله تماماً أو يلغى العمل بالـ tool أو يبيع الـ tool different vendor .defect fixes والـ upgrades فى الدعم والـ vendor .open-source / free tool Risk
- وقف العمل فى مشروع الـ platform الجديدة مثلـ .الاحتاجات الغير متوقعة، زى عدم القدرة على دعم

## :Special Considerations for some Types of Tools :6.2.2

### :Test Execution Tools

النوع **test scripts** باستخدام **test objects** execute **test execution tools** بـ، ده من ال **tool** عادة بـيطلب مجهود كبير للوصول لـ **benefits** كبيرة.

عمل **tests** capturing عن طريق تسجيل ال **actions** اللي بـيعملها ال **manual tester** يبيان انها حاجة جذابة، لكن الطريقة دى مش مقاييس لعدد كبير من ال **automated test scripts**، ال **captured script** هو **linear representation** مع بيانات و **actions** معينين كجزء من كل **script**، النوع ده من ال **script** ممكن ما ييقاش مستقر لما تحصل أى **events** غير متوقعة.

طريقة ال **test inputs (the data)** بـتفصـل بين ال **data-driven testing** عادة فى **spreadsheet input** (زى ملف إكسيل مثلاً)، وبـتستخدم **test script** عام بحيث **يقدر يقرأ ال data** نفس ال **test script** بـبيانات مختلفة، التيسـترز اللي مش **familiar** مع **predefined test data** ممكن ساعتها يعملوا **create** لـ **test data** لل **scripting language** دـى **scripts**.

فيـه **data-driven techniques** تانية بـتـخدم فى الـ **hard-techniques**، فـبدل ما أعمل **generate** وـأحـطـهم فى **spreadsheet coded data combinations** بنـاء على **configurable parameters** فى الـ **run time algorithms** يعني مثلـا ال **tool** مـمـكن تـسـتـخدم الـ **algorithm** بيـ **generate** بـشكل عـشوـائـى **user ID** وـعـشـان التـكرـار فى الـ **pattern** فيه أساس مـحـطـوط للـ  **الحكم** فى الـ **randomness**.

فى الـ **keywords spreadsheet**، الـ **keyword-driven testing approach** بـتحـتـوى على **actions** اللي بتـتـاخـد (معروفة كـمان بـاسم **action words**) وبـتعـمل **test** للـ **بيانـات**، التـيسـترـز حتى لو مش **familiar** مع الـ **scripting language** يـقدرـوا ساعـتها يـعـرفـوا الـ **tests** باـستـخدـام الـ **keywords**، اللي مـصـمـمة مـخـصـوص لـ **الأـبـلـيـكـيشـن** اللي بـيـتـعـملـ علىـهـ التـيسـتـ.

الـ  **الخبرـةـ التـكـنـيـكـالـ** **technical** فى الـ **scripting language** مـطلـوبـةـ لـ **كلـ الـ طـرـقـ** (سواء من التـيسـترـز أو من المـتـخـصـصـينـ فـ الـ **test automation**).

وـبعـضـ النـظـرـ عنـ الـ  **المستـخـدمـ**، فالـ **expected results** لـ **كلـ تـيسـتـ** مـحتاجـةـ تـخـزنـ لـ **المـقارـنةـ** اللي هـتـمـ بعدـ كـدهـ (معـ الـ **actual result**).

### :Static Analysis Tools

ال static analysis tools مطلوبة source code ممكن نمشيه على coding standards لكن لو applied على كود موجود بالفعل ممكن يطلع كميات كبيرة من messages warning، ال executable program messages مابتوقف الكود عن ترجمته implementation maintenance عشان ال maintenance لل코드 تبقى أسهل ف المستقبل، ال التدريجي initial filters analysis tool للاستخراج بعض ال messages يعتبر طريقة فعالة.

### :Test Management Tools

ال test management tools بتحتاج تتفاعل مع spreadsheets أو tools تانيين عشان تنتج معلومات مفيدة بشكل يناسب احتياجات organization.

## :Introducing a Tool into an Organization :6.3

### Background

#### الاعتبارات الرئيسية في اختيار tool لل organization بتشمل:

- تقييم ال organizational maturity ونقط القوة والضعف وتعريف الفرص المتاحة improved test process ل tools اندعمت بال .
- التقييم بناء على ال requirements الواضحة وال objective criteria .
- ال proof-of-concept عن طريق استخدام test tool أثناء مرحلة التقييم لتحديد ما إذا كانت بتؤدي بفاعلية مع السوفتوير اللي بيتعمل عليه التيسير وخلال ال infrastructure الحالية أو لتحديد التغييرات المطلوبة لل infrastructure . tool دى لاستخدام فعال لل .
- تقييم ال vendor (شامل التدريب والدعم والنوافذ التجارية) أو ال service support . non-commercial tools فى حالة ال suppliers .
- تعريف المتطلبات الداخلية للتدريب على ومراقبة استخدام ال tool .
- تقييم التدريب المطلوب بناء على مهارات ال team الحالية فى ال test automation .
- تقدير معدل ال cost-benefit بناء على business case ملموسة .

### عشان تقدم ال tool المختارة فى ال organization لازم تبدأ بمشروع تجريبي أهدافه:

- تعلم تفاصيل أكثر عن ال tool .
- تقييم مدى مناسبة ال tool لل practices وال processes الحالية وتحديد إيه اللي ه يحتاج يتغير.
- تقرير الطرق ال standard فى استعمال وإدارة وتخزين وصيانة ال tool وأصول التيسىست (زى مثلا اتخاذ قرار بخصوص طرق التسمية naming conventions لل files وال testing .).
- عمل ال libraries وتعريف شكل ال test suites .
- تقييم ما إذا كانت ال benefits هتحقق بنكلفة معقولة ولا لا.

### أما عوامل النجاح عشان أعمل tool فى ال organization لل deployment بتشمل:

- طرح ال tool فى باقى ال organization بشكل تدريجي.
- تكيف وتحسين ال processes عشان تناسب استخدام ال tool .
- تقديم التدريب / التوجيه لل new users .
- تعريف ال usage guidelines .
- عمل طريقة لاكتساب معلومات الاستخدام من الاستعمال الفعلى.
- مراقبة استخدام ال tool وال benefits .
- تقديم الدعم لل tool لل test team .
- استخلاص الدروس المستفادة من كل ال teams .

وبكده نكون خلصنا مع بعض شرح منهج ISTQB الرسمى، فى حالة لو عايز تتحدى فيه 4 امتحانات موجودين فى Part 3 ف آخر الكتاب من ضمنهم الامتحان الرسمى اللي نازل على موقع ISTQB.

الامتحان عبارة عن 40 سؤال بيتحلو فى 60 دقيقة، طبعاً لازم تشوف نماذج كتير من الامتحانات قبل ماتدخل الامتحان الحقيقي لإنه مش بالبساطة اللي انت متخيلاها دى ...

أما عن مكان الحجز للامتحان، فلو انت من مصر تقدر تحجز الامتحان من SECC... دى هتلقيها فى القرية الذكية فى مبنى ITIDA، وده نفس المكان اللي هتستم منه الشهادة لما تنجح إن شاء الله... فربنا معاك.

دلوقتى خلصنا Part 1 من الكتاب... تعالوا بقى للشغل العملى... تعالوا Part 2.

Part 2  
**Testing Practices**

## Day 1

### :Software Development models

#### Background

من الشائعات المنتشرة أوى عن مجال التيسينج إن التيسينج ببجي بعد مايتعمل الكود، فهنا هنتكلم أكثر عن النقطة دى، ونشوف إيه حكاية التيسينج وموقعه من الإعراب ف الليلة دى.

بس ياسيدى... عملية ال development ف أى شركة برمجة بتتم طبقا لحاجة اسمها **Development Life Cycle (SDLC)**، ودى عباره عن الخطوات أو المراحل اللي بيمر فيها أى برنامج من الصفر لغاية بعد ما يتسلم للعميل، المراحل دى هي:

- ف المرحلة دى بتتجمع ال requirements من العميل.
- بنحو ال requirements دى ونهيئها عشان تتعمل بالكود (زى ال wireframes ) وال flowcharts وكده يعني).
- بنعمل كود البرنامج اللي بيحق الكلام ده.
- بنتيست ع الكود اللي معمول ده ونطلع منه ال bugs.
- بعد ما نسلم البرنامج للعميل لو ظهرت حاجة بنصلحها.

المراحل دى فى ال SDLC بتتنفذ بعدها **model** مختلفين عن بعض فى طريقة التفكير والتنفيذ، لكن كلام بيدوروا حولين نفس المراحل، فتعالوا نشوف ال models دى.

#### :SDLC Models

**Sequential models -1**

**Iterative Models -2**

**Incremental Models -3**

**Agile Models -4**

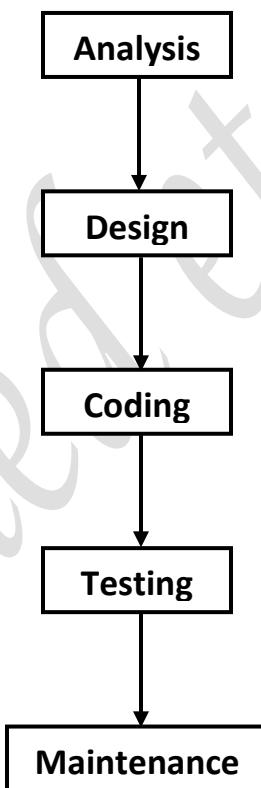
ودولوقتى هنتكلم عن كل واحد فيه بالتفصيل.

:Sequential models -1

ف الموديل ده المراحل بتاعة ال **SDLC** بتتم بشكل متسلسل ورا بعضه **sequential**، أشهر مثال ع الموديل ده ال **waterfall**.

**Waterfall model**

ال **waterfall model** ده الخطوات بتتنفذ ورا بعضها، يعني ف الأول بنعمل ال **Analysis** وبعد مايخلص نعمل ال **requirement** وبعد مايخلص ندخل ع الكود وبعد كده التيسست وف الآخر بنسلم الكلام ده للعميل، فيكون ماشى بال **sequence** ده:



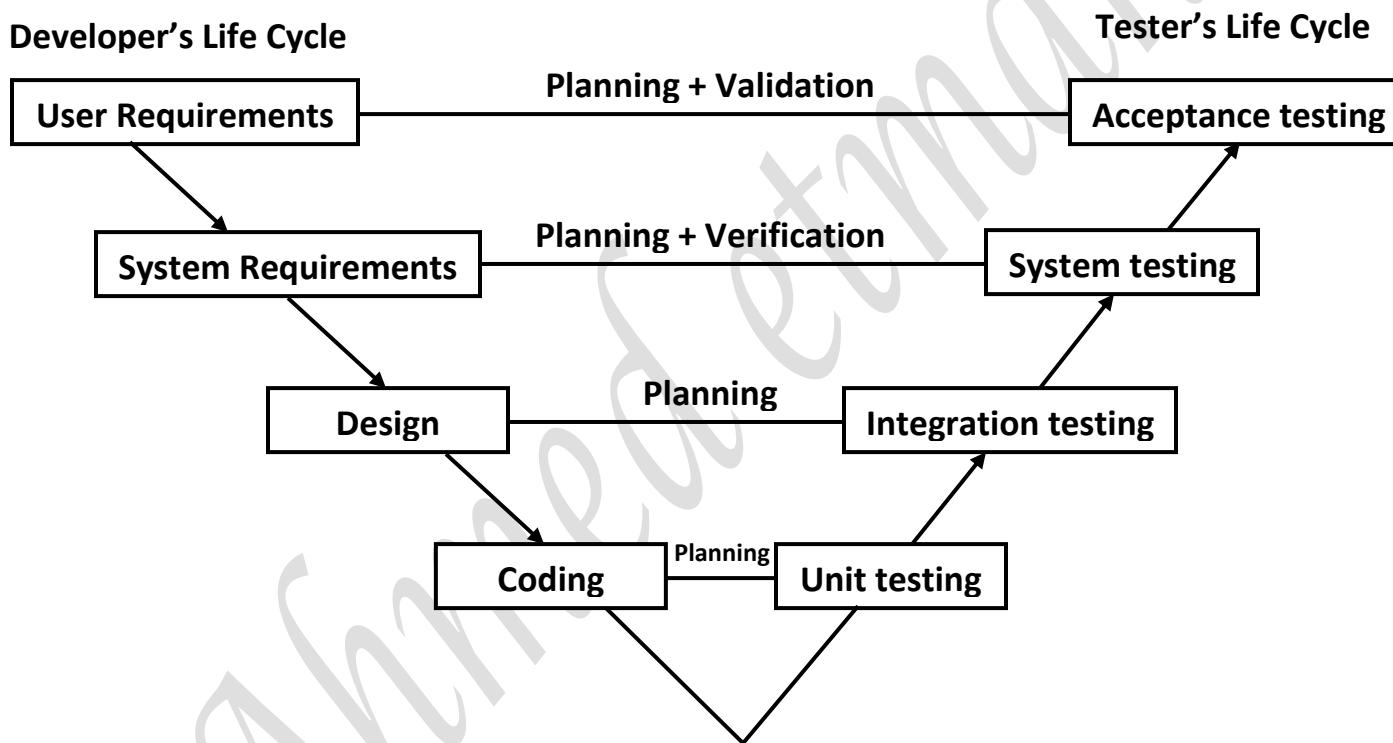
في مجرد بدأنا في مرحلة وخلصت بتدخل ع اللي بعدها علطول وماينفعش نرجع لها تانى (وده واضح من اسمها أنها عاملة زى شلال المياه ماشى من فوق تحت وماينفعش يرجع من تحت لفوق تانى)، وده عيب كبير ف الطريقة دى لأن لو طلعت **Bug** ف مرحلة التيسينج ناتجة عن فهم خاطئ لل **requirements** مثلًا بضطر أعيد كل المراحل دى من أول وجديد عشان أصلاح البجاية دى، وده بيكلف كتير زى ماقلنا قبل كده في part 1، ده غير الوقت الطويل جداً اللي بيتأخذ في مرحلة التيسينج وانا بعمل **planning** **execution** **unit testing** **integration testing** وال **system testing** لل **execution**

وال acceptance testing وفى الغالب مابييقاش الشركة عندها الوقت الكافى للتيسينج قبل معاد التسليم للعميل.

عشان كده عدوا الطريقة دى لطريقة أحسن اسمها V-model .

### :V-model

الطريقة دى عباره عن موديل بتمشى التيسينج بشكل متوازى إلى حد ما مع باقى المراحل الثانية على هيئة حرف ال V ، الشكل ده هيوضح الدنيا أكثر:



يعنى إيه بقى الكلام ده؟

الموديل باختصار قايمه فكرته على تقليل وقت التيسينج بحيث يتم عمل بشكل مظبوط عن طريق عمل developing أثناء تنفيذ كل مرحلة من مراحل the test planning

- يعني في مرحلة أخذ ال requirements عمل قصادها لل acceptance planning .  
حيث أتأكد إن السيستم حق اللي طلبه اليوزر (Validation).
- وفي مرحلة استلام ال system requirements بشكل مفصل أكثر بنعمل planning لل system testing وأحط السيناريوهات المختلفة اللي هتمشى ع السيستم كله.

- ولما ننزل بتفاصيل أكثر في السيسن وأعمل ديزاين الصفحات والwireframes بقى بتاعتها .integration testing للأكواد بعمل planning للflowcharts
  - وأثناء تنفيذ الكود بنعمل unit test planning للtest planning بحيث يبقى التيسن جاهز للتنفيذ علطول لما الـ components تجهز.
  - فلما تجهز الـ components المختلفة نبدأ بقى نعمل execution لكل مراحل التيسن اللي كنا مخططينها قبل كده، وبالتالي تكون اختصرنا الوقت بتاع الـ waterfall planning ف الـ requirements إعادة الشغل في حالة اكتشاف bug في التيسينج مرتبطة بالـ components
- طيب في الشكل اللي فوق فيه مصطلحين الـ Validation والـ Verification ... ايه حكايتهم دول؟ تعالوا نشوف ...

### :Validation & Verification

**Verification:** هو التأكيد إن الـ function شغالة مطبوط (طبقاً للـ requirements) ... يعني لما أعمل function تجمع رقمين أتأكد إنني لو دخلته رقمين هيجمعهم ويطلعلي النتيجة الصح بتاعتهم (يعنى لو دخلته 1 و 4 أتأكد إن الناتج بتاعتهم 5).

**Validation:** هو إجابة على السؤال (هل ده اللي عايزة اليوزر؟).

يعنى إيه؟

يعنى لو اليوزر طالب برنامج يجمع رقمين الـ validation وظيفتها إنها تتأكد إن البرنامج اتعمل بالمواصفات اللي عايزة اليوزر، يعني تتأكد إن البرنامج بيجمع رقمين مش بيضررهم ف بعض مثلاً. ممكن برضه نوصف الـ Validation والـ verification ف السؤالين دول:

**Verification:** "Are we building the product right?"

**Validation:** "Are we building the right product?"

الـ early test design بتساعدنا كمان في تقليل الـ faults اللي ممكن تحصل في المستقبل، لأنك ساعتها هتبتدى تسأل عن كل حاجة، ولو فيه حاجة مش واضحة أو ناقصة بتكميل وبالتالي بتقل الغلطات قدام.

الـ sequential models من ضمن عيوبهم كلهم إن اليوزر أو الـ stakeholder مش بيشفوف البرنامج إلا لما يخلاص في الآخر خالص، يعني لو المدة اللي بعمل فيها البرنامج 6 شهور أو سنة مش هقدر أوري حاجة للعميل إلا بعد ما كل الشغل يخلاص، وبالتالي لو طلت حاجة مش عايزة اليوزر أو عايزة غير حاجة هنضطر نعمل شغل كتير جداً عشان نحقق اللي هو عايزة...

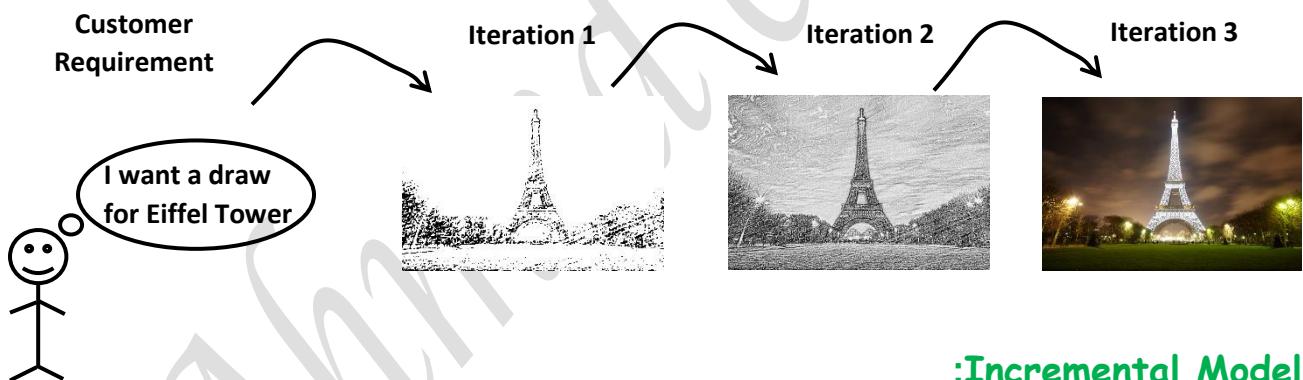
عشان كده ظهر الـ iterative model

**:Iterative Model**

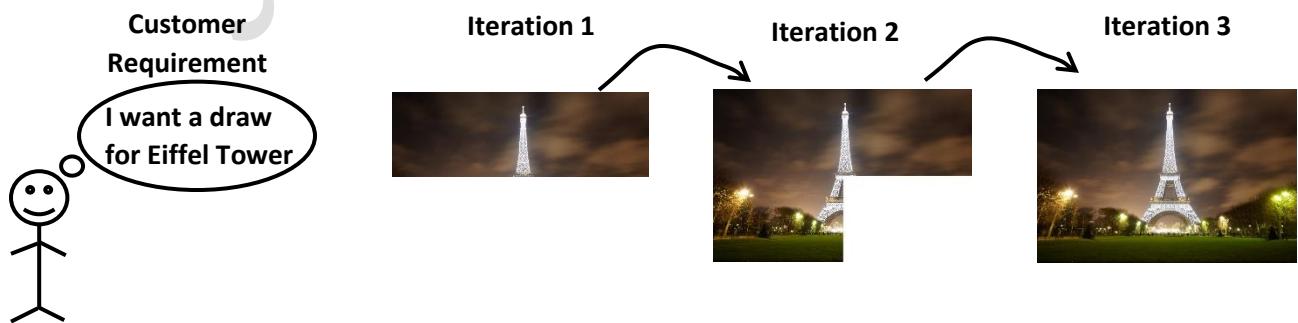
الـ model ده فكرته قائمة على إن البرنامح يتقسم لمجموعة **waterfall iterations**، فى كل iteration منه أطلع جزء غير مكتمل فى البرنامج وأعرضها على العميل ولو الدنيا تمام أبدأ أضيف عليها فى الـ iteration اللي بعدها وهكذا لغاية مايخلص البرنامج.

من ضمن مميزات الـ model ده إنى ممكن أبدأ الشغل حتى لو ما عنديش requirements كاملة، فبطبع الشغل على قد المعلومات اللي عندي لغاية ما نشوف رد فعل العميل ويديلنا باقى الـ requirements، كمان لو فيه حاجة محتاجة تتصلح أكيد هتاخذ وقت لتصليحها أقل من الـ waterfall للبرنامح كله.

**مثال:** لو قولنا اننا هنرسم رسمة لبرج إيفل، فطبقاً للـ Iterative model هناخد الـ requirements البدائية لشكل الصورة اللي العميل متخيلاها ونطلع أول iteration للصورة مثلاً مرسومة بشكل كروكي، بعد كده يبدأ يدي requirements وتعديلات بشكل مفصل أكثر فنبأ نرسمها بالتفصيل بالقلم الرصاص ونطلعها في iteration 2، ولما يشوفها العميل وتنقى الدنيا تمام نبدأ نضيف الألوان والتفاصيل النهائية ونطلع الصورة بشكلها النهائي في iteration 3 زى كده:

**:Incremental Model**

أما بقى الـ incremental Model فهو نفس الـ Iterative model بس الفرق إنى **يعمل جزء صغير من البرنامج لكن كامل** فى build وأعمل جزء تانى أضيفه ع الأولانى وهكذا زى كده مثلاً:



**:Agile Model**

الـ **waterfall model** والـ **Iterative model** كانوا عبارة عن **Incremental model** لكن على **scale** صغير وبيختلفوا في كيفية تطبيق الـ **waterfall process** دى، إنما الـ **Agile** يعتبر مختلفاً، حالياً الـ **SDLC model** هو الـ **agile** اللي بيتجه ليه أغلب الشركات في شغلها.

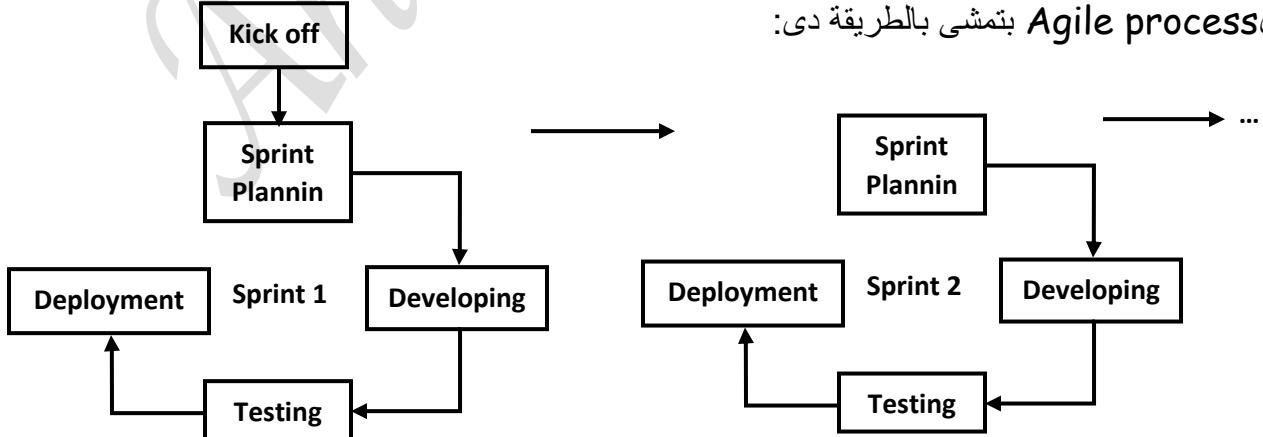
الـ **Agile** ده حوار كبير جداً لدرجة إن ISTQB نزلته منهجه وامتحان وشهادة لوحده، الـ **agile** ليه أكثر من **methodology** زي مثلاً الـ XP والـ TDD والـ Kanban والـ Scrum.

الـ **Agile model** هو زي الثانيين لكن الفرق إن ف الـ **agile** العملية بتكون من (الـ **Scrum master** والـ **Product Owner (PO)** والـ **(developers + testers)**) بين الـ **Scrum Master** والـ **PO** والـ **dev team** يكون بشكل مباشر وكمان **interaction** العامل بينهم وبين الـ **customer** يكون بشكل مباشر بدون وسيط، لدرجة إنه ف الـ **best practice** للـ **agile** إن الـ **customer** بييجي بنفسه أو بيعت مندوب عنه لمقر شركة البرمجة وبيقعد مع الـ **dev team** عشان لو فيه استفسار أو حاجة غلط تتصلح ف وقتها.

الـ **system** ف الـ **agile** بيقسم على كذا **sprint**، كل **sprint** بتبقى مدتها من إسبوعين لشهر، في خلال الـ **sprint** دى بيتم تحضير الـ **requirements** في **sprint planning meeting** (عادة ساعتين لكل إسبوع) وبعدين يبدأ الشغل.

في أول كل يوم الـ **dev team** والـ **Scrum Master** وأحياناً الـ **PO** (لو الـ **dev team** طلبو حضوره معاهم) بيعملوا ميتنج إسمه **Daily Standup Meeting** كل واحد بيقول هو عمل إيه امبارح وهي عمل إيه النهاردة ولو فيه أي صعوبات واجهته، بحيث يكون كل التيم عارف كل واحد واصل لفين وينسقوا الشغل مع بعض، وفي آخر الـ **sprint** بيعملوا **retrospective meeting** بأهم الدروس المستفادة والمشاكل اللي حصلت أثناء الـ **sprint** دى لتفاديها في الأسبرينتات الجاية.

الـ **Agile process** يتمشى بالطريقة دي:



وعشان تعرف معلومات أكثر عن Agile ممكن تدخل ع اللينك ده:  
[https://en.wikipedia.org/wiki/Agile\\_software\\_development](https://en.wikipedia.org/wiki/Agile_software_development)

### :Testing within a Life Cycle Model

أيا كان الـ **model** اللي ماشى عليه، فيه مجموعة خصائص مشتركة بينهم كلهم عشان تعمل تيسىت زى البنى أدمن.

الخصائص دى هي:

- كل **development activity** بيقابلها **testing activity** (عشان نضمن اننا غطينا كل مراحل البرنامج، وكمان عشان نكتشف الأخطاء بدرى على قد مانقدر وبدقه عاليه على قد مانقدر).
- كل **test level** لازم يكون ليه أهداف واضحة عشان نقدر تحديد الاتجاه اللي هنمشى فيه ف الشغل بتاعنا.
- عملية الـ **analysis** والديزاین لأى **test level** لازم تبتدى أثناء الـ **development activity** الموازية ليه.
- التيسيرز لازم يكونوا مشاركين فى مراجعة الـ **documents** المختلفة من البداية خالص عشان لازم يكون فاهم كل شئ، وكمان لو فيه حاجة مش واضحة أو ناقصة أو bug معينة تتلحق من البداية.
- الـ **test levels** ممكن تندمج أو تتنظم مع بعضها حسب طبيعة الـ **project** أو الـ **system** **architecture** **Commercial-Off-The-Shelf (COTS)**، يعني لو عندى برنامج (COTS) المفترض إنه هيفى جزء من سيسنتم معين أو هيتعامل معاه، ففى الحالة دى ممكن نعمل **integration testing** على مستوى الـ **system level** (يعنى هنعمل **integration testing** بين السيسنتم والـ **COTS**) وكمان **acceptance testing** بين الـ **2 systems** (Functional and/or non-functional, and user and/or operational **(testing)**.

### :Different stages of SDLC with STLC

إيه بقى حكايتهم دول؟

**SDLC:** Software Development Life Cycle.

**STLC:** Software Testing Life Cycle.

فهنا هنشوف الفرق بينهم بالظبط

STLC (Testing)	SDLC (Developing)	Stage
بنراجع ع requirements ولازم تتأكد أنها واضحة ومحضرة وماتحتويش على أى كلمات غامضة .may not may	بنأخذ ال requirements من العميل عشان تعمل أول خطوة ف ال software development.	Requirements -1 Gathering
بنراجع على ال functional specifications إنها دققة (أصل دى اللي هيتبني عليه كل شئ بعد كده).	بنوصف الشكل الخارجى وبنشوف ال functionality هتمشى إزاي وبنحدد المعلومات والبيانات اللازمه للديزاین.	Functional -2 Specification
بنراجع على كل ال design products (ال UML مثلا).	ال software specifications بتتحول ل design models التفاصيل بتاعة ال data system structures interface architecture وال components المختلفة.	Design -3
بنراجع ع الكود عشان نطلع defects وتعالج بدرى وكمان لتحسين جودة الكود نفسه.	هنا ال designs كلها بتتحول لأكوا德.	Code -4
Unit testing + Integration testing	بيتعمل different software units وبيتعمل بينهم integration تمهدًا لبناء سیستم كامل.	Building -5 Software
System testing	بيتم الشغل فى ال non-functional requirements زي installation procedures وال ...configurations إلخ.	Building -6 System
Alpha & Beta testing (Acceptance testing)	البرنامج جاهز للعميل عشان يستخدمه	Release for -7 use

## Definition of testing

اتكلمنا خلاص عن الـ **SDLC** وعرفنا ان التيسينج دى مرحلة مهمة جدا من ضمن مراحله، لكن يعني ايه أصلا **testing**؟

التيستينج هو عمل analyzing و execution للسوftware عشان:

- نتأكد إن السوftware يتحقق requirements معينة.
- لإكتشاف bugs أو errors.
- قياس نتائج معينة أو تقييم نواحي معينة ف البرنامج بعد تشغيله في ظروف معينة.



## Testing Misconceptions

فيه فكرتين متاخدين غلط عن التيسينج لازم ناخذ بالنا منهم:

1- "التيستر لازم يتأكدوا إن البرنامج خالي من الbugs بنسبة 100%".

الحقيقة إن الفكرة ده غلط تماما زى ماقلنا قبل كده فى part no.2 فى صفحة 13 (Exhaustive testing is impossible).

2- "تأثير التيستر ببيان بعد الbugs اللي طلعها قبل مايروح السوftware لليوزر".

كمان الفكرة دى غلط، تأثير التيستر ببيان فى حالتين:

- نوع الbugs اللي بيطلعها (Critical - High - Normal - Low): يعني تأثير التيستر الشاطر من التيستر العيان ببيان لو طلع bugs critical كتير ومؤثرة فعلا.
- احتمال اكتشاف اليوزر لbugs أثناء التشغيل الفعلى: اليوزر مايهموش التيستر طلع كام بجاية قبل مايطلع البرنامج للتشغيل الفعلى، المهم عنده ان البرنامج مايطلعش فيه أى أخطاء وهو شغال خصوصا لو bugs دى تقيلة و بتوقف شغل البرنامج (Critical)، فكل ماتقل احتمالات حدوث bugs خطيرة كان ده مؤشر كويس على مدى كفاءة تيم التيسينج ف الشركة دى.

## Day 2

### :Testing and Quality

قلنا قبل كده إن التيسينج بيديننا الثقة في ال quality بتاعة السوفتوير لو لقينا بحات قليلة أو مالاقيناش خالص، حتى لو لاقينا quality defects بتزيد لما البجات دى بتصلح، وساعتها بنعرف أسباب حدوث ال defects دى فنجنها بعد كده ف الشغل، وده برضه بالتأكد بيساهم في ارتفاع مستوى جودة السيستم اللي بيتعمل.

كل الكلام ده يودينا لسؤال مهم جداً... يعني إيه ال software quality؟

### :Software Quality

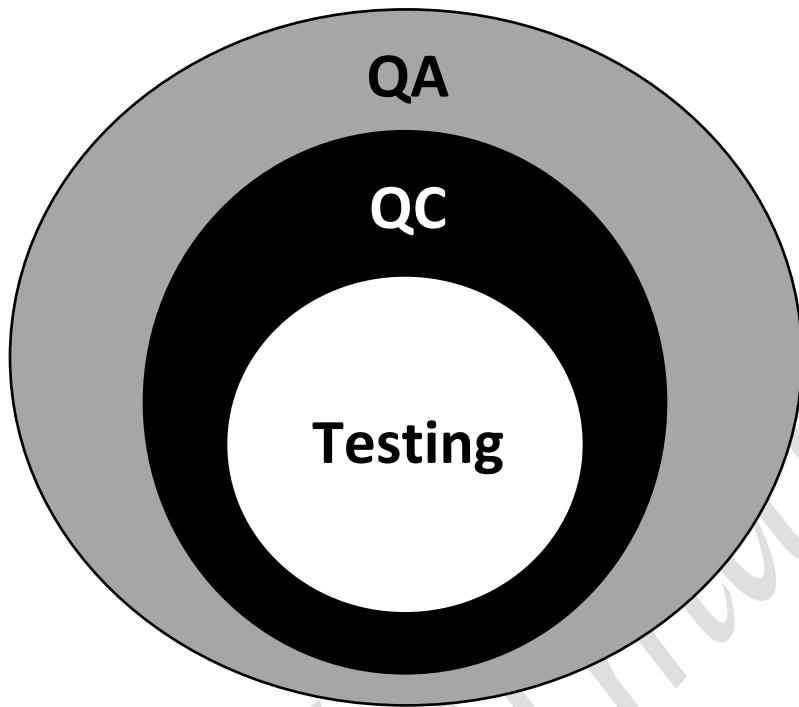
طبقاً للتعریف ال IEEE Standard Glossary of Software Engineering Terminology فال process هي الدرجة اللي عندها السيستم أو component منه أو Software Quality معينة تحقق requirements معينة أو تتحقق متطلبات أو احتياجات اليوزر.

### :Testing, QA, and QC

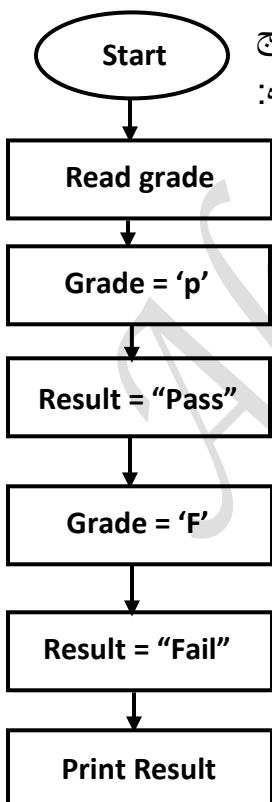
دائماً بنسمع ال 3 مصلحات دول ... فإيه بقى الفرق بينهم؟

بص ياسيدى:

- ال bugs هي عملية اكتشاف ال testing
- ال Quality Control (QC) هو التأكد إن السيستم تم بتحقق طلبات اليوزر وال requirements اللي طلبها (ال validation يعني)... التيسينج جزء من مهام ال QC لكن QC كمان بيهم بمنع ال defects قبل ما تحصل عن طريق تحسين محتوى ال documents وأى شئ يخص السوفتوير.
- ال Quality Assurance فهو مفهوم شامل على العملية كلها، ال QA هدفه الأساسي التأكد من إن ال process ماشية على standards معينة (CMMI – ISO ... إلخ). بحيث برضه يمنع ال defects الناتجة عن عدم اتباع ال standards من إنها تحصل في المستقبل، ال QA عملية كبيرة من ضمن أجزائها ال Testing و ال QC ، الشكل الجاي هيوضح الدنيا أكثر:



### لسه الدنيا مش واضحة؟ طب تعالوا نشوف المثال ده:



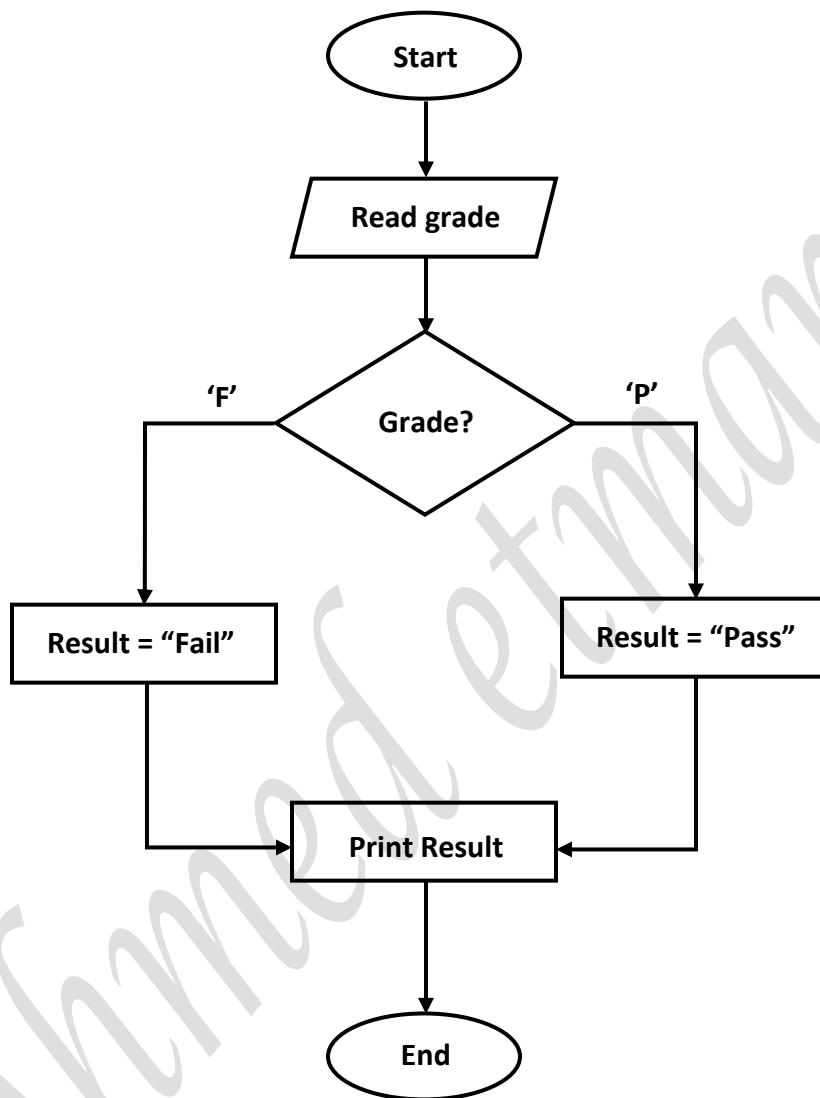
لو فرضنا اننا هنعمل برنامج يستقبل الدرجة بتاعة طالب، لو الطالب كتب p البرنامج هيطبع إنه ناجح ولو كتب f فهيطبع إنه ساقط، فلو كانت ال flowchart بالمنظر ده:

لو هنتكلم من منظور ال testing مبدأياً كده الكلام ده كله غلط لأنى حتى لو دخلت ال p قيمة لل grade هيطبعلى failed ف الآخر.

أما لو هنتكلم من منظور ال QC فالكلام ده برضه غلط لأن العميل طالب إن البرنامج لو كتب p هيطبع pass ولو كتب f هيطبع fail وده مابيحصلش دلوقتي.

اما لو هنتكلم من منظور ال QA فالكلام ده كله غلط تماماً لأن المفروض الشرط يتحقق ال flowchart بشكل ◇ مش مربع زى ما هو معمول دلوقتي، وكذلك لما بتقرأ input بنعبر عنه بالشكل ده □

فلو صلحناه وبقى الكلام كده:



فكده من منظور الـ **testing** الدنيا تمام لأن لو دخلت P هيطلع Pass ولو دخلت F هيطلع Fail.

ومن منظور الـ **QC** الدنيا تمام لأن حق اللي عايزه اليوزر.

ومن منظور الـ **QA** الدنيا ماشية تمام لأن الـ **flowchart** ماشية على الـ **standards** المعروفة بتاعتتها.

### :Testing and Debugging

Testing	Debugging
The activity that initially finds failures in a software item, by running test cases.	The development activity that finds, analyses, and removes the cause of the failure.
Done by testing team.	Done by developing team.
Intention behind is to find as many as possible.	Intention is to remove those defects.
Can be done without looking at the code	Need to work with the code
Can be automated or partially automated	Can't be automated
Can be manual	Use a debugger

## Day 3

### :Developing vs. Testing.... A way of thinking

بيقولك إن الـ mindset المستخدمة في الـ developing تماماً عن الـ testing، لأن الـ creative developing الهدف منها إنى أعمل شئ باستخدام الكود، أما التesting فهو عملية هدامة destructive الهدف منها إنى أبوظ الكود وأطلع عيوبه، وبالتالي ماحدث ينفع يشتعل في الوظيفتين دول مع بعض، ولو اشتغل مش هيبقى مظبوط في ناحية من الناحيتين دول.



غالباً الـ developer مايحبش يطلعه بجات في شغله... الإنسان بشكل عام في الحياة مايحبش بيان إنه غلطان أو عمل حاجة غلط، وكمان مايحبش حد يقعده يطلعه في الغلطات دى، وده بيفسر سر العلاقة اللي مش تمام بين الـ developer والـ tester لو التester خد الموضوع بشكل انتقاد شخصي، عشان كده خليك خفيف ع الناس لما تعلّمهم بجات وماخذش الموضوع بشكل شخصي.

### :Developers vs. Testers

تعالوا نشوف الفرق بين الاثنين عشان نفهم كل واحد ماشى في دنيته إزاى:



#### :Developers

- هما اللي بيكتبوا الكود - ناس مبدعة - من غيرهم ما فيهش سيسا تم أصلًا.
- مش شرط يكون عندهم communication skills كويسة.
- غالباً بيبقوا متخصصين في لغة برمجة أو أكثر (زى Java - C# - Python ... إلخ).



#### :Testers

- الأشرار بتوع المكان - بيفرحو بس لما بيلاقوا bugs وأخطاء كتير.
- لازم يكونوا كويسين في الـ communication skills ويتصرفوا بدبليوماسية خصوصاً مع الـ developer.
- يفضل يكون عندهم مهارات كتير سواء team skills أو technical skills أو testing skills.

### :Test-to-pass and test-to-fail

بمناسبة التيسيرز... هما في شغفهم بيشتغلوا بطريقتين ال **test-to-fail** وال **test-to-pass** ... فتعالوا نشوف الفرق بينهم.

#### :Test-to-pass

- بيتأكد إن السوفتوير ع الأقل شغال تمام.
- مابيحملش جامد على إمكانيات السوفتوير.
- ال **test cases** فيه بتكون بسيطة وواضحة.
- مابيحاولش بيوظ البرنامج.

#### :Test-to-fail

- بيعمل **design** لل **test cases** run ويعلمها **run** بعرض إنه بيوظ السوفتوير.
- ال **test cases** اللي بتتنفذ النوع ده بيتم اختيارها بدقة بحيث تبوظ أكبر قدر ممكن من السوفتوير.

طب عشان التيسير يوصل لأعلى مستوى وكفاءة في شغله... إيه اللي لازم يعمله؟  
الإجابة هنا:

### :Objectives of a software tester

- لازم التيسير يفهم ال **application** كويس جدا (من ناحية البيزنيس).
- لازم التيسير يدرس ال **functionality** بالتفصيل عشان يعرف ال **bugs** احتمال تحصل فين أكثر.
- لازم التيسير يدرس الكود عشان يتتأكد إن التيسير بتاعه مغطى كل سطر كود (لو شغال **white box** طبعا).
- لازم التيسير يعمل ال **test cases** بتاعته بحيث تطلع ال **bugs** المخفية ويتأكد كمان إن السوفتوير قابل للإستخدام وموثوق فيه.
- وطبعا لازم يلاقي ال **bugs** بدرى على قد مايقدر ولازم يتتأكد إنها اتحلت.

## Day 4

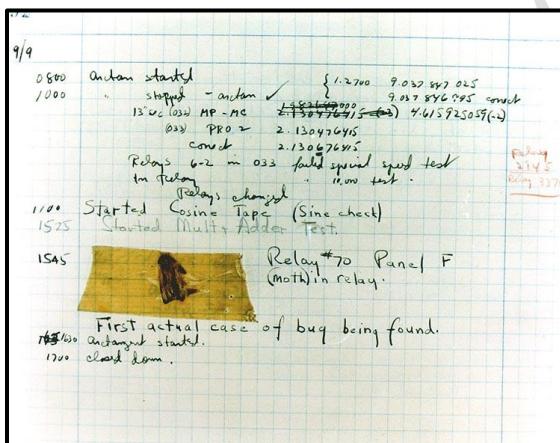
### What is a "bug"?

قلنا قبل كده إن:

- هو أي حاجة بيعملها الإنسان ينتج عنها نتيجة غير صحيحة. **Error**
- هو توصيف **error** فى السوفتوير، ولو البرنامج اتعمله **Defect - Fault - Bug**
- بالدوى ممكن يسبب **bug execution failure**
- هو انحراف السوفتوير عن الخدمة أو الحاجة اللي متوقع إنه يعملها. **Failure**



فالfailure ده event fault فهو حالة في السوفتوير نتجت عن error، يعني من الآخر bug هى عبارة عن اختلاف بين الـ expected result والـ actual result ف السوفتوير.



### Why we call them "Bugs"?

مصطلح bug تم استخدامه لأول مرة عن طريق رائدة الكمبيوتر **Grace Hopper** اللي نشرت أول حالة عطل حصلت في تاريخ الكمبيوتر.

Grace Hopper دى كانت طابط في البحريية الأمريكية وكانت شغالة على جهازين كمبيوتر في جامعة هارفارد ممولين من البحريية الأمريكية، الجهازين دول إسمهم **Mark III** و **Mark II**.

المهم... في سنة 1946 الناس المسئولة عن تشغيل الكمبيوتر **Mark II** لاحظوا وجود error في شغل الجهاز، بعد ما دوروا على أسباب المشكلة دى اتضح وجود **حشرة ميتة** (ليه ف الصورة دى) على جزء من أجزاء الكمبيوتر وهى اللي كانت معطلة الدنيا، فشالوها بعنایة شديدة وتم تسجيل الكلام ده في ال **log book**، ومن ساعتها تم إطلاق مصطلح bug على أي error يتم اكتشافه في البرامج، ودى صورة من ال **log book** اللي تم تسجيلها بخصوص الحدث التاريخي ده.

### :Actual Result vs Expected Result

قولنا إن الـ **bug** هو أي اختلاف بين الـ **actual result** والـ **expected result** ... طب إيه هما الـ **actual & expected results** دول؟

الـ **Actual Result (AR)**: هي النتيجة الفعلية اللي طلعت معايا لما عملت **execution** للبرنامج.

أما الـ **Expected Result (ER)**: فهي النتيجة اللي كنت متوقعها والمفروض إنها تطلع.

ف العادي الـ **Expected Result** بتكون موجودة في الـ **Requirements**، لكن فيه مصادر تانية كتير للـ **ER**.

زى إيه مثل؟

تعالوا نشوف ....

### Other sources of Expected Results:

- Life experience.
- Common sence.
- Communication.
- Standards.
- Statistics.
- Valuable opinion.
- Other sources

تعالوا نشوف كل واحد فيهم بالتفصيل ...

### :Life experience

أى شئ اتعلمنا منه حاجة سواء عن طريق النت أو القراءة الكتب أو الشغل أو الدراسة أو حتى بالتعاملات اليومية مع البشر... إلخ، كل ده يندرج تحت بند "life experience"، عشان كده لو ما فيهش **specifications (requirements)** فى حاجة معينة ممكن نعوضها من خبرات الحياة المختلفة .**expected result** ونقدر نستنتج منها الـ

**مثال:** لو ماقيش عندى specifications والديفلوير عمل رسالة بتقول "error" على حاجة غلط حصلت بدل مايدبني رسالة بتوضح الغلط فين، ممكن كتيستر ف الحالة دى أعمل bug report بالرسالة دى وأقول فيها إن الرسالة دى لازم يكون فيها شئ مفيد لليوزر عشان يقوله الغلط اللي عمله فين بالظبط و/أو المفروض يتصرف إزاي...

طب على أى أساس قلت الكلام ده؟

لإن طبقة الخبرة اللي عندى الليوزر مش هيستفيد حاجة ومش هيعرف يتصرف لو قلتله "error" بس كده وسيته عايش مع نفسه... لازمأوضحله الخطأ فين ويعمل إيه عشان مايحصلش معاه الخطأ ده.

ف الحالة دى وارد إن الديفلوير يقول إنه شايف إن "Error" كفاية... وهذا يبدأ الجدال، فأفضل حل للحوار ده إننا يكون عندنا specifications كافية فيها الكلام ده بوضوح شديد عشان تبقى الأساس بتاعنا واحدنا بنتكلم مع بعض.

### **:Common sense**

لو قلنا إننا بنتيست website فيه إمكانية لليوزر إنه يرفع صوره عليه، ال specifications بتقول إن الليوزر يقدر يرفع صورة واحدة بس ف كل عملية upload، لكن إيه الوضع لو الليوزر عايز يرفع 200 صورة؟ هي عمل عملية ال upload ويكررها 200 مرة؟ الكلام ده مش منطقى طبعا.

عشان كده لازم أتكلم مع ال stakeholders وأوضح لهم النقطة دى ونشوف الدنيا هتمشى إزاي.

### **:Communication**

حتى لو ف إيدك أحسن specifications ف العالم هتفضل فيه جزئيات معينة مش واضحة أو ناقصة تفاصيل ولو صغيرة، ف الحالة دى لازم تتواصل مع أى حد يفديك ف اللي انت بتسأل عليه سواء زميلك أو ال stakeholders أو أى حد تاني يساعدك، وماتتسفتش إنك تسأل لإنك كتيستر لازم must نفهم كل حاجة ف السيستم عشان تطلع الدنيا مطبوبة.

### **:Standards**

#### **مثال:**

لو عندي industry standard بتقول إن الليوزر لازم يستقبل ميل confirmation على website اللي عمله على ال registration، يبقى لازم ف الحالة دى ال website يلتزم بال standard ده، ولازم أعمل حسابي على كده كمان وانا بعمل التيسينج ع الموقع ده.

### مثال تانى:

لو الشركة اللي شغال فيها عندها **standard** داخلية ماشية عليه إن رسائل ال **error** لازم تكون بال **format** ده: (الخط "Times New Roman" – لون الخط "#FF0000" (ده الكود بتاع اللون الأحمر) – وحجم الخط "110%").

يبقى ف الحالة دي خلاص بقت معروفة إن ال **format** ده بقى **expected result** لازم ألاقيه موجود في كل رسالة **error**.

### :Statistics

طبقاً لموقع [/https://www.akamai.com](https://www.akamai.com) فمتوسط الوقت اللي هيستناء ال **online shopper** على الموقع عقبال ما تحمل أي صفحة فيه هو **4 ثوانى** بس قبل ما يسيب الموقع ده ويدور على حاجته في موقع تانى (طبعاً موقع البيع الأونلайн مافيش أكثر منها وكل موقع عايزة يجيب أكبر عدد ممكن من الزبائن، فوقت تحميل الصفحة بالنسبة لهم مهم جداً).

الإحصائية دي ممكن استخدمها كأساس أقدر أنسد عليه وانا بعمل **bug report** عن ال **performance** بتابع الموقع وووقت تحميل الصفحات وكده.

الكلام ده مهم جداً لأن العميل بمجرد ضغطة زرار ممكن يقفل الموقع بتابعك ويروح لموقع تانى منافس.

### :Valuable Opinion

ده ممكن يكون رأى المدير بتابعك أو أى حد تانى خبير في الحاجة اللي بتسائل عليها، فرأى الناس دول مهم (مش تطبيق والله لكن فعلًا الناس دي قديمة وخبرة ورأيهم مهم).

### :Other sources

ممكن يبقى فيه **sources** تانية أقدر أعتمد عليها طبقاً لطبيعة ال **project** اللي شغال فيه، يعني مثلاً لو بعمل تيست على موقع لنشر الصور زي [/http://www.photobucket.com](http://www.photobucket.com) فانت تحتاج في الحالة دي تقرأ أكثر عن الحوار ده من المجلات المهتمة بال **professional photographers** عشان تعرف أكثر عن المجال ده.

## Day 5

### :Software bugs and spec bugs

قلنا قبل كده لو ال specifications ناقصة والديفلوبير كتب كلمة "Error" بس فى رسالة الخطأ الى بتطلع لليوزر وإن من حقى كتىستر أعمل report بالكلام ده، وإن ما فيه مشكلة لو خلينا رسالة الخطأ اللي بتطلع لليوزر تكون شبه دى كده :

"Oops, error on page, ZIP code should have 5 digits"

ف الحقيقة الديفلوبير رسميًا سواء كتب دى أو دى مش غلطان، ده لأن ال specifications مواضحتش بالضبط ال text اللي المفروض يتكتب فى ال error message، ففى الحالة دى ال bug طالع من ال specifications لأن النتيجة الفعلية (ما فيه text للرسالة) غير النتيجة اللي كان المفروض تكون موجودة فى ال specifications، ف الحالة دى بنسمى الموقف ده "Spec bug".

أى بجاية نلاقيها (سواء ف السوفتوير أو ف ال specifications) لازم يتعلملها report فى ال bug tracking system (هنشوفه بعددين)، وما فيه أى استثناء نهائيا لأى bug ف الحوار ده، ال professional tester يعتبر من أكثر الحاجات المهمة اللي بتتل على إنك reporting.

لازم تأخذ بالك إنك كتىستر لما تعمل bug report لديفلوبير معين ده مش معناه إنك بتكره الشخص ده أو مابتحترموش، بمنتهي البساطة انت لاقيت mismatch بين ال expected result وال actual result وبتطلب منه إنه يصلح المشكلة دى.

لو حد من الديفلوبيرز خد ال bug على إنها انتقاد لشخصه فلازم تظهر له إحترامك ليه ويكون عندك الصبر إنك تشرحله إن ما فيه حاجة شخصية ف الشغل وإن الكلام ده جزء من شغالك.

### :Causes of software defects

قلنا قبل كده إن:

- الإنسان ممكن يغلط ويعمل error .
- ال document error ينتج عنه defect/fault/bug فى الكود أو فى ال .
- ولو حصل execution bug دى لسه موجودة بينتاج عنه failure:
  - يالما فشل إنه يعمل حاجة المفروض يعملها.
  - يالما عمل حاجة المفروض ماي عملهاش.

فإيه بقى أسباب حدوث ال bugs دى؟

### :Causes of software defects

#### The human factor •

- Humans make mistakes ○
- ممکن الواحد يعمل error بسبب نقص في المعرفة والتدريب عنده.
- ممکن الواحد مع ضغط الوقت يغلط من غير مايقصد.
- ممکن الإنسان مع الكود المعقد اللي الدنيا فيه ملختطة يفقد جزء من تركيزه ويغلط ف حاجة معينة.
- ممکن ال infrastructure نفسها تكون معقدة لدرجة إن الإنسان يفقد بررمه تركيزه عند نقطة معينة ويعمل خطأ.
- ممکن الإنسان لو متعدد يشتغل على technology جديدة أو تانية ممکن النقلة دى تخلية يغلط أو تسقط منه حاجات.

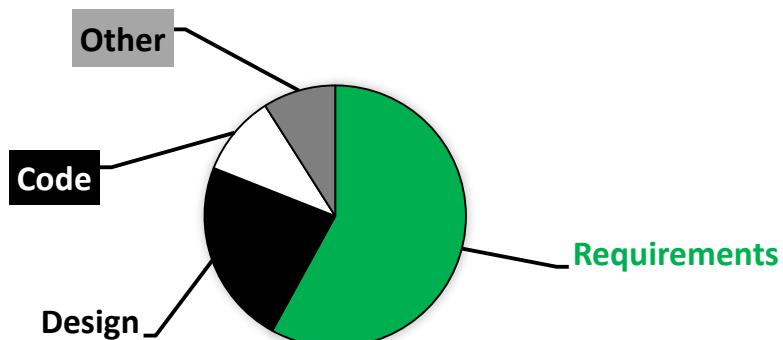
#### :Organizational factors •

- Inefficient communication ○
- Unclearly defined requirements ○

#### :Environmental conditions •

- زى الإشعاعات - المجالات المغناطيسية - المجالات الكهربائية - التلوث...الخ.
- كل دول ممکن يغيروا من حالة الهايدروير فممکن تأثر على الشغل بتاعنا (زى لو بنعمل مثلا برنامج لقياس درجة الحرارة أو درجة التلوث...الخ).

أغلب defects وال failures بتحصل نتيجة خلل في requirements وبدرجة أقل من design وبدرجة أقل من الكود وبدرجة أقل خالص من أسباب تانية... زى الشكل كده مثلا:



ال الكلام ده هيبيان أكثر ف المثال الجاي:

مثال:

لو فرضنا اننا عايزين برنامج ندخله الرقم يطبعنا التربيع بتاعه يعني لو قلتله 3 يطلعلي ناتج الـ  $3^2$  اللي هي بـ 9، فالديفلوبير عمل الكود بالشكل ده (ده كود C#):

```
static void Main(string[] args)
{
    Console.WriteLine("Enter x");
    int x = int.Parse(Console.ReadLine());
    int result = x * 2;
    Console.WriteLine("Square (x) = {0}", result);
    Console.ReadKey();
}
```

- هنا الديفلوبير عمل error لما فهم إن تربيع الرقم هو ضربه في نفسه.
- فالكلام ده نتج عنه bug إنه عرف الـ result على إنها  $2 \times X$  زى ما هو مكتوب ف السطر اللي متحدد بالمربع ده.
- وبالتالي لما يتعمل run للكود هيحصل failure ف العملية كلها لأنى لو دخلته رقم زى 3 كان المفروض يطلعلى 9 إنما هو طلعلى 6.

**Note:** دايما خليك فاكر إن التيسىت اللي مابيطلعش حاجة مضيعة للوقت .. التيسىت الناجح هو اللي يطلع أكبر قدر ممكن من الـ bugs والـ failures.

## Day 6

### What are the “right” things to test?

عشان التيسست بتاعنا يكون مؤثر لأبعد درجة ممكنة لازم نسأل نفسنا الأسئلة دى:

- هل بنعمل التيسست على ال component كله ولا product منه؟
- إيه ال critical functions ف ال product ده؟
- إيه ال functions اللي بيحصل فيها fail كتير؟
- إيه ال functions اللي عرضة للخطأ بدرجة كبيرة؟
- إيه ال functions اللي بتسخدم كتير؟
- إيه ال functions اللي اتغيرت أو بتتغير علطول؟
- إيه ال functions اللي بتتعب الموظفين في شغفهم؟
- فين المناطق اللي ف السيستم اللي فيها شغل كتير؟
- إيه ال technologies اللي بطلب skills جديدة؟

وعشان الكلام ده كله صعب يتعمل كله مرة واحدة فلازم نعمل prioritization للكلام ده كله.

### :Fail is not always a bug

لو نتيجة ال test case execution failed فده مش مقاييس بالضرورة إن ده bug، فمثلاً حصل requirement change request معينة من غير مايعرف التيسستر بالحوار ده، فلما بيجي التيسستر يشتغل ع الجزء ده يلاقى ال expected result اللي عنده غير ال actual result اللي طلعت فيعملها bug report، ف الحقيقة ال Fail مش من البرنامج نفسه لكن بسبب خطأ في ال expected result اللي عند التيسستر.

### :Test levels

قلنا قبل كده إن فيه 5 test levels هما:

Unit testing -1

Integration testing -2

System testing -3

Acceptance testing -4

Maintenance testing -5

فهنتكلم دلوقتى عنهم بالتفصيل:

### :Unit testing -1

ال unit هى أصغر حاجة ف السوفتوير أقدر أعمل عليها test ، ممكن تبقى Database Stored function معينة أو Report عن حاجة أو حتى form معينة (صفحة معينة فى برنامج).

بنعمل ال unit test عشان نتأكد إن ال unit/component ده بتحقق ال functional requirements اللي مطلوبة منها أو بتتوافق مع ال design structure specifications اللي احنا مخططين له.

ال unit testing بيتعمل بعد ما يخلص الكود بتاع ال module اللي عليه الكلام، وبيعمله الديفلوبير لكن مافيش مانع إن التيسير يعمله تانى بعد الديفلوبير.

طب ليه بنعمل ال unit testing ؟

تخيل لو ديفلوبير X خلص ال module بتاعه وسلمه للتيستر من غير ما يعمل عليه تيست وكان شغال معه ف التيم ديفلوبير Y و ديفلوبير Z.

جه التيسير يعمل شغله قام مطلع Bug لديفلوبير X، وبالتالي لازم ال module بتاعه يرجعله عشان يصلحه، وبالتالي شغل Y و Z هيتعطل بناء على الشغل اللي يصلحه X.

خلص X شغله وجيت طلعت بجایة L Y اللي مترب عليها شغل Z و module تانى L X، وبالتالي شغلاهم برضه هيتعطل لغاية ماتتحل البجاية دى، وبالتالي هنفضل ف الدوامة دى فترة طويلة.

إنما تخيل بقى لو  $x \wedge z$  كل واحد فيهم عمل تيست ع الـ **module** بتاعه قبل ماييعته للتيستر، بالتأكيد البجات هتقل بشكل كبير، وبالتالي التيست هيأخذ وقت أقل، فالدنيا تمشي بشكل أسرع ف الشغل، ع الأقل مش هنوقف الدنيا عشان بجات تافهة وبسيطة حصلت والديفلوبر ماخدش باله منها.

### :Integration testing -2

هنا بقى بنادد الـ **modules** اللي خلصت ونبداً نجمعهم مع بعض عشان نشوف الـ **interaction** بينهم واصل لفين، الـ **integration** ده أول خطوة في تكوين ملامح السيستم ككل هيبي ماشي إزاي.

عندنا 3 أنواع لـ **integration** هنتكلم عن كل واحد منهم بالتفصيل وإزاي بنعمل تيست ف كل نوع.

### :Integration testing types

Big Bang testing -1

Top Down testing -2

Bottom Up testing -3

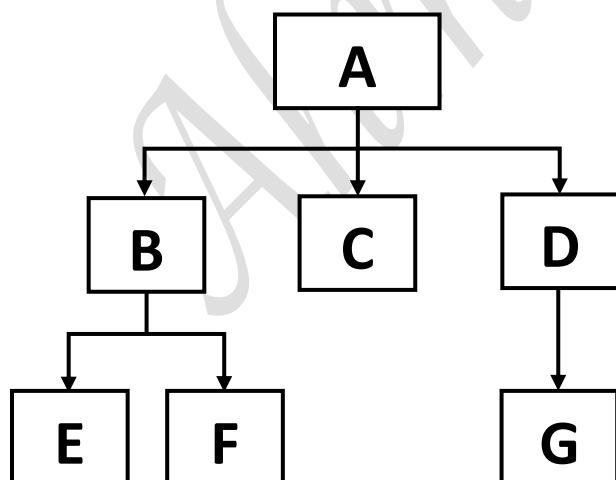
فتعالوا نشوف إيه حكاية كل واحد فيهم

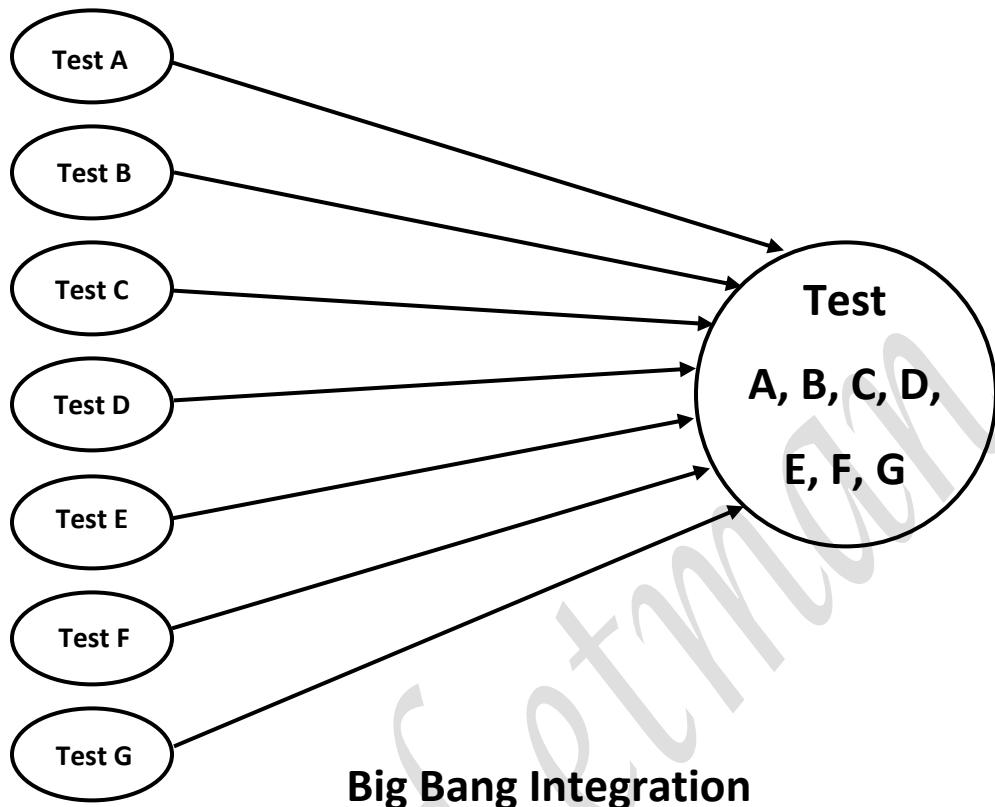
### :Big Bang testing -1

تعالوا نبص ع الشكل ده كده:

لو بصينا ع الشكل اللي ع الجنب ده هنلاقي **B, C, and D modules** تحته **module A** وال **B** تحتها الـ **E** وال **F** وال **D** تحتها **G**.

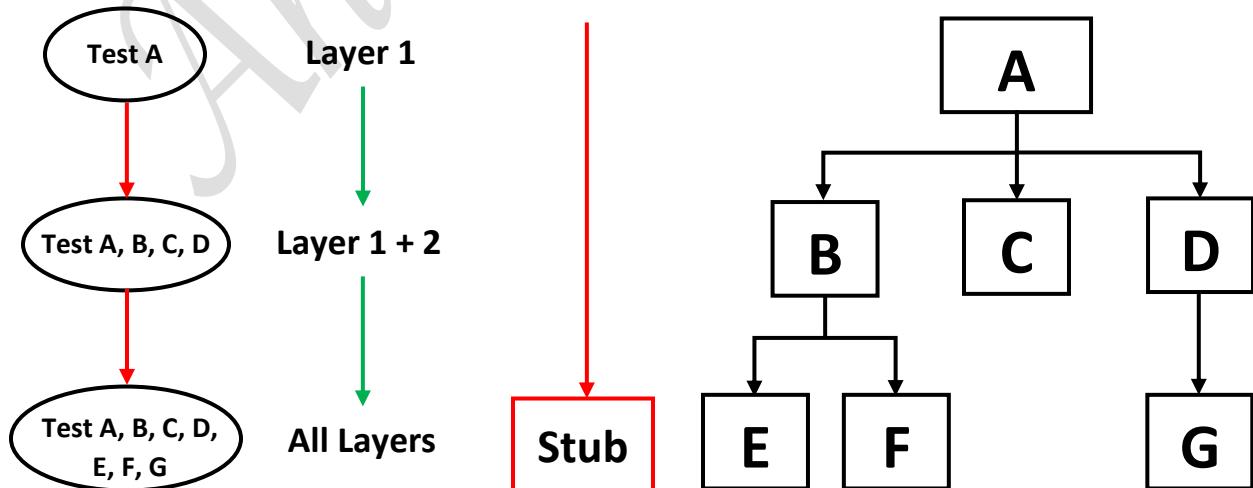
الـ **Big Bang Integration** فكرته إنك بعد ماعملت تيست على كل **module** فيهم بشكل منفصل تبدأ تعمل **Integration** بينهم كلهم مرة واحدة في عملية واحدة بس، زى ما هو واضح ف الشكل اللي ف الصفحة الجاية دى.





طبعاً الطريقة دي بتختصر جداً معايا ف الوقت، لكن عيبها إن لو طاعت Bug ف وسط الهيصة دى كلها بيبقى صعب جداً يتعمله fix وسط كل الزحمة دي وبياخد وقت طويل، عشان كده فكروا في طرفيتين لتسهيل الدنيا دي ف ال integration، الطرفيتين دول هما ال bottom up وال top down، فتعالوا نشوف ظروفهم إيه.

### :Top down testing -2



ف الطريقة دي بنتيست من فوق لتحت، بمعنى إنى هتيست أول Layer وبعد كده نشوف اللى تحتها علطول بالتدريج وبعد كده ننزل ع الـ layer اللي تحتها وهكذا، يعني ف الشكل اللي فوق مثلا هنتيست الـ A الأول وبعد كده ننزل ع الـ layer الثانية فنتيست الـ integration بين الـ A والـ B وبين الـ A والـ C وبين الـ A والـ D، بعد كده ننزل ع الـ layer الثالثة، فنتيست الـ integration بين الـ A والـ B وبين الـ E وبين الـ A والـ F وبين الـ A والـ G، وبالتالي تكون عملنا تيست على الـ modules بين كل الـ integration اللي ليها علاقة ببعض من غير وجع دماغ الـ big bang، بحيث لو لقينا bug معينة بنعرف هي سببها إيه وجایة من عند مين بالضبط.

طب لو كان الـ A جاهزة لكن B مش جاهزة وعايز أشوف الـ integration بينهم أعمل إيه؟

ف الحالة دي هتسخدم حاجة اسمها الـ Stub (هتكلم عنه كمان شوية).

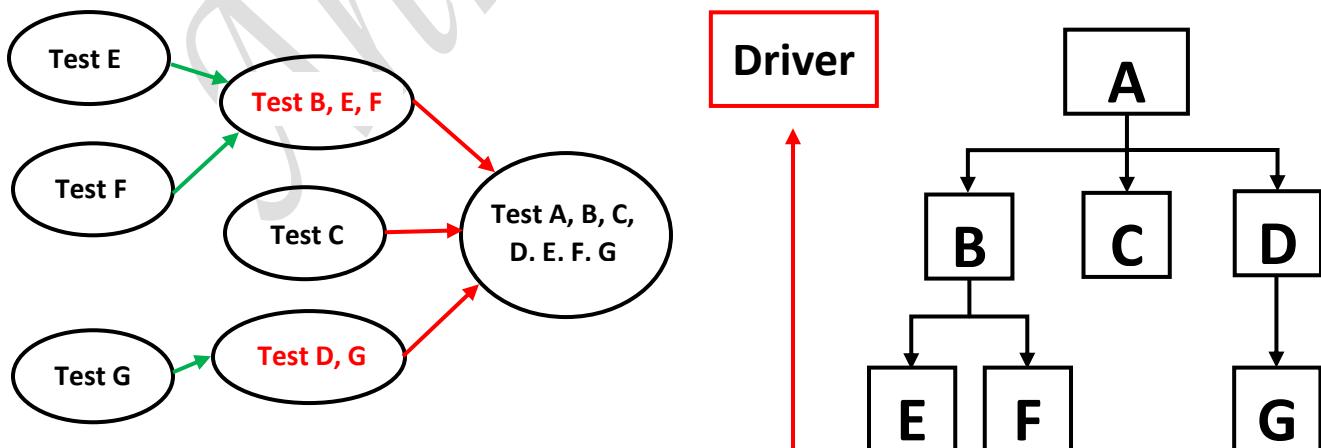
### :top down technique

- غالبا بيتم عمل تيست للحاجات المهمة جدا الأول.
- بنقدر نتحكم ف السيسن من بدري.

### :top down technique

- بيعتاج stubs أحيانا (فى حالة إن الـ Lower function ماكانتش جاهزة).
- التفاصيل بتتساب للأخر (يتبقى موجودة أكثر ف الـ lower functions).
- ممكن يكون صعب اننا نشوف detailed output (لكن لازم يتم التيست ع الكلام ده فى الـ component testing).
- ممكن السيسن بيان إنه خلسان أكثر من الواقع الفعلى.

### :Bottom up testing -3



أما بقى ف ال Bottom up فبتيست من تحت لفوق، بمعنى إنى بنتيست ال E الأول وبعد كده نشوف ال E مع ال B وكذلك نتبيست ال F الأول ونشوف ال B مع ال B ونشوف ال E وال B مع ال A وكذلك ال F مع ال B مع ال A، وكذلك برضه هنعمل التبيست على ال G وبعد كده نشوف ال G مع ال D وبعده كده نشوف ال D وال G وال A، وكذلك ال C كذلك برضه بنشوف ال A و هكذا.

في حالة إن ال B مثلًا مش جاهزة لكن ال E مثلًا جاهزة بنس تخدم حاجة إسمها Driver (برضه هنتكلم عنها دلوقتى).

### :Bottom up testing

- ال lowest levels بيتعمل عليها التبيست الأول، فلو فيه حاجة مش مطبوبة بتنطبع م الأول كده.
- ال external interfaces technique ده كويس واحداً بنتيست ال الخاصة بال .
- ال software أو hardware environment (سواء
- ال details بتنتفاف الأول، وبالتالي ممكن نتجنب مشاكل كتير ممكن تحصل بعد كده لو ركزنا مع التفاصيل الصغيرة الأول.

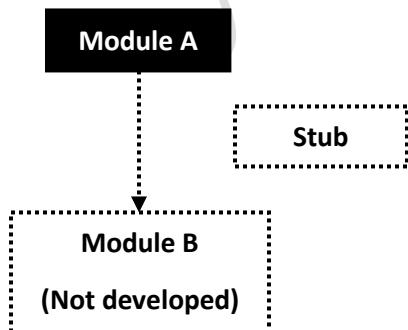
### :Bottom up testing

- مافيش سистем أقدر أشغله لغاية آخر baseline عندى.
- ال drivers technique ده بيحتاج ال .
- المشاكل العامة لل control بخلافها ف الآخر، ودى ممكن تعسفنا شوية حسب المشكلة.

### :Stups & Drivers

دلوقتى هنتكلم عن ال drivers وال stups اللي قولنا عليهم من شوية... إيه بقى حكايتهم دول؟

### :Stups



ال Stup هو Component بيتتم استدعاءه من ال unit اللي بنعمل عليها التبيست و بترجع fake values ساعة ما بنعمل ال top down testing، الهدف الأساسي منها إننا نتأكد إن ال unit بتاعتنا شغالة تمام، فالنتيجة بتاعة ال stub ف الحالة دي ماتهمنيش على قد ما يهمنى إنى أتأكد إن ال tested unit شغالة تمام.

مثال: لو عندنا 2 وحدة بتسingle رقمين والثانية بتعمل عليهم عمليات الجمع والطرح والضرب والقسمة، فهياكون الكود بالشكل ده (ده كود C#):

```
static void Main (string [] args)
{
    //this is the tested module
    Console.WriteLine("Please enter the 1st number");
    int a = int.Parse(Console.ReadLine());
    Console.WriteLine("Please enter the 2nd number");
    int b = int.Parse(Console.ReadLine());
    Calculate (a,b);
    Console.ReadKey();
}

static void Calculate (int x, int y)
{
    //this is the stub (dummy code)
    Console.WriteLine("This is the stub function");
} // end of the function
```

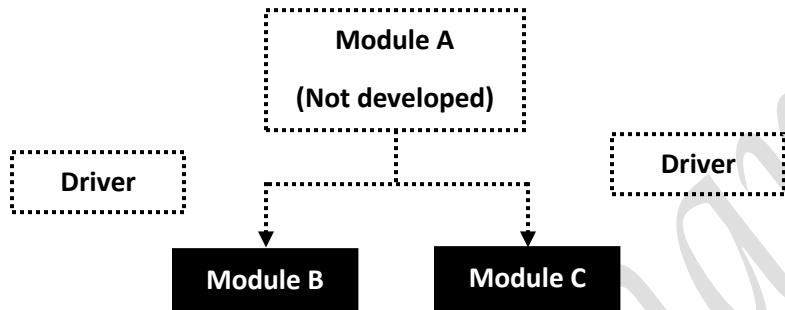
هذا المفروض الـ "Main" function بتستقبل رقمين وتبعتهم للـ "Calculate" function عشان تعمل العمليات الحسابية المختلفة.

اللى حصل إن الـ Main جاهزة لكن الـ calculate مش جاهزة لسبب ما... فكان الحل اذنا نكتب فى الـ main بتستدعي الـ calculate dummy code صح بالرقمين اللي دلهم اليوزر.

ف الحالة دى الـ calculate تكون stub ساعة مانحب نعمل calculate والـ main بين الـ .

**:Drivers**

أما الـ **driver** بقى فهو عبارة عن **tested unit/component** بيستدعى الـ **dummy code**، وبيطلع **values** من الـ **tested unit**، وبيطلع **bottom up integration testing** عشان نتأكد إنها شغالة تمام، فيكون شكله عامل كده:



عشان الدنيا توضح أكثر تعالوا ناخذ نفس الكود اللي فات ونشوف الفرق:

```

static void Main(string[] args)
{
    //this is the driver (dummy code)
    Calculate(8, 2);
    Console.ReadKey();
}

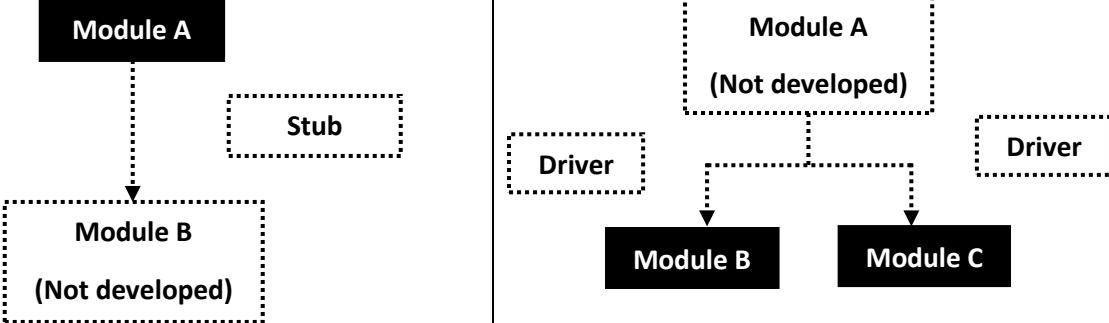
static void Calculate (int x, int y)
{
    //this is the tested module
    Console.WriteLine("Num 1 + Num 2 = {0}", x + y);
    Console.WriteLine("Num 1 - Num 2 = {0}", x - y);
    Console.WriteLine("Num 1 * Num 2 = {0}", x * y);
    Console.WriteLine("Num 1 / Num 2 = {0}", x / y);
} //end of the function
  
```

هنا بقى الـ Main مش جاهزة انها تستقبل رقمين من اليوزر لسبب ما، فاستخدمنا driver واحداً بعنوان driver على الـ Main في dummy code عبارة عن bottom up integration testing التي اتعمقت إلى Calculate بـ calculate بأي رقمين عشان بس أتأكد إن الـ calculate اللي اتعمقت ماشية مطبوع وإنها بتطلع العمليات الحسابية مطبوعة، هنا الـ Main هي الـ driver في الحالة دي.

### :Summary

فملخص الليلة دي تاني عشان مانتاخبطش:

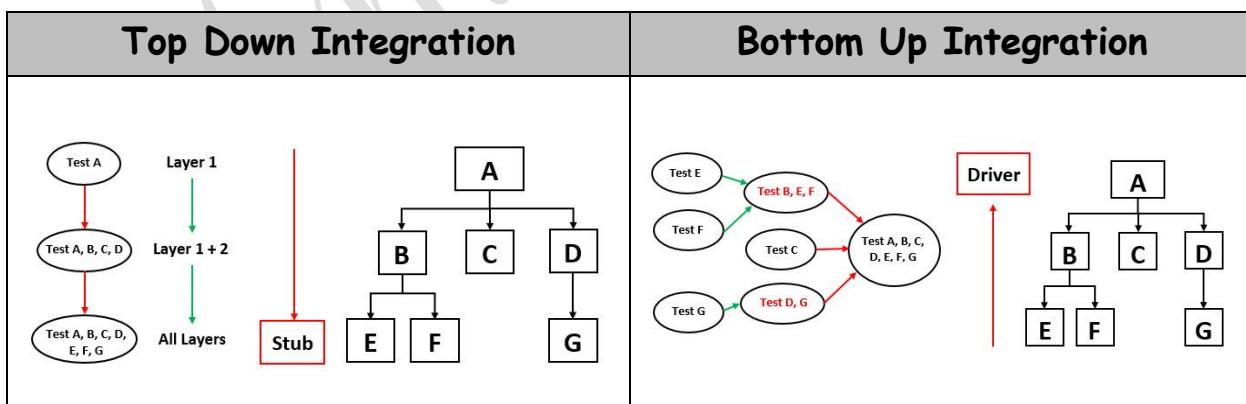
Stub	Driver
Used in <b>top down</b> integration	Used in <b>bottom up</b> integration
Dummy code in the lower level	Dummy code in the higher level
Called by the higher level	Calls the lower level



```

graph TD
    subgraph Stub [Stub]
        A1[Module A] --> B1[Module B<br/>(Not developed)]
        B1 --- S1[Stub]
        S1 --> A1
    end
    subgraph Driver [Driver]
        A2[Module A<br/>(Not developed)] --> B2[Module B]
        A2 --> C2[Module C]
        B2 --- D1[Driver]
        C2 --- D2[Driver]
    end

```



### :System Testing -3

هنا بقى بنتيست السيستم كله بسيناريوهات كتير جدا عشان:

- نتنيست ال functionality بتاعة السيستم ككل (functional testing).
- نعمل تنيست لل non-functional characteristics كمان اللي موجودة ف السيستم ككل (non-functional testing).
- نتأكد إن ال system configurations مطبوبة و بتسمع ف configuration data.
- السيستم بشكل صحيح.

ف الغالب بنعمل ال system testing (هنجيلهم كمان شوية) في ال Black box techniques، لكن ده مايمعنعش إننا ممكن نعمل ال White box techniques أحياناً برضه.

#### البيئة التي عمل لها ديزاين من: test cases

- System and software requirement specification
- use cases
- Functional specification

Risk analysis reports: التقارير اللي بتطلع عن أماكن معينة ف السيستم احتمال يكون فيها مشاكل كتير سواء مشاكل business أو مشاكل خاصة بال technical، فطبعاً لازم ناخذ الكلام ده ف اعتبارنا واحداً بعمل ال test cases.

ممكن ف السيستم تيستينج نستخدم ال unit test cases لكن لازم نعمل test cases جديدة تمشى مع ال business sequence بتاعة السيستم ككل.

واحداً بعمل ال system testing لازم نحاول بقدر الإمكان إن ال test environment تكون قريبة بدرجة كبيرة لل environment الحقيقة اللي هيستغل عليها السيستم عند العميل، وده طبعاً عشان نقل risk بتاع ال environment-specific failures (لاحظ إننا بنبدأ نجهز السيستم خلاص للتشغيل الفعلى عند العميل في المرحلة دي).

ملحوظة: في المرحلة دي بالذات لازم اللي يعمل التنيست independent test team بس (بدون مشاركة الديفلوبرز أو ال customers في عملية التesting).

### :Acceptance testing -4

هنا بقى بنهايأ السيستم للتشغيل الفعلى عند العميل، عشان كده التيست ده اللي بيعمله العميل أو اليوزر اللي هيستعمل السيستم بنفسه عشان:

- بيتأكد إن الـ business processes على السيستم كله مأشية تمام.
- بيتأكد إن عمليات التشغيل والصيانة على السيستم مأشية كويس وزى ما هو عايز.
- بيتأكد إن السيستم بيستجيب للأوامر اللي اليوزر بيديها للسيستم.
- بيتأكد إن الـ forms اللي طلبها اليوزر أو العميل موجودة زى ما هو عايز.
- بيتأكد إن السيستم بيطلعه الـ reports اللي هو عايزها.
- بيتأكد إن الـ configuration data بتحقق المطلوب منها وفعالة فعلاً ف السيستم.

الهدف الرئيسي من الـ Acceptance testing هو اكتساب الثقة ف السيستم أو أجزاء منه أو حتى non-functional characteristics معينة موجودة فيه، اكتشاف الـ defects ف المرحلة دى مش هو الهدف الرئيسي.

الـ Acceptance testing ممكن يقيم استعداد السيستم للـ deployment والتتشغيل الفعلى زى ما قلنا قبل كده، مع إنه ممكن ما يكونش آخر level ف التيسينج، يعني مثلاً الـ acceptance test (between 2 systems integration) integration test للسيستم.

- كمان الـ Acceptance testing ممكن يحصل أكثر من مرة ف الـ life cycle، زى مثلاً:
- الـ acceptance test ممكن يتعمله COTS software product لما يتعمله install أو لو فيه integration مع سистем تانى.
  - الـ Acceptance testing للـ component usability بتاعة component معين ممكن يحصل أثناء .component testing
  - لو فيه functional enhancement جديدة ممكن نعمل ف الحالة دى acceptance .system testing عليها قبل مانعمل الـ

### :Acceptance testing types

فيه 4 أنواع من ال acceptance testing

User acceptance testing -1

Operational (acceptance) testing -2

Contract and regulation acceptance testing -3

Alpha and beta (or field) testing -4

فتعالوا نشوف كل واحد فيهم بالتفصيل

### :User acceptance testing -1

ده بيتأكد إن السيستم مناسب لليوزرز عشان يستخدموه (سهل ف استخدامه وفهمه).

### :Operational (acceptance) testing -2

ده بقى بيقيس ال acceptance بناء السيستم من ناحية ال system administrators.

- Testing of backup/restore.
- Disaster recovery.
- User management.
- Maintenance tasks.
- Data load and migration tasks.
- Periodic checks of security vulnerabilities.

### :Contract and regulation acceptance testing -3

ال contract acceptance testing بيتعمل للتأكد من تحقيق المعايير والبنود الخاصة بقبول ال acceptance criteria المتفق عليها في العقد، وال custom-developed software لازم تتحط لما الطرفين يتوقعوا على التعاقد.

أما ال Regulation acceptance testing بيتعمل للتأكد إن السيستم لازم يكون متوافق مع القوانين والقواعد اللي لازم الالتزام بيها، زي اللوائح والقوانين الحكومية أو قواعد الأمان... إلخ.

#### :Alpha and beta (or field) testing -4

الديفلوبرز اللي بيعملوا برامج للسوق (COTS) عادة بيحبوا ياخدوا الـ feedback من العملاء الموجودين والمهتمين في السوق قبل ماينزل البرنامج للبيع بشكل تجاري.

##### :Alpha testing

ده بيتعمل في شركة البرمجة لكن اللي بيعمله مش الـ developing team (ممكن النايسنر مع الـ customer).

##### :Beta testing (field testing)

ده بقى بيعمله الـ customers برضه لكن في أماكنهم الخاصة والمؤسسات بتاعتهم وعلى أجهزتهم، ودى بتبقى آخر مرحلة حرفيا قبل التشغيل الفعلى، عشان كده دايما نسمع ان الشركات زي Microsoft نزلت البرنامج الفلاني beta version وبتفضل سايباه بشكل مجاني كده لفترة بتلم فيها الـ feedback من اليوزرر وبعد كده تنزل النسخة النهائية منه للبيع بشكل تجاري.

فيه بعض الناس بيقولوا ع الـ alpha testing و على factory acceptance testing .site acceptance testing الـ beta testing.

#### :Maintenance testing -5

بعد الـ beta testing والدنيا لو تمام بيس丞 العميل البرنامج وبيبدأ الشغل الفعلى عليه، طبعا بيفضل البرنامج شغال لسنين وسنين، في أثناء الفترة دي ممكن يحصل تغيير في الـ configuration data أو environment اللي شغال عليها السيستم بتاعنا، عشان كده لازم كل فترة نطلع جديدة من السيستم بأخر الـ updates اللازمه عشان البرنامج يستمر بنفس كفاءة تشغيله اللي عليها، وكمان عشان لو فيه أي bugs طلعها اليوزر حلها.

الـ bugs اللي بيطلعها اليوزر دى ويقول للشركة عليها، أو المشاكل اللي بيكتشفها فريق الدعم أو بتحصل نتيجة لأى تغيير بعد الـ release للبرنامج لما بتتحل بنعمل نوع من أنواع التesting اسمه Maintenance testing .

فالـ Maintenance testing بنعمله لما السيستم يكون شغال ويطلع بجاية أو نحب نعمل update جديد أو upgrade للسيستم بتاعنا أو لأى تعديلات عموما.

التعديلات ع السیسٹم بتشمل:

- Planned enhancement changes (e.g., release-based).
- Corrective and emergency changes.
- Changes of environment, such as:
  - Planned operating system
  - Database upgrades
  - Planned upgrade of COTS (Commercial-Off-The-Shelf) software.
  - Patches to correct newly exposed or discovered vulnerabilities of the operating system.

ال platform maintenance testing (نقل الشغل أو البيانات من migration للثاني)، ولازم يكون فيه operational tests للenvironment الجديدة زى اللي بيحصل للسوق توير اللي هيتنقل بالظبط)، كمان بنحتاج ال migration (conversion) testing لما تجيلى بيانات من application تانى وعايز أدخلها ع السیسٹم عندي.

كمان ال maintenance testing retirement للسیسٹم، ده بيشمل عمل تیست ع data-retention أو ال data archiving لو هتطول فترة ال data migration

out of date maintenance testing ممكن يبقى صعب جداً لو ال specifications بقت أو ضايعة، أو ال testers اللي عندهم domain knowledge بالسیسٹم ده مش موجودين. خد بالك بقى م الكلام الجاي ده لإنه مهم جداً.

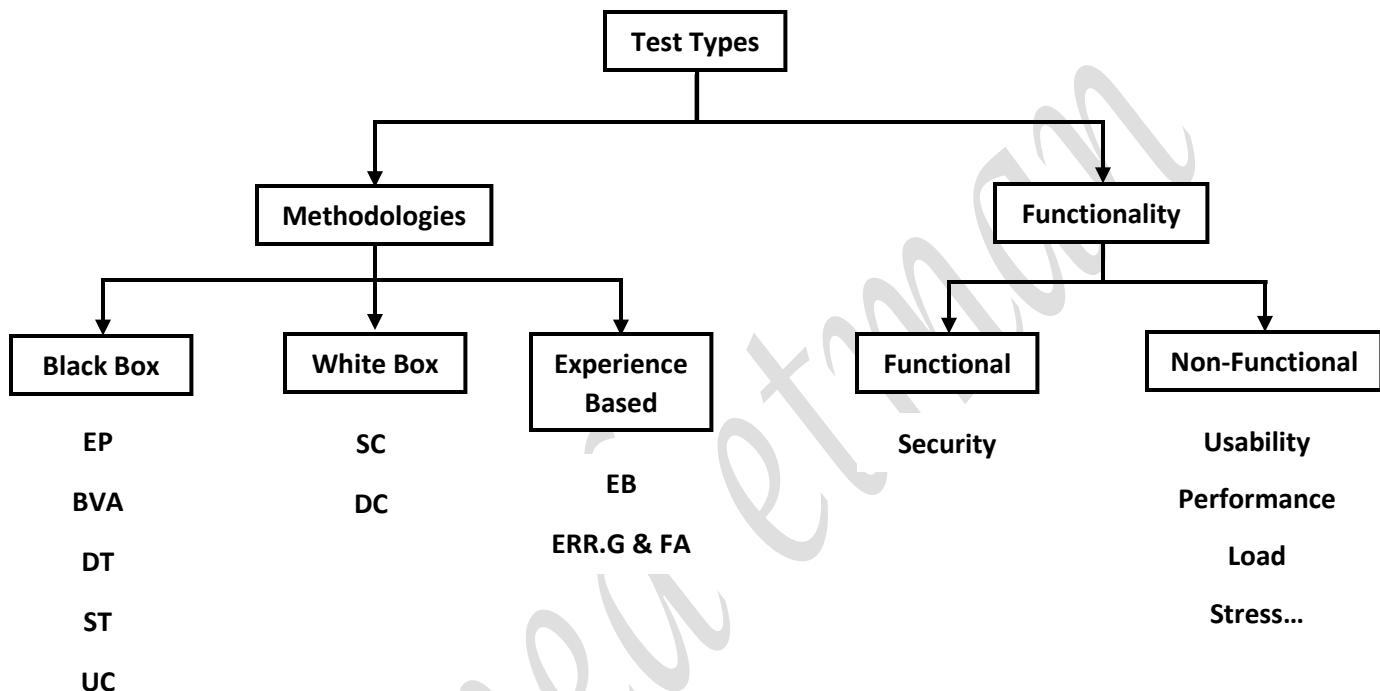
### ال maintenance testing بنعمله على مرحلتين:

- **Re-testing :** وده بنعمله لما بنتيست الحنة اللي فيه المشكلة أو الحاجة اللي عايزين نصلحها.
- **Regression testing :** وده بنعمله للمناطق اللي احتمال انها تتأثر بالتغيير ده ف السیسٹم، وبنحدد المناطق اللي بتتأثر بالتغيير ده عن طريق حاجة اسمها ال impact analysis، ده بيحل وي Shawf تأثير ال module اللي هيتعلن ده مسمى فين تانى ف السیسٹم وبناء على ال regression testing بتحدد ال scope analysis بتاعنا ف ال

## Day 7

### :Test types

التيستينج عندنا ليه أنواع كتير نقدر نصنفهم كالأتنى:



يعنى إيه بقى الكلام ده؟

بص ياسيدى... التيسينج ليه أنواع كتير، الأنواع دى متصنفة حسب ال functionality بتاعة السيستم واللى بنعمل عليها التيسينج، أو حسب ال Methodologies أو الطرق اللي بنعمل بيها التيس، فتعالوا ناخدها واحدة واحدة ونشوف إيه حكاية الاتنين دول بالظبط.

من حيث ال test functionality فيه عندنا نوعين من التيسينج:

### :Functional testing -1

وده بي العمل التيس على ال module بتاعة functions معين أو حتى بتاعة السيستم كله، وده عشان نتأكد إن ال functions بتاعة السيستم مطبوبة وشغالة صح وبتطلعلي outputs صح.

وطبقا لل ISTQB بيعتبروا ال security testing وال interoperability testing نوع من أنواع ال functional testing زى ماقلنا قبل كده فى part 1 من الكتاب، فتعالوا نفكروا نفتقراهم تانى (الكلام اللي جاي من منهج ال ISTQB).

## فنن ضمن أنواع الـfunctional testing

threats: **Security testing** • ي العمل تيست على الـfunctions المرتبطة باكتشاف الـ

(زى الفيروسات) اللي جايالى من بره السيستم زى الـfirewall مثلا.

components: **Interoperability testing** • يقيس قدرة السيستم على التفاعل مع

systems معينة، تقييم التفاعل ده بيشمل:

Customer needs ○

Business needs ○

Technical needs ○

Inputs/outputs format ○

Inputs/outputs availability ○

## :Non-functional testing -2

وده بقى بيختبر الـnon-functional characteristics اللي ف السيستم زى:

processing functions: **Performance Testing** • ده بيتيست معينة خاصة بالـ

time ومعدلات الإنتاج ف السيستم (بيشوف السيستم سريع ف شغله والأداء بتاعه ولا بطئ).

behavior: **Load testing** • يقيس الـbehavior بناع السيستم لما بيتعرض لضغط شغل ع السيستم، يعني

مثلا لو السيستم مصمم إن يشتغل عليه 100 users فى نفس الوقت التيست هنا مهمته يقيس

الـbehavior بناع السيستم والـ100 يوزر دول شغالين مع بعض ف نفس الوقت ع السيستم

ويشوف السيستم سريع ولا بطئ وبيهنج ولا وفية مشاكل ولا لا... كده يعني.

load testing: **Stress testing** • هو نفس الـ load testing لكنى بزود فى الحمل أكثر من المطلوب ف

الـ requirements وأشوف السيستم هيستحمل لغاية فين، يعني لو السيستم مصمم إن يشتغل

عليه 100 يوزر ف نفس الوقت ببتدى أديله 105 يوزر شغالين مع بعض ف نفس الوقت وأشوفه

بيتصرف إزاى، ولو السيستم مثلا مصمم إنه يشتغل على خط نت 1 ميجا أبتدى أقل من سرعة

الـنت وأشوفه هيعامل إزاى وهيستحمل لغاية فين عقبال مايحصله crash.

software product: **Usability testing** • يحدد إلى أى درجة الـ

تعليمي للـuser اللي هيستخدمه وسهل ف الشغل ولا وجداب للـuser فى ظروف معينة ولا لا.

Maintainability testing: **Maintainability testing** • ده بيقيم مدى استعداد السيستم للتعديل أو لتصحيح

defects أو لتحقيق requirements جديدة أو للصيانة عموما.

- **Reliability testing**: يقيس قدرة software product لعمل ال functions المطلوبة في ظروف معينة لفترة معينة من الوقت أو لعدد معين من ال operations (بشفوف السيستم أقدر أثق في تشغيله وإنه مش هيقع مني كل شوية ولا لا).
- **Portability testing**: يقيس مدى سهولة ومرنة ال software product في الشغل لما ينتقل من hardware/software environment لـ environment تانية سواء هاردوير أو سوفتوير.

أما من ناحية ال methodologies فالtesting له 3 أنواع:

### :Black box testing -1

ال black box testing هو عمل تيست ع السيستم – سواء functional أو non-functional – من غير ما يكون عارف ال internal structure أو الكود بتاع ال component أو السيستم، فالسيستم بالنسبة عامل زى الصندوق الأسود مش عارف إيه اللي جواه.

ال black box testing معروف كمان بإسم specification based testing، وده لأن الأساس اللي بعمل عليه التesting هو ال requirements اللي طلبه العميل، وال test case تكون actual result = expected result pass بس فى حالة إن ال

### :5 techniques ليه black box testing

Equivalence Partitioning - Boundary Value Analysis - Decision Tables - State Transition - Use Case.

### :White box testing -2

ال white box testing بعمل التيست بتاعي على أساس ال internal structure أو الكود اللي يكون شايفه قدامى سواء الكلام ده لـ component أو لسيستم معين، عشان كده معروف برضه بإسم glass box testing أو structure based testing.

غالبا بيسخدم ال component testing فى white box testing وال integration testing.

### :2 techniques ليه white box testing

ال Statement Coverage و Decision Coverage

وهنتكلم أكثر عن ال Black-box & White-box test design techniques كمان شوية.

### :Experience based testing -3

هو نوع من أنواع الـ **testing** قائم أساساً على مهارة وخبرة وحدس التيسير في التعرف على **bugs** أو **defects**، التكنيك ده بعتمد عليه لما يبقى عندي **Specifications** ناقصة أو مش كاملة أو حتى لو ما عنديش **specifications** خالص أو ضایعه، فبيتدى هنا أعتمد على خبرة التيسير وحدسه في البرامج المشابهة اللي اشتغل عليها قبل كده، والكلام ده بيتم في خلال **time-boxes** عشان مش هنفضل نعمل تيست لآخر العمر طبعاً، ده غير إنى يكون مرتبط بوقت للتسليم، ففي الحالات دى التكنيك ده مناسب جداً.

فيه 2 في الـ **experience based testing techniques** هنتعرف عليهم دلوقتى.

### :Experience based testing techniques

#### :Error Guessing & Fault attack -1

ده **technique** بيعتمد على إن التيسير بيتوقع **errors** معينة في أماكن معينة (**error guessing**) فبieroح علطول ينفذ التيسير بتاعه عشان يطلع الـ **(fault attack)**.

#### :Exploratory testing -2

ف الـ **technique** ده التيسير مابيكونش معاه أي **requirements** أو بتبقى ناقصة، فيكتشف السيستم حسب خبرته بالبرامج المشابهة وهو ورزقه بقى.

**:Summary**

الجدول ده هيلخص الكلمتين اللي فاتوا دول:

Black-box testing	White-box testing	Experience-based testing
Based on requirements	Based on code and internal structure of the system	Based on the knowledge of the tester
Tests are created from the requirements	Tests are created from the code	Tests are created from the tester's experience
Also called "Specification-based testing"	Also called "Structure-based testing" or "Glass-box testing"	-
Black-Box techniques: <ul style="list-style-type: none"> <li>• Equivalence Partitioning</li> <li>• Boundary Value Analysis</li> <li>• Decision Table</li> <li>• State Transition</li> <li>• Use Case testing</li> </ul>	White-box techniques: <ul style="list-style-type: none"> <li>• Statement Coverage</li> <li>• Decision Coverage</li> </ul>	Experience-based techniques: <ul style="list-style-type: none"> <li>• Error guessing &amp; fault attack</li> <li>• Exploratory testing</li> </ul>

## Day 8

### :Black box test design techniques

قلنا قبل كده إن ال Black box مافيهاش أى معرفة بالكود أو Specification based الـ structure بتاعة السيسن، يعني م الأخر بدخل inputs وأطلع outputs بس وماليش دعوة باللى بيحصل جوه الكود، الكلام ده عشان يتم بنعمله بالـ 5 techniques اللي قلنا عليهم قبل كده، فتعالوا نشوفهم بالتفصيل.

#### :Equivalence Partitioning (EP) -1

التكنيك technique ده بستخدمه فى حالة إن عندى مجموعات مختلفة من ال inputs، فكرة Equivalence Partitioning باختصار قايمه على تقسيم ال inputs اللي عندى لمجموعات، كل مجموعة منهم بستنى منها output واحد، وبختار قيمة واحدة بس من كل مجموعة عشان تمثل المجموعة دى ف التيسن وأعمل عليها التيسن بتاعى.

**مثال:** لو عندى textbox بيستقبل الأرقام من 1 ل 100 بس، فى الحالة دى هنقسم ال inputs عندى لـ 3 مجموعات زى كده:



كده عندى 3 مجموعات، الأولى هي ال range بتاعى من 1 ل 100 والمجموعة الثانية هي الأرقام اللي أصغر من ال 1 والمجموعة الثالثة هي الأرقام اللي أكبر من ال 100.

فهناخد أى رقم من كل مجموعة يمثلها ونعمل عليه ال test case بتاعتنا، يعني هنا ممكن ناخذ 7 و 24 و 104 في التيسن بتاعنا ونجرب ندخلهم على ال textbox ونشوف هيقبلاهم ولا لا.

لكن هل الطريقة دى مجديّة؟

تعالوا نشوف من غيرها إيه اللي هيحصل....

لو جه تيستر مش محترف و هي عمل تيسن ع الكلام ده، فهيتست بكل قيمة لوحدها من أول ال 0 لغاية ال 101 فيبقى عندى 102 test case بـ 102 input.

بس في ال execution فييقى كده هنفذ التيست للجزء الصغير ده بس في 102 دقيقة (ساعة و42 دقيقة)، لكن باستخدام ال EP عملنا 3 لو كل واحدة انتفذت ف دقيقة هتاخذ معانا 3 دقائق بس مع تحقيق نفس النتائج، وبالتالي أحسن.

الطريقة دي كويصة جدا... لكن إيه ظروف ال 1 وال 100 ف الليلة دي؟ بمعنى تانى هل هما جوه ال range ولا براه؟ دي نقطة ضعف في ال Equivalence Partitioning.

عشان كده طلع تكتيك تانى أفضل شوية اسمه Boundary Value Analysis (BVA) ... تعالوا نشوفه.

### :Boundary Value Analysis (BVA) -2

ال BVA هو نفس ال EP لكن الفرق إن ال BVA بيختار القيم اللي على حدود كل مجموعة من ال inputs، لأن القيم دي بتكون فرصه حدوث bugs فيهم أكبر من أي قيمة تانية، فمثلا لو عندنا نفس المثال اللي فات بتاع ال textbox اللي بيستقبل أرقام من 1 ل 100 وقسمنا ال 3 inputs إلى مجموعات.



فهنا هناخد القيم اللي على حدود كل مجموعة، يعني هناخد ال 0 وال 1 وال 2 وع الناحية الثانية هناخد ال 99 وال 100 وال 101 فه تكون ال inputs بالشكل ده:



القيم دي مثلت كل مجموعة منمجموعات ال inputs وكمان اتأكدت عن طريقها إن ال 1 وال 100 معايا ف ال range، وبالتالي ضربت عصفورين بحجر.

لكن طالما ال 1 وال 100 كده معايا في ال range وبيمثلوا ال range بقى فكده هيتحققوا نفس النتيجة اللي هيتحققها ال 2 وال 99، عشان كده مالوش لازمة إنى أخذ ال 2 وال 99 معايا ونكتفى بس بال 0

وال 1 وال 100 وال 101 في ال BVA، وبكده أكون حققت ال best practice إنى حفقت النتائج المطلوبة وقللت عدد ال test cases من 6 لـ 4 بس.

### :Decision Table -3

التكنيك ده عباره عن جدول بنجمع فيه كل الاحتمالات لكل ال inputs وال outputs بتابعتها، وعادة بنستخدمه في ال business rules المعقدة أو فيها شروط كتير مرتبطة بعضها، فيكون الجدول شكله عامل زى كده مثلا:

	Test 1	Test 2	Test 3
Input 1	T	T	F
Input 2	T	T	F
Input 3	T	- (don't care)	F
Input 4	T	F	T
Output 1	Y	Y	N
Output 2	Y	N	Y
Output 3	N	Y	N

### ملاحظات:

- ال - دى بتعبّر عن حالة ال input يعني أيّا كان ال input اللي ف الحالة دى مش هيفرق معايا.
- عادة ال  $N = No$   $F = False$  وال  $Y = Yes$  وال  $T = True$ .

أبسط مثال على ال Decision Table هو ال login:

مثال: لو قلنا اننا عايزين نعمل تيست على login page، فالمفروض إنى لو دخلت ال username وال password صح يسمح بال login غير كده مش هيسمح.

فعندنا هنا احتمالين لل User Name وكذا لل Password:

User Name <True  
False } 2 & Password <True  
False } 2

فعشان نحسب عدد الـ **test cases** أو الـ **combinations** اللي ه تكون ف الـ **decision table** فيه طریقین:

1- هنضرب احتمالات كل واحد فيهم ف بعض، يعني إنى هنضرب احتمالات الـ **User Name** فى احتمالات الـ **Password** يعني هيكون:

$$\text{Test cases} = 2 \text{ (User Name)} * 2 \text{ (Password)} = 4$$

2- أو هيكون العدد بتاع الاحتمالات ألس العدد بتاع التكرار بتاعهم، ف تكون:

$$\text{Test cases} = (\text{True} \& \text{False})^{(\text{User Name} + \text{Password})} = 2^2 = 4$$

المهم... هنستخدم أي طريقة منهم، ونحدد الجدول بتاعنا، فه يكون بالطريقة دى:

	1	2	3	4
User Name				
Password				
Login?				

نيجي بعد كده نشوف هنجمع الـ **true** والـ **false** ف الجدول ده إزاى، فيبيولك إن أول صف هنшوف عدد الأعمدة أو الـ **test cases** اللي ف الجدول ونقسمهم على عدد الاحتمالات بتاعة الصف الأولاني، يعني هيكون الكلام كده:

$$\text{Test Cases No (Columns no.) / User Name Probs.} = 4 / 2 = 2$$

فهنهعمل في كل عمود قصاد الـ **user name** اتنين **True** واثنين **False**، فيكون الجدول شكله عامل كده:

	1	2	3	4
User Name	T	T	F	F
Password				
Login?				

نيجي بقى للباسورد.... هيكون بنفس الطريقة لكن بدل ما هيكون عدد الأعمدة كله هيكون عدد الأعمدة ف الاحتمال الواحد، وده عشان نعمل أكبر قدر ممكن من الـ **combination** مابين كل الـ **inputs**، يعني أصح هنقسم أعمدة الـ **username** ف الـ **True** على احتمالات الباسورد، ف تكون زي كده:

$$\text{User Name Probability (True or False) / Password Probabilities} = 2 / 2 = 1$$

فيبقى شكل الجدول كده:

	1	2	3	4
User Name	T	T	F	F
Password	T	F		
Login?				

طب الـ *test cases* رقم 3 و 4 ظروفهم إيه؟

طالماوصلنا لنهاية الاحتمالات وماوصلناش لنهاية الجدول فهنكرر الاحتمالات بنفس الـ *sequence* لغاية مانغطي كل الـ *test cases* ف الجدول، فيكون شكل الجدول كده:

	1	2	3	4
User Name	T	T	F	F
Password	T	F	T	F
Login?				

تعالوا بقى نشوف الـ *outputs* بتاعتنا:

هو قال م الأول إن لو الـ User Name صح هيسمح بالـ login، غير كده مش هيسمح، ففي الجدول بتاعنا:

- Test Case 1: User Name = True & Password = True .... Login = Yes.
- Test Case 2: User Name = True & Password = False .... Login = No.
- Test Case 3: User Name = False & Password = True .... Login = No.
- Test Case 4: User Name = False & Password = False .... Login = No.

فيكون الجدول عندنا شكله كده:

	1	2	3	4
User Name	T	T	F	F
Password	T	F	T	F
Login?	Y	N	N	N

بعد كده نحاول اختصار **test cases** على قد مانقدر بحيث حقق الـ **optimum solution** بمعنى أعلى نتيجة ممكنة بأقل مجهد ممكن.

بس إزاي اختصر ف الكلام ده؟

**أقولك ياسيدى... القاعدة بتقول عشان اختصار **test cases** لازم تحقق الشروط دى:**



- 1- الاختصار هيكون بين **2 test cases** بس ف المرة الواحدة.
- 2- لازم تكون الـ **outputs** واحدة فى الـ **2 test cases** (الاتنين ليهم نفس النتائج بالظبط).
- 3- لازم يكون **input** واحد بس مختلف بين الـ **2 test cases**.
- 4- بعد مانختصر ونضم الـ **2 test cases** على بعض الـ **input** المختلف ده هيبقى **don't care** وبنعتبر عنه بالـ **-**.

فتعالوا نشوف إيه الأعمدة اللي تتطبق عليها الشروط دى ف الجدول:

	1	2	3	4
User Name	T	T	F	F
Password	T	F	T	F
Login?	Y	N	N	N

- لو شفنا العمود رقم 1 مع العمود رقم 2 .... اليوزر نيم واحد والباسورد مختلف، لكن الـ **output** مش واحد فمش هينفع معانا.
- رقم 2 و3 ... الـ **inputs** مختلفة ف اليوزر نيم والباسورد فماينفعش.
- رقم 3 و4 ... اليوزر نيم واحد والباسورد مختلف والـ **output** زى بعضه... فدول ينفعوا، فنحط عليهم مستطيل عشان نحددهم مع بعض.

	1	2	3	4
User Name	T	T	F	F
Password	T	F	T	F
Login?	Y	N	N	N

دلوقتى هنضم 3 على 4 فى **test case** واحدة بس، فهيبقى الـ **user name** زى ما هو ف الاتنين والباسورد هيكون (-) والـ **login don't care** (Z) زى ما هو، فيبقى شكل الجدول كده:

	1	2	3
User Name	T	T	F
Password	T	F	-
Login?	Y	N	N

تعالوا كده نشوف المنطق ف اللي عملناه ده... هل لو الـ user name دخل غلط أصلا هليفرق مع السيسن إذا كان الباسورد صح ولا لا؟ أكيد مش هيفرق ومش هيسمح بالـ login، وبالتالي الشغل اللي عملناه ده صح 100%... بعد كل اختصار لازم نسأل نفسنا السؤال ده.

نيجي بعد كده نختصر تانى... قولنا قبل كده إن 1 و 2 ماینفعوش مع بعض لإختلاف الـ outputs، و 2 و 3 كمان ماینفعوش عشان الـ inputs مش واحدة، فيكون ده الشكل النهائي للـ login test cases المعمولة بالـ decision table واللى هتنفذ.

حد هيقولى إن الدنيا بسيطة وماكانتش مستاهلة كل الهرى ده... أقوله لا الكلام ده مهم جدا وهتبان أهميته أكثر لما الـ combinations بين الـ inputs والـ outputs تكون أكثر من كده، زى المثال الجاي مثلا.

**مثال:** لو فلنا ف المثال اللي فات إن احتمالات الـ user name والـ password هيبي معاهم كمان الـ empty (يعنى الـ user name بسipp القيمة فاضية مايدخلش حاجة)، فهنا الكلام هيختلف.



تعالوا نشوف الـ test cases بتاعتتنا، فهنحسبهم زى المرة اللي فاتت، فيكون:

$$\text{Test cases} = 3^2 = 3 \times 3 = 9$$

فيكون الجدول عندي شكله عامل كده:

	1	2	3	4	5	6	7	8	9
User Name									
Password									
Login									

فتعالوا المرة دي بقى نشوف هنقسم الـ inputs ازاي

عندي الجدول بـ 9 اعمدة هيتقسموا على 3 احتمالات لـ user name، فيبقى 3 لكل احتمال، فيكون الجدول كده:

	1	2	3	4	5	6	7	8	9
User Name	T	T	T	F	F	F	E	E	E
Password									
Login									

نيجي بعد كده للباسورد، ف هنا عندنا برضه 3 احتمالات هيتقسموا على الـ 3 احتمالات بتوع الـ user name، فهنا خط احتمال واحد مع كل احتمال user name مع التكرار طبعاً، ف تكون الدنيا عاملة كده:

	1	2	3	4	5	6	7	8	9
User Name	T	T	T	F	F	F	E	E	E
Password	T	F	E	T	F	E	T	F	E
Login									

نيجي بعد كده نشوف الـ outputs بتاعتنا، اللي هي عبارة عن إن الـ username والباسورد صح هيسمح بالـ login، غير كده مش هيسمح، فيكون الجدول شكله عامل كده:

	1	2	3	4	5	6	7	8	9
User Name	T	T	T	F	F	F	E	E	E
Password	T	F	E	T	F	E	T	F	E
Login	Y	N	N	N	N	N	N	N	N

نيجي بقى لمرحلة الاختصارات، طبعاً فهمنا القاعدة ماشية إزاي، وبعد تطبيق القاعدة هنكون وصلنا للتجمیعات دى:

	1	2	3	4	5	6	7	8	9
User Name	T	T	T	F	F	F	E	E	E
Password	T	F	E	T	F	E	T	F	E
Login	Y	N	N	N	N	N	N	N	N

فهنضم 2 على 3 و 4 على 5 و 7 على 8، فيكون الجدول عامل كده بعد الاختصارات:

	1	2	3	4	5	6
User Name	T	T	F	F	E	E
Password	T	-	-	E	-	E
Login	Y	N	N	N	N	N

ملحوظة: في العمود رقم 2 المقصود بيها **Empty** أو **False** (أى حاجة غير الـT).  
ونعيد الاختصار تانى، فيكون:

	1	2	3	4	5	6
User Name	T	T	F	F	E	E
Password	T	-	-	E	-	E
Login	Y	N	N	N	N	N

ملحوظة: الـ**don't care** ماینفعش نضمها على **don't care** تانية إلا لو كل الـ**inputs** والـ**outputs** زى بعضها بالظبط... يعني ماینفعش نضم 2 و 3 عشان الـ**user name** بتاعهم مختلفين، ولا ينفع نضم الـ**don't care** على أى **option** تانى زى 3 و 4 و 5 و 6، لكن ممكن نضم 4 و 6 على بعض، فيكون الجدول بعد التعديل شكله عامل كده:

	1	2	3	4	5
User Name	T	T	F	-	E
Password	T	-	-	E	-
Login	Y	N	N	N	N

تعالوا كده نحسبها بالمنطق:

- في رقم 2 حتى لو دخلت الـ user name صح لكن دخلت الباسورد فاضى أو مدخلتوش خالص مش هيسمحى بالدخول.
- في رقم 3 لو دخلت الـ user name غلط مش هيفرق معاه إذا كان الباسورد صح ولا لا أو حتى فاضى خالص.... كده كده مش هيسمح بالدخول.
- في رقم 4 لو سببت الـ Password فاضى مش هيفرق معاه إذا كان الـ User Name صح ولا لا أو حتى فاضى خالص.... كده كده مش هيسمح بالدخول برضه.
- في رقم 5 لو سببت الـ User Name فاضى مش هيسمحى بالدخول أيا كانت قيمة الـ Password سواء كانت صح أو غلط أو فاضية.

**ملحوظة:** الكلام ده قشطة أوى لو مش هيسمح بالدخول وخلاص، أما لو بيجيب رسالة لكل واحد فيهم إن ده غلط أو ماينفعش يتساب فاضى ففى الحالة دى لازم نزود الكلام ده ف الـ output زى المثال ده كده:

لو قلنا إن فيه نظام تأمين صحي معين بيقبل الناس اللي فيهم الشروط دى:

- يكون أكبر من 50 سنة
- مايكونش مدخن

لكنه (بالإضافة للشروط اللي فاتت) بيعمل خصم 10% للناس اللي:

- females
  - اللي عندها مشاكل ف القلب
- ففي الحالة دى هنشتغل كالأتنى:



$$\text{Test cases} = 2^4 = 2 \times 2 \times 2 \times 2 = 16$$

فهيكون الجدول كده:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
>50	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Smoker?	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
Gender	M	M	F	F	M	M	F	F	M	M	F	F	M	M	F	F
Heart problems?	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	f
Insurance?	N	N	N	N	Y	Y	Y	Y	N	N	N	N	N	N	N	N
10% discount?	N	N	N	N	N	N	Y	N	N	N	N	N	N	N	N	N

نعمل اختصار اتنا الحلوة فتبقى كده:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
>50	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
Smoker?	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
Gender	M	M	F	F	M	M	F	F	M	M	F	F	M	M	F	F
Heart problems?	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Insurance?	N	N	N	N	Y	Y	Y	Y	N	N	N	N	N	N	N	N
10% discount?	N	N	N	N	N	N	Y	N	N	N	N	N	N	N	N	N

وبعد الاختصار بقى ده الجدول:

	1	2	3	4	5	6	7	8	9
>50	T	T	T	T	T	F	F	F	F
Smoker?	T	T	F	F	F	T	T	F	F
Gender	M	F	M	F	F	M	F	M	F
Heart problems?	-	-	-	T	F	-	-	-	-
Insurance?	N	N	Y	Y	Y	N	N	N	N
10% discount?	N	N	N	Y	N	N	N	N	N

نعمل اختصار أكثر في طلع الجدول كده:

	1	2	3	4	5	6
>50	T	T	T	T	F	F
Smoker?	T	F	F	F	T	F
Gender	-	M	F	F	-	-
Heart problems?	-	-	T	F	-	-
Insurance?	N	Y	Y	Y	N	N
10% discount?	N	N	Y	N	N	N

ونعمل اختصار أكثر في طلع الجدول كده:

	1	2	3	4	5
>50	T	T	T	T	F
Smoker?	T	F	F	F	-
Gender	-	M	F	F	-
Heart problems?	-	-	T	F	-
Insurance?	N	Y	Y	Y	N
10% discount?	N	N	Y	N	N

وبكده وصلنا لأفضل نتيجة بأقل مجهود ممكن

#### :State Transition -4

ال state transition: ده انتقال من حالة في النظام لحالة تانية، زي مثلا إشارات المرور... ماينفعش أروح للأحمر من الأخضر إلا لما أعدى ع الأصفر مثلا، وكذلك الأحمر ماينفعش أروح منه للأخضر إلا لما أعدى ع الأصفر.

black box test: هو State Transition testing فال design technique فيه بنصمم test cases عشان تنفذ



كل الـ state transitions سواء الـ valid أو الـ invalid ممكن يتعرف برضه باسم N-switch testing

### عادة الـ state transition testing بيسخدم في عمل التيس ع الحاجات دى:

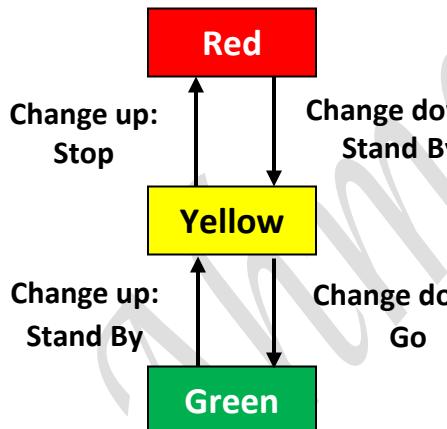
- Screen dialogues
- Website transitions

### أثناء استخدامنا للـ state transition testing بنستخدم الكلام ده:

diagram ده بنعبر فيه عن الـ states اللي ممكن السيس تم يتنقل بينهم، وكمان بيكون فيه الـ events اللي بتحصل أثناء النقلات دى.

State table -2: ده جدول بنوضح فيه النقلات دى ونتائجها لكل state، الجدول فيه طبعا الـ valid transitions والـ invalid transitions.

مثال: تعالوا نشوف المثال بناع إشارات المرور، فقولنا إن الأحمر مابيروحش للأخضر إلا لما يروح للأصفر، وكذلك الأخضر مابيروحش للأحمر إلا لما بيعدى عن الأصفر، فنقدر نعبر عنها بالشكل ده:



	Change Up	Change Down
Red	Null	Stand By / Yellow
Yellow	Stop / Red	Go / Green
Green	Stand By / Yellow	Null

State table

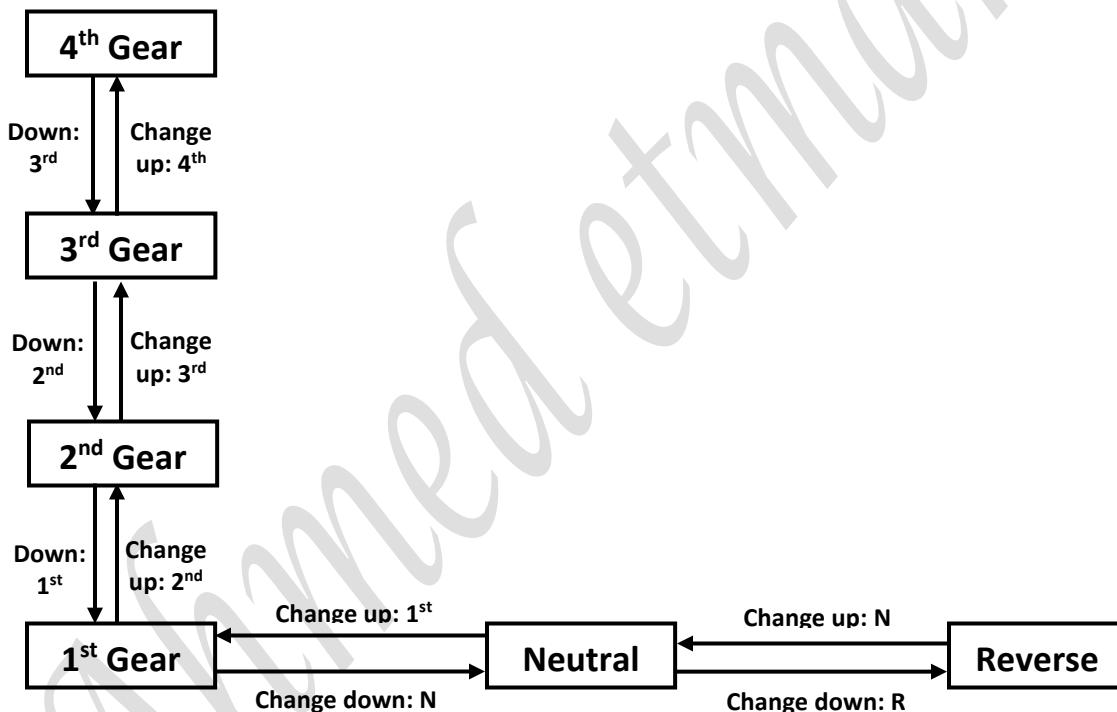
State transition diagram

**مثال تانى:** تعالوا نشوف الفيتيس *vitesse* بتاع العربية.



لو قلنا إن الفيتيس بتاع العربية فيه 4 حارات وحارة Reverse عشان لما العربية ترجع لورا، فطبعاً وانا راكن تكون فاكم الغيار Neutral (اللى هو الخط اللي ف نص الصورة بين الأرقام، وعشان أتحرك لازم أروح للأولاني لو هطلع لقدم أو أحرك الفيتيس ناحية ال R لو هرجع لورا، وعشان أزود السرعة لازم أطلع من الحارة الأولى للثانية والثالثة والرابعة بالترتيب، بمعنى إنى مانفعش أطلع من الأول للثالث علطول لازم أعدى ع الثاني الأول وهكذا مع كل الحارات.

فلو هنعمل الكلام ده بال state transition هيكون بالشكل ده:



فنيجي نعمل الـ state table فيكون زى كده:

	<b>Change up</b>	<b>Change down</b>
<b>Reverse</b>	Neutral	Null
<b>Neutral</b>	<b>1<sup>st</sup> Gear</b>	<b>Reverse</b>
<b>1<sup>st</sup> Gear</b>	<b>2<sup>nd</sup> Gear</b>	<b>Neutral</b>
<b>2<sup>nd</sup> Gear</b>	<b>3<sup>rd</sup> Gear</b>	<b>1<sup>st</sup> Gear</b>
<b>3<sup>rd</sup> Gear</b>	<b>4<sup>th</sup> Gear</b>	<b>2<sup>nd</sup> Gear</b>
<b>4<sup>th</sup> Gear</b>	Null	<b>3<sup>rd</sup> Gear</b>

### :Use Case Testing -5

الـ Use Case (UC) هى عبارة عن document بخصوص الـ requirements بشكل process flow دى، والكلام ده كله بيتمثل على هيئة User functional system testing أو سيناريو معين، عادة الـ use case بتكون مفيدة فى الـ Acceptance Testing (UAT) وكمان فى الـ integration interfaces وكمان فى الـ Acceptance Testing (UAT) المختلفة.

#### الـ Use Case بت تكون من الحاجات دى:

- دول الـ users بتنوع السيستم: **Actors**.
- الشروط اللي لازم تكون موجودة قبل مانشتغل ع الـ UC دى: **Pre conditions**.
- الحالة اللي هيكون عليها السيستم لما نخلص تنفيذ الـ UC دى (الـ outputs **Post conditions** • يعني).
- ده بنعبر فيه عن الـ Actors والعمليات اللي بيعملوها بالتفصيل: **UC diagram**.

الأمثلة الجاية هتووضح الدنيا أكثر:

#### :Example 1

لو عندي سيستم Bank ATM، فالسيستم ده الـ requirements بتاعته إن اليوزر يقدر يشيك ع الحساب بقاعة ويعمل عمليات السحب والإيداع والتحويل من المكنة دى، وفيه موظف مختص بالمكنة دى بيعملها صيانة دورية زى مثلا انه يحطلها فلوس ويصلحها لو فيها أى مشاكل.

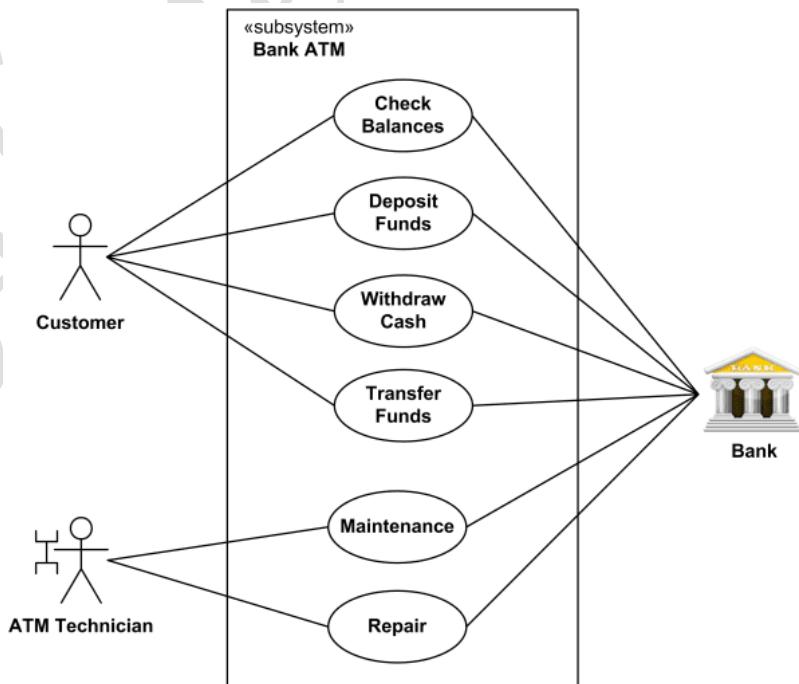
لو جينا نحل الكلام ده عشان نعمل الـ **UC** بتاعتنا هنطلع بالنتيجة دى:

- **Actors:** Customer - ATM Technician.
- **Processes:** check balance - deposit funds - withdraw cash - transfer funds - maintenance (e.g., money update) - repair.
- **Precondition:** Customer must have an account (عشان يشتعل ع السيسن).
- **Postcondition:** due to transactions, balance increased or decreased.

تعالوا بقى نرسم الـ **diagram** بتاعنا، واحنا بنرسم الـ **diagram** لازم نراعى القواعد دى:

- الـ **Actors** بيتعبر عنهم بأشكال أشخاص.
- لوفيhe **Actors** متفرعين من الـ **Actor** الأصلى ومرتبطين ببعض ممكن يتوصل بينهم بسهم خطوطه متقطعة (--->)، والسهم ده بيشارور على الـ **Actor** الأصلى.
- العمليات بيتعبر عنها بشكل بيضاوى، وبيكتب جواها عنوان مختصر جداً للعملية
- بيتم الربط بخطوط بين الـ **actors** والعمليات اللي بيعملوها
- لو فيه عمليات فرعية من العمليات الرئيسية بتتحط برضاها بشكل بيضاوى وبرضاها بتترتبط بالعملية الأصلية بسهم خطوطه متقطعة.
- ممكن برضاها يكتب وصف مختصر على الخط للـ **Action** اللي بياخده الـ **Actor** فى العملية دى (بس مش لازم).

فلو طبقنا القواعد دى على المثال بتاعنا هيطلع الـ **Use Case Diagram** بالشكل ده:

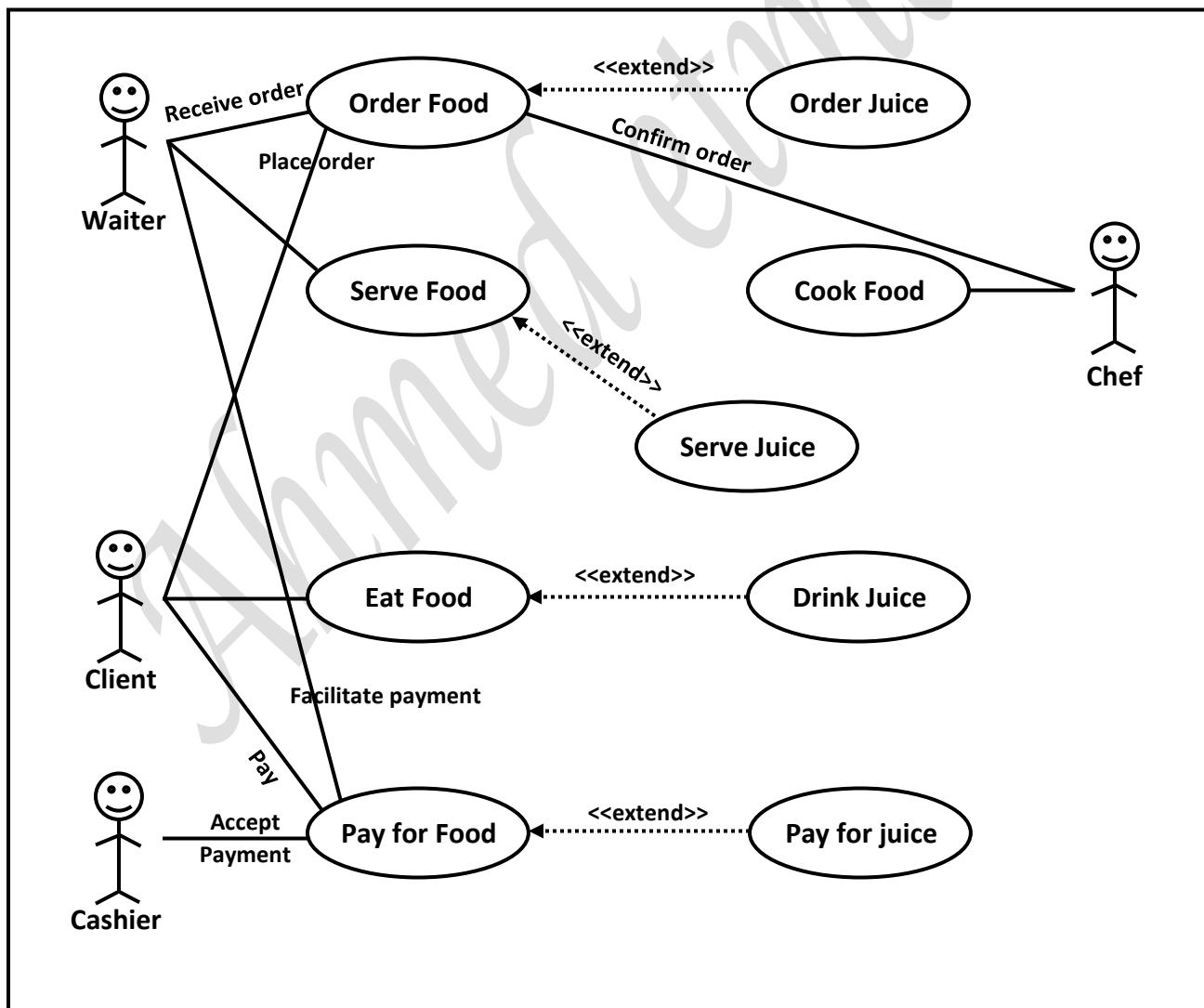


### :Example 2

لو قلنا عندنا مطعم فيه:

- **الـclient:** بيطلب الأكل أو عصير - بياكل أو يشرب الحاجة اللي طلبها - بيدفع تمن الحاجة اللي طلبها.
- **الـwaiter:** بياخد الطلبات م الزباين - بيقدم الأكل أو العصير المطلوب للزبون - بيستقبل الفلوس من العميل.
- **الـchef:** بياخد الطلبات من الـwaiter - بيطبخ أو يجهز الأكل أو العصير المطلوب.
- **الـcashier:** بيستقبل فلوس الأوردرات سواء من الـclient أو من الـwaiter.

فيكون الـUC diagram عندنا شكله كده:



**Note:** أيا كان الـ **technique** اللي بتسخدمه حاول دايما تشف كل الحالات لكل **field** ف السيستم بتاعك ع الأقل مرة واحدة... يعني مثلا لو عندك **textbox** فى **false** و **true** حاول ع الأقل تشف كل احتمال منهم مرة واحدة ع الأقل.

### :White box techniques

أما بقى الـ **white box** أو الـ **structure based** فهو مبني أساسا على المعرفة الكاملة بالكود والـ **structure** بتاعة البرنامج وبشتغل ع الأساس ده.

قلنا قبل كده إن الـ **white box** ليه 2 **techniques** تعالوا نشوفهم.

### :Statement Coverage -1

الـ **technique** ده فكرته إنك لازم تتعدي على كل سطر كود وتنفذه ع الأقل مرة واحدة، والمعيار اللي على أساسه بحدد نسبة الـ **statement (code) coverage** هو القانون ده:

$$\text{Coverage measurement} = \frac{\text{Number of statements executed} * 100}{\text{Total number of statements}}$$

### :Example

عندنا برنامج بيستقبل عمر الشخص، لو أكبر من 20 هيكتبه إنه أكبر من 20 سنة، ولو أكبر من 30 هيكتبه إنه أكبر من 30، فالكود هيكون شكله كده (ده كود C#):

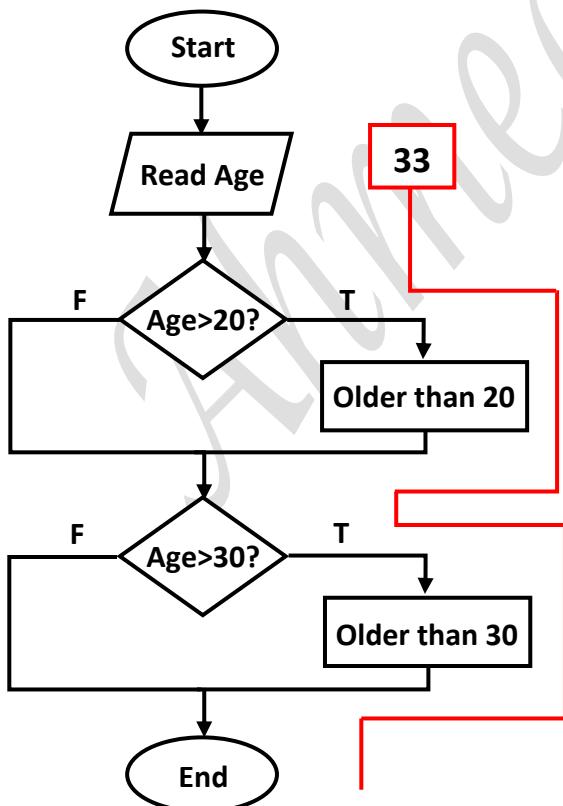
```
static void Main(string[] args)
{
    Console.WriteLine("Please enter your age");
    int age = int.Parse(Console.ReadLine());
    if (age > 20)
        Console.WriteLine("You're older than 20 years old");
    if (age > 30)
        Console.WriteLine("You're older than 30 years old");
    Console.ReadKey();
}
```

لما يكون عندك كود زى ده وتحب تعمله تىست ماتدخلش تعمل تىست علطول كده لا إستنى لأن الموضوع ممكن يلغبط... وياما ناس وقعت فيه سواء ف امتحانات ال ISTQB أو في شغل فعلى، الموضوع ده ليه أصول وخطوات لازم تتعلم وتركيب عشان تضمن انك بتعمل أفضل نتيجة بأقل مجهد.

### الخطوات دى عباره عن:

- 1- ارسم Flowchart للكود وركز أولى عشان الرسمة دى مهمة جدا وبيتبني عليها كل اللي بعدها.
- 2- وانت بتحدد ال inputs حاول على قد ماقدر تدخل inputs تنفع معأغلب السطور اللي قدامك لو ماكانتش كلها.
- 3- بعد ما تحدد ال inputs امشى بالقلم على ال input شوف ال flowchart ده هيوصلك لفين، ولو مش هيوصلك للأخر ياندخل قيمة تانية توصلك لأخر الكود ياتعمل input test case بـ تانية تكميل ع الكلام ده وتوصلك لأخر الكود.
- 4- وانت بتعمل statement coverage لازم تتأكد إن ال test cases بتاعتك مغطية كل سطر ف الكود ع الأقل مرة واحدة.  
فتعالوا نطبق الكلام ده ع الكود بتاعنا:

- 1- نرسم ال flowchart (لاحظ إن ال if بتاع ال 30 مش تحت ال if بتاع ال 20).



- 2- نشوف ال input اللي ممكن يغطي الكود كله، فنقول قيمة زى 33 مثلا.

- 3- ال 33 دى نمشيها بالقلم (اعتبره الخط الأحمر)، فلما نمشيها ع ال flowchart هنلاقى أنها تقدر تغطي كل سطور الكود (أكبر من ال 20 وال 30).

- 4- لو بصينا ع الكود هنلاقى إن ال input بتاعنا مغطى كل سطور الكود اللي عندنا فمافيش أى داعى ل inputs تانية.

تعالوا نحسب نسبة statement coverage لل input اللي دخلناه ده:

عندنا هنا ف الكود 8 سطور لما دخلنا ال 33 عدى عليهم كلهم وانتفدو، فبكله يكون:

$$\text{Code Coverage} = (8 * 100) / 8 = 100\%$$

الکلام ده کویس جدا... لکن طبقا لل technique ده ماینفعش ادخل ای inputs تعددی علی حالات ال false لإن ال ما فيهوش ای سطور کود سواء ف الشرط الأولانی او الثاني، وبما إن لازم ننتیس کل حاجة ف البرنامج ع الأقل مرة واحدة، فالتكنیک ده مش دقیق اوی.

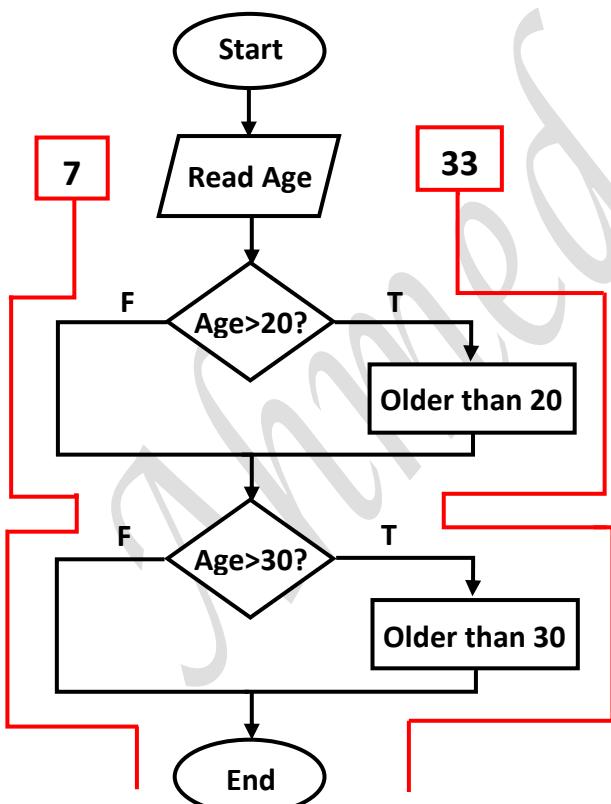
المشكلة دی تم حلها في Decision Coverage.

### Decision Coverage -2

التكنیک ده هو نفس ال statement coverage بالضبط، لكن الفرق إن هنا بشتغل على ال decisions وبغطيهم سواء في حالات ال true أو ال false حتى لو ما فيهش کود في واحد منهم، ومقیاس ال Decision coverage عندی بیتحسب بالقانون ده:

$$\text{Coverage measurement} = \frac{\text{Number of Decisions executed} * 100}{\text{Total number of Decisions}}$$

### :Example



فی نفس المثال اللي فات بتاعنا هنشيك على  
ال decisions في حالة ال True وفي حالة  
ال False (ودي اللي كانت ناقصانا ف التكنیک اللي  
فات)، فهنحضر input يعدي على كل حالات  
ال false عندنا حتى لو فاضية برضه، فلو قلنا مثلا  
7 هنلاقى إنها أقل من ال 20 وأقل برضه م ال 30،  
وبالتالي هتحققلي اللي انا عايزه من تغطية حالتين  
ال false اللي عندی، فنشتغل بيها بالضبط زى  
ال 33 ف المثال اللي فات، ويكون ال inputs  
عندي هما ال 33 وال 7 زى ما هو واضح كده  
ال flowchart.

طبقا للقانون اللي فوق ده ال decisions اللي  
عندي ونفذناهم كلهم، فيكون:

$$\text{Dec. Coverage} = (2 * 100) / 2 = 100\%$$

**ملحوظة:** Decision Coverage غطى البرنامج ككل أكثر من statement coverage لأنه حتى الحالات بناءة الشروط اللي مافيهاش كود غطاها برضه على عكس الـ statement coverage. عشان كده لو الـ decision coverage rate = 100% ده معناه إن أكيد الكود كمان كله اتغطي، يعني الـ statement coverage rate = 100% لكن العكس غير صحيح بمعنى إن 100% decision coverage مش شرط تحققى statement coverage أو Decision coverage = 100% لكن نسبة الـ Statement coverage كانت 50% بس لأنه مادعاش على حالتين الـ false، أما في المثال الثاني نسبة الـ decision coverage كانت 100% لأنه عدى على كل الكود ونسبة الـ coverage كانت 100% برضه لأنه عدى على جميع حالات الشرطين اللي عندي... النقطة دي دايماً بتتجلى في امتحانات الـ ISTQB وأسئلة الـ interviews.

### :Choosing the right technique

عشان أشتغل على برنامج معين وأقول أنا هشتغل white box أو black box أو حتى أحدد معين على جزء معين من البرنامج فيه شوية معايير هي اللي بتحكم اختياراتي وتخلينى اختار الاختيار الأمثل اللي يوصلنى لأفضل نتيجة ممكنة.

#### المعايير دي هي:

- Type of system
  - Standards
  - Customer or contractual requirements
  - Level of risk
  - Type of risk
  - Testing objectives
  - Documentation available
  - Knowledge / skills of the testers
  - Time and budget
  - Development processes
- الهدف اللي عايزه من التيسىت ممكن يوجه لإسلوب معين في العمل.

## How much testing is enough?

قلنا قبل كده إن قرار إنى أوقف التيسينج معتمد على **Risk**:

### الـ risk ده بيشمل:

- إن اليوزر يلاقى **faults** مهمة (احتمال إن ده يحصل قد إيه بعد ماخلصنا تيست).
- **Risk** تكلفة الغير مكتشفة (هل لو فيه **failure** اكتشفها اليوزر هتكلف كتير على شغله علينا ولا لا).
- **Risk** إنى أطلع **Release** من غير تيست خالص أو التيست بتاعه لسه ماكملش.
- **Risk** خسارة المصداقية بتاعة شركة والـ **market share** لو نزلت المنتج بتاعى بالوضع الحالى أو المتخطط له.
- **Risk** خسارة **market window** معين.
- **Risk** إنى أعمل **over-testing (exhaustive testing)** أو تيست مش مؤثر.

التيست دايماً وقته قليل، عشان كده بنسخدم الـ **risk** لتحديد:

- أعمل تيست على إيه الأول؟
- أعمل تيست إيه أكثر؟
- إيه الـ **items** اللي هركز عليها ف التيست بتاعى؟
- إيه اللي مش هعمله تيست؟ (ع الأقل الـ **release** ده)

كل ده بيساعدنى إنى أحدد الوقت المطلوب للتيسينج وكمان أضبط التيست بتاعى على الوقت المتاح عندي عن طريق الـ **prioritizing**، فلازم أعمل **prioritization** للتيست بتاعى بحيث بحث منين ماوقف تيست بيقى عملت أفضل تيست ممكن ف الوقت المتاح.

## Day 9

### :Test Case

بعد مانحدد ال techniques والبيانات المناسبة عشان نتيست حاجة معينة بنحول الكلام ده لشكل منظم أكثر إسمه test case عشان أقدر استخدمه وانا بعمل execution للتسينج.

الtest case هى نوع من أنواع ال test documentation بيعمله ويكتبه التيسير ، وبنعمله عشان نشيك على ال expected results اللي عندي على عملية تيست معينة.

التسير كيس test case ممكن تكتب على tools كتير زي ال TFS مثلًا أو حتى على ملف Excel، وبيكون شكلها شبه الشكل ده مثلًا:

Banking System							
ID	Title	Pre-condition	Steps	Test Data	Expected Result	Priority	Author
"Deposit" Operation							
ITI_BS_Deposit_002	Deposit a sum of money	1- Manager Should be logged in 2- "Deposit" page should be opened 3- At least 1 customer should be registered before 4- Balance should contains [sum] of money.	1- Check Balance 2- Enter a valid "Account No". 3- Enter a valid "Amount Deposit". 4- Enter a valid "Description". 5- Click on "Submit" button	26515 Any amount like "24000" Cache	Balance = [sum] Account number should be accepted. Amount deposit should be accepted. Description should be accepted. 1- "Amount Deposite" page should be opened. 2- "Transaction Details" table should be appeared. 3- "Transaction Details" table should contains Current Balance = [sum +Added sum].	3	Ahmed etman
ITI_BS_Deposit_003	Reset depositing a sum of money	1- Manager Should be logged in 2- "Deposit" page should be opened	1- Enter a valid "Account No". 2- Enter a valid "Amount". 3- Enter a valid "Description". 4- Click on "Reset" button	26515 Any amount like "24000" Cache	Account No should be accepted. Amount should be accepted. Description should be accepted. "Account No" text box, "Amount" textbox, and "Description" text field should be blank.	1	Ahmed etman
ITI_BS_Deposit_004	Deposit a sum of money to unregistered account	1- Manager Should be logged in 2- "Deposit" page should be opened	1- Enter an unregistered "Account No". 2- Enter a valid "Amount Deposit". 3- Enter a valid "Description". 4- Click on "Submit" button	777777 Any amount like "24000" Cache	Account number should be accepted. Amount deposit should be accepted. Description should be accepted. 1- An error message should be appeared. 2- Message is: "Account does not exist".	2	Ahmed etman

### Test Case

### :Test Case attributes

التسير كيس بتكون من:

- ID •
- Title •
- Environment •
- Pre-conditions •
- Steps •
- Test Data •
- Expected Result •
- Priority •
- Status •
- Revision history •

### :Test Case ID -1

ده أهم حاجة في التيس كيس كلها، فالتيست كيس من غير ID عامل بالظبط زى بيت من غير عنوان، فهو يعتبر عنوان التيس كيس ولما بحب أتعامل مع التيس كيس أو أتكلم عنها بستعمل ال ID كعلامة مميزة للتيست كيس دى عن غيرها، عشان كده ال ID لازم يكون Unique (يعنى مابيكرر ش تانى) مش على مستوى ال test suite (اللى بنكتب فيها التيس كيس) بس لكن كمان على مستوى الشركة كلها.

السبب ف كده ببيان أكثر فى المستقبل لما التيس كيس دى تخزن وأحب أعمل إحصائيات أو updates معينة على ال test cases أو حتى أنقلها من test suite للثانية... كده يعني.

ال ID يإما ال tool بتعمله بشكل أوتوماتيك يإما بيعمل manually، بس ف الحالة دى لازم تتفق على قاعدة معينة ف التسمية مع ال QA.

طريقة تسمية ال Unique ID ممكن تختلف من شركة للثانية، فمن ضمن النماذج دى:

- [Company Name] \_ [project Name] \_ [module name] \_ unique number.
- [Project name] \_ [day date] \_ [unique number]
- [unique number only with specific format]

فمثلاً ممكن نكتب ال test case ID بالطريقة دى:

"MyCompany\_MyWebsite\_Login\_001"

### :Test Case Title -2

ال title هو وصف التيس كيس دى وبتعمل إيه وكده، حاول فى وصف التيس كيس تكون مختصر على قد ماتقدر وماتكرر نفس الكلام مع كل تيست كيس، فمثلاً لو التيس كيس بتاعتنا إننا بنعمل login ب user name وباسورد صح فممكن ف الحالة دى نكتب ال title بالشكل ده:

"Login with a valid User Name and a valid Password"

### :Environment -3

الحاجة اللي شغالة عندك ع الجهاز واللى بشتغل بيها عشان أنفذ التيس كيس بتاعته زي مثلاً ال OS بتاعك أو نسخة ال IDE بتاعك أو ال Browser... إلخ، زي كده مثلاً:

"Windows 10 - Google Chrome - Visual Studio 2017"

### :Pre-conditions -4

دى الحاجات أو الشروط المعينة اللي لازم تتحقق قبل ما نفذ التيسىت كيس بتعاتى، زى مثلا لو بعمل login لازم يكون فيه User Name وباسورد متسجلين فى ال database مثلا، فممكן التعبير عنها بالطريقة دى:

"Database must contains an user name and a password"

### :Steps -5

دى بقى الخطوات اللي بنفذها فى التيسىت كيس بتعاتى عشان أتأكد إن ال Expected Result بتعاتى إتحقق، زى كده مثلا:

- 1- Go to URL: "www.mysite.com"
- 2- Enter a valid "User Name"
- 3- Enter a valid "Password"
- 4- Click on "Login" button

طبعا واحنا بندخل الداتا لازم نخلى أسامي ال fields بين " " عشان الدقة فى التيسىت كيس بتعاتى.

### :Test Data -6

دى كل البيانات اللي ممكن أحتاجها وأنا شغال ف التيسىت كيس بتعاتى زى كده:

User Name: "ahmed etman"

Password: "123"

### :Expected Results -7

دى النتيجة أو النتائج اللي متوقع إنى ألاقيها بعد ماخلص كل الخطوات أو خطوة معينة، ولو لاقيت نتائج تانية غير ال Expected results دى بعمل bug report بالكلام ده (هنجيله كمان شوية).

ال Expected results واحنا بنكتبها لازم نكتبها بصيغة المفروض إن يحصل كذا لأن ده الواقع بتعاتها، إحنا متوقعين إن يحصل كذا مش حصل كذا فعلا، فممكnen ال Expected Result تكتب بالشكل ده مثلا:

"System should be logged in"

### :Priority -8

احنا بنرتب أولوية تنفيذ التيسىت كيس، بحيث لو فى أى وقت جاتلنا تعليمات إننا نوقف التيسىت بيقى عملنا **test cases** عندنا (زى ما قلنا قبل كده)، وال**priority** دى بيتعبر عنها من **P1** لـ **P4** لـ **P1** **priority test cases** لأقل **P4** **test cases** مهمة.

### :Status -9

دى بقى حالة الـ **test case** بعد التنفيذ، و بتقسام لـ 3 حالات:

- **Passed**: لو التيسىت كيس نفذتها و طلعتنى الـ **Expected Result**.
- **Failed**: لو التيسىت كيس طلعتنى نتيجة غير الـ **Expected Result** (bug يعني).
- **Blocked**: لو مش قادر أنفذ التيسىت كيس دى لأى سبب من الأسباب، زى مثلا الصفحة مش راضية تحمل أو زرار الـ **Login** مش مفعول فمش قادر أكمل **execution**، ف الحالة دى برضه ممكن أعمل **bug report** بالحاجة اللي معطلانى.

### :Revision History -10

الـ **field** ده بيبقى فيه كل الـ **history** بتاعة التيسىت كيس دى، والـ **history** ده عباره عن:

- **Created (date / name)**: لما تتعمل أول **version** من التيسىت كيس دى، وبيكون فيها إسم اللي عملها بتاريخ الإنشاء.
- **Modified (date / name)**: لما يحصل أى تغيير ع التيسىت كيس بيكتبوا فيه تاريخ التعديل ومين اللي عدل وعدل إيه بالظبط.
- **Reason**: سبب التغيير إيه.

غالبا الـ **field** ده بيبقى موجود أكثر فى الـ **tools** اللي بنكتب فيها التيسىت كيس، فيبيس جله بشكل شبه أوتوماتيك حسب الـ **tool** نفسها.

فالتيست كيس ممكن تكون بالشكل ده:

ID: MyCompany\_MyWebsite\_Login\_001

Title: Login with a valid User Name and a valid Password

Environment: Windows 10 - Google Chrome - Visual Studio 2017

Pre-conditions: Database must contains an user name and a password

Steps:

- 1- Go to URL: "www.mysite.com"
- 2- Enter a valid "User Name"
- 3- Enter a valid "Password"
- 4- Click on "Login" button

Test Data:

- User Name: "ahmed etman"
- Password: "123"

Expected Result: System should be logged in

Priority: P1

Status: Passed

Revision History: Created By "Ahmed Etman" at 01-01-2018 12:07 AM

Maintainability of test cases

لو عندنا سيسـتم مثلاً فيه زرار ال "Login" قررنا تغيير إسـمه لـ "Sign-in"، وبالتالي هنغير كل ال "Login" في كل ال test cases بـ "Sign-in" ، الكلام ده ينفع لو عندى 3 test cases مثلـاً، طب لو عندى 500 test case فيهـم كلمة "login" هل هنغيرـهم 500 مرة؟ ممكن نكسلـهم على أساس إنـ ال login معناها مشابـه للـ sign-in، لكن فـ الآخر هنلـاقـي إنـ التـغيـيرـات الصـغـيرـة الليـ مش مستـاهـلة دـى بـقتـ كـثيرـ بمـرورـ الـوقـتـ، وبالتاليـ التـيـسـتـ كـيسـ هـنـقـدـ جـزـءـ منـ الـ practical valueـ بـتـاعـتهاـ لأنـهاـ هـتـبـقـيـ أـصـعـبـ فـ الـ executionـ، أوـ نـضـطـرـ بـقـىـ تـغـيـيرـ الـكلـامـ 500ـ مرـةـ، وـدهـ بـرضـهـ حلـ مشـ عـملـىـ وهـيـاـخـدـ منـاـ وقتـ طـوـيلـ بـرضـهـ.

طب نتحر ولا نعمل إيه؟ إيه حل الحوار ده ياعم؟

الحل إننا واحنا بنعمل التيسست كيس لازم نخليها سهل التغيير فيها، فلو مابنفكرش في ال **test case** يبقى مابنفكرش ف بكره، وده غلط جدا.

### فعل الموضوع ده:

- بنعمل **document** بنسميها **QA KnowledgeBase** وده بنحط فيها كل الخطوات اللي ممكن تتكرر في **test cases** كتير أو اللي تحتاج تفاصيل كتير في شرحها، أو حتى الحاجات اللي سهل توقيعها، وبنعمل **link** أو **reference** للحاجة اللي بحطها في ال **QA KB** في التيسست كيس.
- لو بنعمل سيرش بكلمة معينة أو بندخل ميل أو باسورد معين مانكتبوش في ال **test case** الكلمة أو الميل ده، لازم نخليها بشكل عام، ولو فيه طريقة معينة عشان نجيبها من ال **Database** بنحطها في ال **(QA KB)** **QA KnowledgeBase** ونعملها لينك في التيسست كيس.
- لو بتعمل **generate** لحاجة معينة زي رقم جديد لحاجة فبتحط **article** للكلام ده ف ال **QA KB** وبتعمله لينك في التيسست كيس.

فيشكل عام القواعد اللي نقدر نعملها عشان نحسن ال **maintainability** بتاعة التيسست كيس متوضحة في الكلمتين الجايين دول.

### :Measures we took to improve the maintainability of test cases

- خلى التيسست كيس بتاعتاك **data driven** يعني قابلة لاستقبال أي داتا مش واقفة على داتا معينة.
  - ماتخليش الخطوات المتكررة والسهل توقيعها في التيسست كيس بتاعتاك (**اعملها ف ال QA KB** زي ماقلنا قبل كده).
  - ماتكتبي بالتفصيل الممل أوى ف التيسست كيس بتاعتاك (زي مثلا ال **(search keyword)**).
  - لما تلاقى سيناريوهات أو خطوات متكررة كتير في أكثر من تيسست كيس انقلها لل **QA KB** زي ماقلنا قبل كده واعملها **link** في التيسست كيس بتاعتاك.
- المثال الجاي هيوضح الدنيا أكثر.

### :Example

لو قلنا ان فى سيناريو لـ **test case** معينة ان اليوزر هيعمل **login** ويدخل بضيف **credit card** جديدة على حسابه ويعمل **logout**.

ف الأول هناخد الكلام ده عادى خالص، فتكون الخطوات عندي كالتالى:

Test case steps:

1. Go to URL "www.mysite.com"
2. In "User Name" textbox write "Ahmed"
3. In "Password" textbox write "123"
4. Click on "Login" button
5. On the home Page, click on "Add Credit Card"
6. At "Credit Card type" drop-down list, Choose "Visa Card"
7. At "Credit Card ID" textbox, write "xyz123"
8. Click on "Save" button
9. Click on "Menu" drop-down list.
10. Click on "Logout" option.

لو حبينا نغير أى بيانات فى التيسست كيس دى هنعمل تعديلات كتير لو متكررة فى **test cases** كتير، لكن بعد مانطبق قواعد الـ **maintainability** بتاعتنا ممكن تتعمل بالشكل ده:

Test case steps:

1. Login with a valid User Name and Password (see QA KB).
2. Add new Credit Card (see QA KB).
3. Logout from the system (see QA KB).

## QA Knowledge Base:

### Login with a valid User Name and Password:

1. Go to URL "www.mysite.com"
2. In "User Name" textbox write any valid user name like "Ahmed"
3. In "Password" textbox write any valid password like "123"
4. Click on "Login" button

### Add new Credit Card:

1. On the home Page, click on "Add Credit Card" button
2. At "Credit Card type" drop-down list, Choose any valid type like "Visa Card"
3. At "Credit Card ID" textbox, write any valid card ID like "xyz123"
4. Click on "Save" button

### Logout from the system:

1. Click on "Menu" drop-down list.
2. Click on "Logout" option.

طبعاً الـ **QA Knowledge Base** دى فى document لوحدها وبنعمل link على الكلام اللي فـ **steps**، وبكده نكون حقنا الـ **maintainability** فى التيسىت كيس بناعلى بحيث:

- 1- قلنا عدد الخطوات فـ التيسىت كيس من غير ما نغير المفهوم العام للـ التيسىت كيس.
- 2- فى أي عملية **maintenance** للـ التيسىت كيس هتغير البيانات مرة واحدة بس بدل ما بغيرها أكثر من مرة طب احنا فـ التعديل ده عملنا إيه؟
- 1- خلينا الـ **test case** ماتعتمدش على داتا معينة وممكن تتغير خالص (**Data Driven**)، إحنا إدیناله حاجة شبه الداتا دى يمشي عليها، زي "123" any valid password like "123".
- 2- خلينا الخطوات التكرارية والسهل توقعها بره التيسىت كيس.
- 3- ماءدیناش تفاصيل مملة فى التيسىت كيس.
- 4- حرکنا الـ **scenarios** اللي ممكن تتكسر كتير أو تفاصيلها واضحة للـ **QA KB** وما خليناهاش فـ التيسىت كيس.

### :The number of expected Results inside one test case

أحياناً ممكن نلاقى عندنا أكثر من **expected result** في نفس التيسىت كيس، ف العادى التيسىت كيس لازم ليها **expected result** واحدة بس، لكن في حالة لو التيسىت كيس ليها أكثر من **expected result** نعمل إيه في الحالة دي؟

**بعض الحالات التي قد تحدث:**

1- يالإما تقسم الـ **test cases** لـ **2 descriptions** **test case description**.

2- يالإما ماتغيرش حاجة في الـ **test case description** ويبقى عندك **2 Expected Results** في نفس التيسىت كيس، لكن في الحالة دي التيسىت كيس ه تكون **Pass** في حالة واحدة بس هي إن الـ **2 Actual Results** يتحققوا في الـ **expected results** غير كده التيسىت كيس دي ه تكون **fail**، وفي الحالة دي بنكتب الـ **expected result** لكل واحدة فيهم قدام كل خطوة بتحققها، زى كده مثلاً:

10- Do something, compare the AR (Actual Result) and the ER (Expected Result).

11- Do something else, compare the AR and the ER.

وفي الـ **expected result** نكتب:

- After step no.10, the result is [ER]
- After step no.11, the result is [ER]

### :Example

لو عندنا تيسىت كيس بتعمل فيها **upload** على صورة وبتطلعك رسالة بعد الصور اللي رفعتها، فتكون الخطوات بالشكل ده:

1- Check number of uploaded photos (see QA KB).

2- Upload a photo, Compare the AR and the ER (see QA KB).

3- Upload another photo, Compare the AR and the ER (see QA KB).

في خانة الـ **expected result** هنكتب بقى:

- After step no.1, a message should be displayed that "Number of uploaded photos is [number]" لاحظ إننا مانكتب الرقم لأنه بيتغير مع كل عملية رفع
- After Step no.2, a message should be displayed that "Number of uploaded photos is [number +1]"
- After Step no.3, a message should be displayed that "Number of uploaded photos is [number +2]"

فلو كان مرفوع قيل كده 5 صور، بعد أول خطوة هيقولك إن عدد الصور 5 وبعد تانى خطوة هيقولك إن عددهم 6 وبعد تالت خطوة هيقولك إن عددهم 7 ... الكلام ده بسيط وسهل ومش مستاهل اننا نعمل تيست كيس لكل واحدة فيهم.

### :Bad Test Case Practices

فيه 3 حاجات حاول ماتعملهمش وانت بتعمل الـ **test cases**، الحاجات دى هي:

- **الـ dependency** بين الـ **test cases** إن تيست كيس شغلها يكون معتمد على تيست كيس تانية.
- **الـ description** الضعيف للخطوات بتاعة التيست كيس (مش واضحة).
- **الـ title** الضعيف للـ **Expected Result** بتاعة التيست كيس أو الـ **description**، لأن دول من أهم الحاجات المرتبطة بمضمون التيست كيس.

### :Example

لو عندي 2 test cases زى دول مثلا:

Test Case #1:

Title: Preparing for purchase operation

Steps:

1. Login
2. Go to purchase page
3. Purchase any smart phone
4. Check your "shop cart"

Expected Result: smart phone should be added to the user's shop cart.

Test Case #2:

Title: Cancel purchasing products

Steps:

1. Repeat steps 1, 2, 3 and 4 from test case #1
2. Click on "Cancel Purchase" button

Expected Result: purchase canceled

تعالوا ننشوف ال 2 test cases دول ظروفهم إيه

أول تيست كيس ال title بتابعها Preparing for purchase operation وده مش واضح... لازم عنوان التيست كيس يكون واضح زى مثلا Adding Smart Phone to the shop cart كده... بقى أوضح.

برضه الخطوات فى أول تيست كيس مش واضحه، زى مثلا:

>Login: فين البيانات اللي هدخل بيها؟

Go to purchase page: مكان الصفحة دى فين؟ شبح أنا عشان أجيبها من سايت معرفش عنه حاجة!

Purchase any smart phone: الكلام ده غير دقيق... المفروض إنى بعمل تيست إنى أضيفه فـ shop cart مش بشتريه.

تاني تيست كيس:

1- حنة repeat steps دى أكبر غلط... إفرض غيرنا التيست كيس رقم 1 دى أو إتحذفت لأى سبب م الأسباب... أتسوبح أنا يعني؟ ماینفعش طبعا... لازم كل تيست كيس تكون مستقلة بذاتها وماتعتمدش على تيست كيس تانية.

Expected Result: برضه مش واضحه... المفروض أقول إن الموبайл اتحذف من ال shop cart وال shop cart بقت فاضية وزرار complete Purchase مثلًا بقى ...inactive bugs دقق جدا وبيفرق معايا وممكن يطلع منه .

فال 2 test cases ممكن تكون زى كده مثلا:

### Test Case #1

Title: Adding Smart Phone to the shop cart

Steps:

1. Login with a valid User Name and Password (see QA KB).

2. Click on "Purchase a product" Page

3. Choose any smart phone, and click on "Add to Cart" button.

4. Click on "shop cart" button at the top right part of the page

Expected Result: smart phone should be added to the user's shop cart.

### Test Case #2:

Title: Cancel purchasing products

Steps:

1. Login with a valid User Name and Password (see QA KB).

2. Click on "Purchase a product" Page

3. Choose any smart phone, and click on "Add to Cart" button

4. Click on "shop cart" button at the top right part of the page, and check the Expected Result.

5. Click on "Cancel Purchase" button

### Expected Result:

- After step No.4:
  - The smart phone should be existed at "Shop cart" Page.
  - "Complete Purchase" and "Cancel Purchase" buttons should be active
- After step no.5:
  - The selected Smart phone should be deleted from "Shop cart" page
  - "Shop cart" page should be empty
  - "Complete Purchase" and "Cancel Purchase" buttons should be inactive.

### QA Knowledge Base:

#### Login with a valid User Name and Password:

- 1- Go to "www.mysite.com".
- 2- At the top right part of the page, click on "Login" button.
- 3- At "User Name" type any valid username like "Ahmed etman".
- 4- At "Password" type any valid password like "123".
- 5- Click on "Login" button.

#### :Data-Driven Test Cases

التيست كيس بتاعتنا لازم تكون **data-driven**، بمعنى إنى مأدخلش بيانات معينة فى التيست كيس وأقىد التيستر اللي بيعملها **execution** بالبيانات دى (مش شرط على فكرة إن اللي كتب التيست كيس هو اللي ينفذها، ممكن يكون فى **team** كبير جدا وحد تانى بيعمل الـ**execution**)، فالهدف من البيانات هو نوع البيانات مش البيانات نفسها، بمعنى إنى لو قلتله دخل باسورد المقصود عندي أى باسورد ينفع مش الباسورد "123" مثلًا في حد ذاته.

لو دخلتله بيانات معينة وخليته ملزوم بيها وجييت بعد كده حذفت البيانات دى أو غيرتها لأى سبب من الأسباب سواء عن عمد أو عن غير عمد فكده شغل الرجال ده هيقف ومش هيقدر يكمل لأنه ملزوم ببيانات معينة، أو ع الأقل هيعطل عقبال مايحل الحوار ده، إنما لو قلتله "123" any password like "123"

كده أنا اقترحت عليه حاجة معينة ومازلزمتوش بيها ولو عنده بيانات هو عارفها غير دى هيتعامل بيها عادى حتى لو اللي اقترحته ده اتغير لأى سبب من الأسباب.

غير كده إن فيه **test cases** تانية بتتغير بتغيير البيانات اللي داخلة ليها، ففي الحالة دى لازم التيسى كيس ماتعتمدش على بيانات معينة لأن الـ **expected result** بتختلف باختلاف البيانات دى.

### :Test Case with Assumption

#### Example:

##### Test Case #1

Steps:

1. Set the count of MasterCard transactions in the DB to 0.
2. Buy a book using MasterCard.

Expected result: The transaction success code is "20".

##### Test Case #2

Steps:

- a. Buy a book using MasterCard.

Expected result: the count for MasterCard transactions is "2".

هنا عملت خطأ كبير جدا إنى عملت التيسى كيس رقم 2 على اعتبار إن التيسى كيس الأولانية اتنفذت... طب افرض **#1** دى حذفناها أو ماتنفذتش لأى سبب من الأسباب وجت تتنفيذ رقم 2 ... هل هنطلع الـ ER اللي أنا عايزها؟ أكيد لا، وده غلط وقعت فيه.

#### عشان كده أفضل حاجة وانا بكتب تيسى كيس أراعى الحاجات دى:

1. التيسى كيس لازم ماتشاورش على أى تيسى كيس تانية ولا تعتمد عليها فى الخطوات بتعادلها.
2. لازم التيسى كيس ماتعتمدش على تنفيذ أى تيسى كيس تانية، بحيث لو اتنفذت لوحدها تقدر تدىنى النتيجة اللي عايزها منها.
3. أحاول على قد ماقدر ماستعملش أى اختصارات فى التيسى كيس، لأن اللي واضح بالنسبة دلوقتى ممكن ماييقاش واضح بعد سنتين ثلاثة، واللى واضح ليها ممكن ماييقاش واضح لأى تيسى زميلى.
4. تكملا على النقطة اللي فاتت... لازم وانا بعمل تيسى كيس أعمل حسابي ان حد تانى هيقرأها وينفذها فالمستقبل، فالتيست كيس اللي أنا بس اللي أقدر أعملها **execution** ماتتفعش تتعمل فى أي شركة، ممكن انت تأخذ إجازة أو حتى تسيب الشركة خالص، فالحالة دى اللي هيستلم الشغل

من بعدك هو اللي هيتعسف ف الحوار ده، ممكن إنت شخصيا تنسي إنت كنت بتعمل إيه ف التيسىست كيس دى ومنين بيودى على فين... فلازم تخلى الدنيا واضحة وصريحة معاك ف التيسىست كيس.

ال 5. **Expected Result** مайнفعش إطلاقا إنى أعملها **reference** فى أى **document** ولا حتى فى **QA Knowledge Base** ، الدنيا مش مستاهلة القلق ده كله، ده غير إن **Actual Result** دى اللي بنقارنها بال **Expected Result** وبيعتمد عليها إذا كان فيه bug ولا لأن... عشان كده خلى **ER** بتاعتكم فى مكانها ف التيسىست كيس واشرحها بالتفصيل الممل لو محتاجة شرح... مش عيب ولا حرام والله 😊.

ال 6. وانت بتشرح **ER** حتى لو شايف النتيجة تافهة وبديهية لازم توضحها عشان ماتسقطش منك أى نقطة بعد كده، يعني مثلا لو بتشك على **components** بتاعة الصفحة ماتقولش إنما لو قلت الصفحة فيها **textbox** الفلانى والزرار الفلانى والصورة الفلانية وهكذا كده أنا وصلت لنتيجة مقعنة أقدر أكمل عليها فى أى **maintenance** يحصل ع الجزء ده بعد كده.



### **:Test Suites**

قلنا قبل كده إن **test suite** دى عبارة عن **test cases** أو أى حاجة بيتكتب عليها) **Shopping Cart** (أو أى **Word**) **file** أو **Excel** بنجمع فيه ال **test cases** اللي بتعمل تيسىست على **component** أو صفحة معينة (Spec #1455 "Privacy Rules" مثلا) و/أو **specification** معينة (زى مثلا

### **:Test Suite Items**

- **Author** : الشخص / الأشخاص اللي عملوا ال **test cases** اللي ف **test suite** دى.
- **Spec ID** : **unique ID** بتاع **spec** اللي بنعمل عليها التيسىست اللي ف **test suite** دى، وال **ID** ده لازم يكون **linked** مع ال **spec** الموجودة على ال **intranet** أو الشبكة الداخلية الخاصة بالشركة.
- **Product Manager** (PM) أو أى حد ثانى.
- **developer(s)** المسؤول / المسئولين عن كتابة الكود اللي بيحقق ال **spec** دى.
- **Priority** بتاعة ال **test suite** دى (عادة درجاتها بتكون من 1 ل 4)، بحيث ال 1 ده يكون **top priority** وال 4 ال **lowest priority** عادة ال **test suites** بتاعة ال **priority** بتتطابق مع ال **priority** الخاص بال **spec**.

**Note** : يفضل وانت بتكتب ال **test suite** تستخدم ال **formatting functionalities** (زى **Underline** وال **Italic** وال **Bold** وال **الألوان** ... إلخ) ، ده ليه فايدتين:

- عشان تأكد ع الحاجات المهمة اللي لازم اللي بيقرأ ال Test Suite ياخد باله منها.
- وكمان عشان تسهل القراءة على اللي بيقرأ ال Test Suite وتخلية مصحح معاك عطوطول.

### :Marrying Test Suites

لو عندي test suites 2 متشابهين لبعض ممكن إنى أدمجهم مع بعض فى test Suite جديدة بتكون من جزئين، كل جزء منهم خاص بواحدة من الـ test suites القديمة، وطبعاً الـ test suite الجديدة بتأخذ unique ID، لكن الـ test cases بتفضل على وضعها وبتحتفظ بنفس الـ ID بتاعها لأنها unique فمش تحتاجين نديلها ID جديد.

لو قلنا عندنا 2 test suites :**Example**

Test suite 1: tests with Visa and MasterCard.

Test suite 2: tests with AmEx.

ف الحالة دى ممكن أضمهم على بعض فى test suite واحدة تجمعهم وتكون إسمها حاجة زى Credit Card Payments وبتكون من جزئين:

Part 1: tests with Visa and MasterCard.

Test suite 2: tests with AmEx.

### :Checklists

الـ Checklists هى عبارة عن تيست كيس بسيطة بيتم اعتبار الشخص اللي هي عملها execution عارف الـ application كويس، وما بيبقاش فيها الـ Database advanced verifications مثلًا، أنا شخصياً استعملت الـ checklists قبل كده وانا بعمل Smoke test (هنجيله Queries كمان شوية) على نسخة جديدة على برنامج كنت شغال عليه قبل كده، عادة الـ checklist بيبقى شكلها عامل زى كده مثلًا:

Action	Expected	Check
Do Transaction with Visa	Success	<input type="checkbox"/>
Do Transaction with MasterCard	Success	<input type="checkbox"/>
Do Transaction with AmEx	Success	<input type="checkbox"/>

قبل مانعمل checklist بنطبع test cases execution دى ونبدأ نشتغل، لو الذيا تمام والتيسىت كيس حالتها passed بنعلم على المربع اللي ف check، لو failed أو blocked مابنعلمش عليها.

عادة بنسخدم checklists لو المشاريع أهميتها قليلة حتى لو بعد سنتين من إصدارها، أما لو project مهم ف الحالة دى لازم نعمل formal test case must العادية اللي نعرفها.

## Day 10

### :Bug Reports

عملنا الـ **test cases** وعملناها **test design techniques** وطلعنا **execution** (اختلاف الـ **Expected Result** عن الـ **Actual Result**)... جه دلوقتى دور اننا نبلغ عن الـ **bug** اللي لاقيناها دى عشان تتحل، فهنا بنعمل حاجة اسمها الـ **Bug Report**.

الـ **Bug report** ده ليه **tools** كتير أقدر من خلالها أعمله وأتابع حاليه، الـ **tools** دى بنسميها **Tracking Tools** زى مثلا Mantis, TFS, Redmine, Jira, Bugzilla حتى ممكن أعمل الـ **Bug report** على ملف الـ **Excel**.

الـ **Bug report** ممكن يكون زى ده مثلا:

PMS Bug Report													
Bug ID	Title	Module	Environment	Steps	Expected Result	Actual result	Bug type	Severity	Screen shot(s)	Videos	QC Status	Dev Status	Priority
ACT_PMS_082	"Account Type" and "VIP" drop-down list items are displaying behind "Accounts" grid	PMS - Accounts	Windows 10 - Google Chrome - Visual Studio 2013	1- Login With a valid User name and password 2- Open "Accounts" page 3- Click on "Search Options" on "OK" button 4- Click on "Account Type" drop-down list and check drop-down list items 5- Click on "VIP" drop-down list, and check drop-down list items	"Account Type" and "VIP" drop-down list items should be displayed above "Accounts" grid	"Account Type" and "VIP" drop-down list items are displaying behind "Accounts" grid	UI	Low	ACT_PMS_082_1.png ACT_PMS_082_2.png		Open		
ACT_PMS_083	The system doesn't save "Travel Agent" accounts	PMS - Accounts	Windows 10 - Google Chrome - Visual Studio 2013	1- Login With a valid User name and password 2- Open "Accounts" page 3- Insert a new "Travel Agent" account 4- Close the new account popup 5- Insert a new "Travel Agent" account 6- Click on "Search" button and check search results	1- A message like "Account saved successfully" should be appeared after step no.3 2- The new account should be displayed at search results	1- There wasn't any messages displayed after step no.3 2- The new account isn't exist at search results	Functionality	Critical	ACT_PMS_083.png		Open		

### الـ **bug report** بيكون من:

- **Bug ID**: بيبقى unique وبيتطبق عليه نفس معايير الـ **test case ID** اللي قلناها قبل كده، لكن طبعا مش نفس الـ **ID** بتاع التيست كيس اللي لاقينا فيه الجاية دى، ممكن ألاقي بجاية من غير تيست كيس عادي.
- **Title**: بنوصف الـ **bug** بشكل مختصر، وبرضه بيتطبق عليه اللي قلناه على الـ **test case**.
- **Component**: الجزء أو الـ **Module** اللي لاقينا فيه الجاية دى.
- **Environment**: كنا شغالين على **environment** ايه بالظبط (Browser, Visual Studio 2017...etc).
- **Steps**: الخطوات اللي عملتها لغاية ماوصلات للجاية دى، وممكن نعمله هو كمان زى الـ **test cases** بالخطوات المتكررة وكده.
- **Expected Result**: النتيجة اللي كان المفروض تطلع معانيا.
- **Actual Result**: النتيجة الفعلية اللي طلعت معانيا.

- **Bug type**: نوع البجاية اللي طلعت زى مثلا:
  - **UI (User Interface)**: لو البح اللي طلعت معايا ليها علاقة بديزain مش مطبوط.
  - **Functionality**: لو حاجة فى وظيفة ال function أو ال component نفسه (ما يطبل علىش نتيجة مطبوبة).
  - **Performance**: لو شايف إن الجزء ده بطئ مثلا.
  - **Security**: لو حاجة تخص الاختراقات مثلاً أو خصوصية اليوزر... كده يعني.
- **Severity**: دى درجة تأثير ال bug دى فى ال module أو السيستم عندي، ودى اللي بيحددتها التيسير حسب تقديره للبجاية دى، وليها 4 درجات:
  - **Critical**: لو ال bug دى خطيرة جداً و موقفه شغل ع السيستم زى مثلاً زرار بضغط عليه مش شغال... function معينة مش شغالة خالص... كده يعني.
  - **High**: حاجة خطيرة ممكن ماتوقفش شغلـى و مالهاش أى حلول بديلة workarounds زى مثلاً بدخل رقمين يجمعهم يقوم ضاربـهم ف بعض.
  - **Normal**: هى bug عادية ممكن نشتغل وهى موجودة، زى مثلاً رسالة error الكلام بتاعها غامض ومش مفهوم.
  - **Low**: هى Bug عادية جداً ومش مؤثرة خالص و ممكن نشتغل بيها عادى، زى مثلاً spelling mistake فى جزء اليوزر ما يبرحوش كتير فى السيستم بتاعى.
- **Priority**: ده أولوية حل ال bug عند الديفلوبير، طبعاً الديفلوبير هو اللي بيحددها مش إحنا طبقاً لتقديره للموقف ولتفاصيل شغله، وليها 4 درجات برضه:
  - **P1**: لو هتحتل دلوقتى حالاً أو بأسرع وقت ممكن.
  - **P2**: لو هتحتل بعد ال p1.
  - **P3**: لو هتحتل بس بعد فترة.
  - **P4**: لو هتحتل منين مايفضى الديفلوبير أو حتى هتحتل فى release تانية خالص.
- **Screen shot** للبجاية اللي بعملها ال report.
- **Videos** للبجاية دى لو مانتفعش تتعمل بصورة (زى حاجة بتظهر وتخفى فجأة مثلاً)، وفيه tools ممكن تسمح برفع الفيديوهات عليها.
- **Status**: حالة البجاية دى، الحالات دى هي:
  - **Opened**: لو لسه معهولة والديفلوبير لسه ماشتغلش فيها.
  - **In progress**: لو الديفلوبير شغال فى حلها
  - **Fixed**: لو الديفلوبير حلها خالص (بس لسه التيسير ما شافهاش)
  - **Closed**: لما تبقى البجاية fixed بترجع للتيسير عشان يعمل عليها ال re-test، لوف الحالة دى الدنيا ماشية تمام والمشكلة اتحلت التيسير وال regression test، بيعبر حالتها closed وبنتهى دورة حياة ال bug.

- لو التيستر بيشوف المشكلة بعد مارجعت على انها **fixed** واكتشف ان المشكلة لسه موجودة وماتحلش.
- لو الbug دى اتأجلت عشان تتحل فى **sprint** أو **release** هتنحل دلوقتى (يعنى مش **Deferred**).
- لو الديفلوبير شايف إن دى مش **Bug** أساسا وإنها كده معمولة صح.
- لو الbug دى متكررة أو إتعملها **Duplicated**.

### :Example

لو عندي Bug إني فى صفحة **Login** لما بدخل الـ **email** والباسورد صح وأضغط على زرار الـ **Login** مابيسمحش بالدخول... طبعاً أنا عملت الكيس دى أكثر من مرة بأكتر من **username** وباسورد واتأكدت إنها bug مش مجرد **incident** (هنجيلها كمان شوية).

فهنهعمل **Bug Report** بالكلام ده، فممكن يكون زي كده مثلاً:

ID: MyProject\_Bug\_001

Title: The system doesn't allow logging with a valid UserName and Password.

Module: My Site - "Login" page.

Environment: windows 10 - Google Chrome - Visual Studio 2017

Steps:

1. Go to "www.mysite.com"
2. At the top right of the home page, Click on "Login" button
3. At "User Name" text box, enter any valid User Name like "Ahmed"
4. At "Password" textbox, enter the password for the same user name which entered in step 3 like "123".
5. Click on "Login" button

Expected Result:

1. System should be logged in
2. "My Profile" page should be displayed, containing the user data.

Actual Result: There isn't any changes after clicking on "Login" button.

Bug Type: Functionality

Severity: Critical

Screen shot(s): Login.jpg

Status: Opened

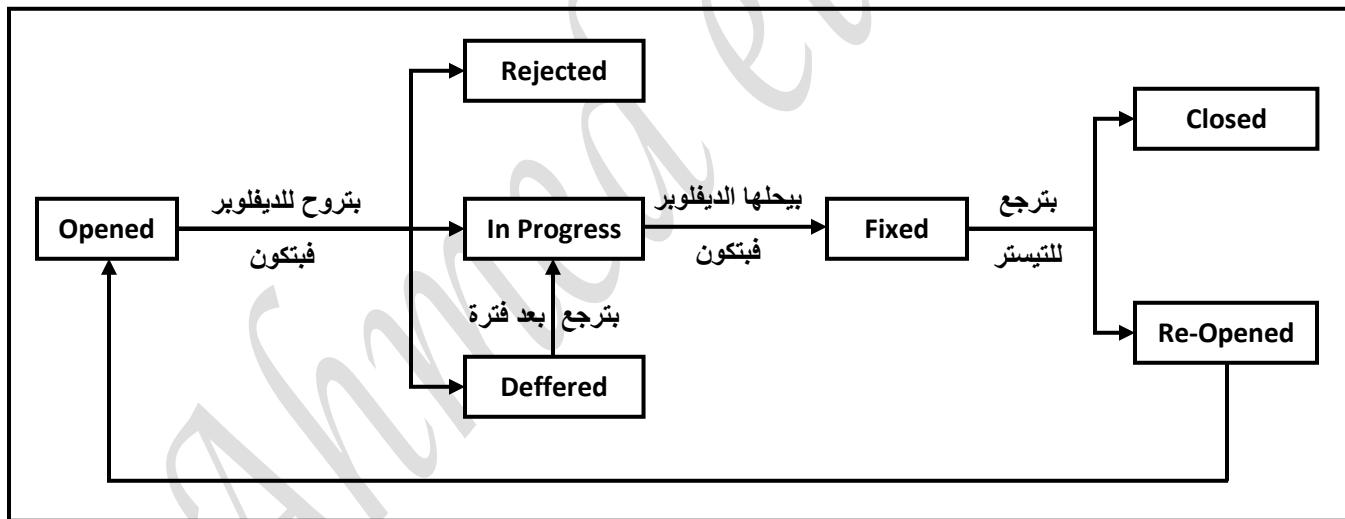
Priority: N/A (الديفلویر اللي بيحددنا مش أنا)

Author: Ahmed Etman 03-01-2018 2:56 PM.

طبعاً bug report ده بيتتحول يا إما مع الديفلویر المسؤول عن الجزء ده يا إما على ال team leader المسئول عن ال developing team أو على testing team وتببدأ تأخذ ال life cycle بانتها.

فدوره حياة ال bug نقدر تلخصها ف الشكل ده:

### :Bug Life Cycle



فاضل بس حاجة أخيرة حابب أتكلم فيها... فيه مصطلحات ماخدوش حقهم الكافى فى الكلام عنهم زى **Smoke Testing** و**Sanity Testing** و**test harness** نشوفهم.

ملحوظة بس... الكلام الجاي ده نتيجة بحث فى موقع وصفحات كتير متخصصة فى التيستينج ومش موجود فى المنهج الرسمى لـ **ISTQB**...انا بس حبيت أجيب المصطلحات دى ف الكتاب بحيث لو اتسألت ف أى انترفيو تعرف ترد وحتى لو طبقت الكلام ده ف شغلك تبقى عارف ده اسمه إيه.

### :Sanity Testing

ال sanity testing بعمله لما تكون سلمت build خلاص والعميل طلب منى update فى جزء معين أو جزء جديد ينضاف ع السيسنتم، ف الحالة دى بنعمل sanity testing ع الجزء أو ال function الجديدة عشان نتأكد إنه شغال بصورة طبيعية وإنه مأثرش ع ال behavior بتاع السيسنتم بصورة سلبية، وعادة بنطبق ال sanity testing بدون استخدام أى documented scripts أو أى test cases.

كلمة Sanity معناها الحرفي "الصحة العقلية"، فمن إسمه واضح إننا بنعمل التيسننج عشان نتأكد إن الحاجة الجديدة ماشيّة بصورة طبيعية وإن ما فيهش أى سلوك مريب أو غريب من السيسنتم بعد ما ضيفنا الجزء الجديد ده.

### :Smoke Testing

ال Smoke Testing بعمله لما بطلع version جديدة من السوفتوير بتاعي، فبعمل Functions الأساسية والمهمة فى كل السيسنتم شغالة تمام وبدون مشاكل.

عادة ف ال smoke testing بنستخدم عدد قليل جدا من test cases اللي بتخدم الغرض بتاعي بس فى إنى أعمل تيسننج على ال functions الأساسية بدون الدخول فى تفاصيل، طبعاً لو لقينا مشاكل بتتحل.

أصل تسمية النوع ده بال smoke test إنه كان مصطلح شائع فى hardware لما بيجمعوا دواير الكترونية وكهربائية مع بعض ويوصلوها بالفيشة هيشفوفوا لو طلعت دخان smoke يوقفوا التشغيل، أما لو ماطلعنوش دخان يكملوا شغلهم عادي.

### :Example

ال Facebook مثلاً فى بدايته طلع منه كذا build، فمع كل build جديد بيطلع لازم يعملوا forget password – logout – login على ال functions الأساسية فى البرنامج زى testing ...send a friend request – passed ... إلخ، لو الدنيا دى كلها تمام يبقى ال smoke test بتاعنا

أما بقى لو طلعوا حاجة جديدة زى ال Reactions على الليكات دى نقدر نعتبرها Update حصل فى موجودة، فهنا لما بيتعملها build لأول مرة بيعملوا sanity test فى الأول قبل ما يكملوا أى documented Cases / Scenarios

### :Testware

طبقاً للـ **testware** هو جزء من الـ **software** فالـ **ISTQB Glossary of terms** يساعد على تنفيذ عملية الـ **testing** لبرنامج أو **Application** معين، والمصطلح ده بيعبّر عن أي حاجة بتساعدنا في مراحل التيسينج المختلفة سواء **execution** أو **design** أو **planning** أو **Expected results** للتيست، زى مثلاً **tool** أو **software** أو **inputs** والـ **scripts** والـ **documents** والـ **Expected results**. تساعدنى في شغلِي وانا بعمل تيسينج.

يعنى م الأخر الـ **testware** هو أي حاجة ليها علاقة بالتيستينج سواء **tools** معينة أو **documents** أو **inputs** أو **Expected results** أو **Expected results** أو **inputs**.

### :Test Harness

الـ **test harness** هو أي حاجة ضرورية لتنفيذ عملية التيسينج، زى مثلاً **stubs** والـ **drivers** وأى وسائل تانية مساعدة، ويستخدم عادة فى:

- زى الـ **Automation testing** : زى الـ **parameters** والـ **test scripts** والـ **scripts** دى ونطلع النتائج عشان نحللها.
- **Integration testing** : بيعمل تيست ع الـ **interaction** بين **2 modules or units** وبنستخدم فيها الـ **stubs** والـ **drivers** زى ماقلنا قبل كده.

### :Example

لو قلنا اننا بنعمل تيست على **application** المفروض انه بيتعامل مع سيرفر ، لكن لسبب ما مافيش سيرفات موجودة، ف الحالة دى ممكن نعمل **test harness** كبديل للسيرفر عشان نخلص شغلنا.

من أشهر الـ **test harness tools** **Junit** بالـ **java** وده شغال بالـ **Nunit** ودى **tool** برضه **.Net Framework** بالـ **VB**.

كده احنا خلصنا كلامنا عن التيسينج... تعالوا بقى ندخل لـ **Part 3**، وده عباره عن نماذج من امتحانات الـ **ISTQB** (منهم الامتحان الرسمي اللي ع الموقع بتاعهم).

لو عايز نصحيتى... فى المسائل بالذات حلها بالورقة والقلم وماتترعش... الـ **BVA** او الأكواب ارسم ع الورق ، والـ **state transitions** امشى مع اتجاهات الأسهم، ولو ظهرتلك اجابتين صح اختار الأصح. كمان لازم تعرف إن نموذج الامتحان الرسمي مش معناه ان الامتحان الفعلى هيبقى زيه بالظبط، ممكن بيجي أصعب وممكن بيجي أسهل، فدور على امتحانات كتير ع النت وحلها، الامتحان عباره عن 40 سؤال يتحلوا في خلال 60 دقيقة، فإمساك لنفسك **stop watch** يامعلم وماتغشش نفسك لأن انت اللي هتشيل ف

الأخر... النجاح بيبدأ من أول 26 من 40... يعني لو حليت 26 سؤال من 40 صح تنجح، فرافق درجاتك في الامتحانات كلها وشوف إيه اللي دايما بتغلط فيه وركز عليه بالمذاكرة تانى... وربنا يوفقك.

كمان برضه يفضل تقرأ المنهج الرسمي لل ISTQB اللي من ع الموقع بتاعهم... أه احنا هنا شرحناه بالتفصيل بالعربي... لكن لازم تقرأه تانى لأنه ساعات بيجيب المصطلحات حرفيا من الكتاب، وده ليك تحميل ISTQB Syllabus 2011 اللي هي آخر نسخة نزلت لغاية النهاردة.

<https://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html>

وده ليك الامتحان الرسمي من ISTQB... ادخل عليه لأن كل شوية ينزلوا امتحان جديد، ولو الامتحان اللي هناك غير الامتحان اللي ف الكتاب حله برضه وشوف ظروفك إيه

<https://www.istqb.org/downloads/category/15-foundation-level-exam-documents.html>

احنا هنا هنشوف آخر نسخة من الامتحان اللي نزل version 2.9

وانت بتحل كل امتحان امساك ورقة خارجية أو اطبع الامتحان وحل مع نفسك الأول وحاول تخلص كل الأسئلة في خلال 60 دقيقة بس، وبعد الامتحان هتلaci الإجابات... صبح لنفسك وشوف انت جبت كام من 40، ولو فيه حاجة واقعة منك ظبطها قبل ماتدخل الامتحان الحقيقي عشان تنجح من أول مرة إن شاء الله.

يلا بینا بقى...

### Part 3

### ISTQB Exams

## ISTQB Official sample exam

### (Version 2.9)

#### Question 1

Which of the following statements BEST describes one of the seven key principles of software testing?

Answer Set:

- a) By using automated testing it is possible to test everything.
- b) With sufficient effort and tool support, exhaustive testing is feasible for all software.
- c) It is normally impossible to test all input/output combinations for a software system.
- d) The purpose of testing is to demonstrate the absence of defects.

#### Question 2

Which of the following statements is the MOST valid goal for a test team?

Answer Set:

- a) To determine whether enough component tests were executed within system testing.
- b) To detect as many failures as possible so that defects can be identified and corrected.
- c) To prove that all possible defects are identified.
- d) To prove that any remaining defects will not cause any failures.

### Question 3

Which of these tasks would you expect to be performed during the Test Analysis and Design phase of the Fundamental Test Process?

Answer Set:

- a) Defining test objectives
- b) Reviewing the test basis
- c) Creating test suites from test procedures
- d) Analyzing lessons learned for process improvement

### Question 4

Below is a list of problems that can be observed during testing or in production. Which one of these problems is a failure?

Answer Set

- a) The product crashed when the user selected an option in a dialog box.
- b) One source code file included in the build has the FALSE version.
- c) The computation algorithm used FALSE input variables.
- d) The developer misinterpreted the requirement for the algorithm.

### Question 5

Which of the following attitudes, qualifications or actions would lead to problems (or conflict) within mixed teams of testers and developers, when observed in reviews and tests?

Answer Set:

- a) Testers and developers are curious and focused on finding defects.
- b) Testers and developers are sufficiently qualified to find failures and defects.
- c) Testers and developers communicate defects as criticism of people, not as criticism of the software product.
- d) Testers expect that there might be defects in the software product which the developers have not found and fixed.

### Question 6

Which of the following statements are TRUE?

- I. Software testing may be required to meet legal or contractual requirements.
- II. Software testing is mainly needed to improve the quality of the product.
- III. Rigorous testing and fixing of found defects could help reduce the risk of problems occurring in an operational environment.
- IV. Rigorous testing is sometimes used to prove that all failures have been found.

Answer Set:

- a) I, II and III are true; IV is false
- b) I is true; II, III, and IV are false
- c) I and III are true; II and IV are false
- d) III and IV are true; I and II are false

### Question 7

Which of the following statements correctly describes the difference between testing and debugging?

Answer Set:

- a) Testing identifies the source of defects; debugging analyzes the faults and proposes prevention activities.
- b) Dynamic testing shows failures caused by defects; debugging finds, analyzes, and removes the causes of failures in the software.
- c) Testing removes faults; debugging identifies the causes of failures.
- d) Dynamic testing prevents the cause of failures; debugging removes the failures.

### Question 8

Which of the following statements BEST describes non-functional testing?

Answer Set:

- a) Non-functional testing is the process of testing an integrated system to verify that it meets specified requirements.
- b) Non-functional testing is the process of testing to determine system compliance with coding standards.
- c) Non-functional testing is testing without reference to the internal structure of a system.
- d) Non-functional testing is testing system characteristics, such as usability, reliability, or maintainability.

### Question 9

When working with software development models, what is it important to do?

Answer Set:

- a) If needed, adapt the models to project and product characteristics.
- b) Choose the waterfall model, because it is the most proven model.
- c) Start with the V-model, and then move to either the iterative or the incremental model.
- d) Change the organization to fit the model, not vice versa.

### Question 10

Which of the following is a characteristic of good testing and applies to any software development life cycle model?

Answer Set:

- a) Acceptance testing is always the final test level to be applied.
- b) All test levels are planned and completed for each developed feature.
- c) Testers are first involved when first piece of code can be executed.
- d) For every development activity there is a corresponding testing activity.

### Question 11

Which of the following is an example of maintenance testing?

Answer Set:

- a) To test corrected defects during development of a new system.
- b) To test enhancements to an existing operational system.
- c) To handle complaints about system quality during user acceptance testing.
- d) To integrate functions during the development of a new system.

### Question 12

Which of the following statements are TRUE?

- I. Regression testing and confirmation testing are the same.
- II. Regression testing shows that all defects have been resolved.
- III. Regression testing is a good candidate for test automation.
- IV. Regression testing is performed to uncover defects as a result of changes in the software.
- V. Regression testing should not be performed during integration testing.

Answer Set:

- a) IV is true; I, II, III and V are false
- b) III is true; I, II, IV and V are false
- c) III and IV are true; I, II and V are false
- d) I, III and IV are true; II and V are false

### Question 13

Which of the following statements comparing component testing and system testing is TRUE?

Answer Set:

- a) Component testing verifies the functionality of software models, program objects, and classes that are separately testable, whereas system testing verifies interfaces between components and interactions between different parts of the system.
- b) Test cases for component testing are usually derived from component specifications, design specifications, or data models, whereas test cases for system testing are usually derived from requirement specifications, functional specifications, or use cases.
- c) Component testing only focuses on functional characteristics, whereas system testing focuses on functional and non-functional characteristics.
- d) Component testing is the responsibility of the testers, whereas system testing typically is the responsibility of the users of the system.

### Question 14

Which of the following describes the main phases of a formal review?

Answer Set:

- a) Initiation, backtracking, individual preparation, review meeting, rework, follow-up.
- b) Planning, individual preparation, review meeting, rework, closure, follow-up
- c) Planning, kick off, individual preparation, review meeting, rework, follow-up
- d) Individual preparation, review meeting, rework, closure, follow-up, root cause analysis

### Question 15

Which of the review types below is the BEST option to choose for reviewing safety critical components in a software project if additionally the review must be demonstrated as a formal process based on rules and checklists?

Answer Set:

- a) Informal Review
- b) Technical Review
- c) Inspection
- d) Walkthrough

### Question 16

Which of the following statements about tool-supported static analysis is FALSE?

Answer Set:

- a) Tool-supported static analysis can be used as a preventive measure with appropriate processes in place.
- b) Tool-supported static analysis can find defects that are not easily found by dynamic testing.
- c) Tool-supported static analysis can result in cost savings by finding defects early.
- d) Tool-supported static analysis is a good way to force failures into the software.

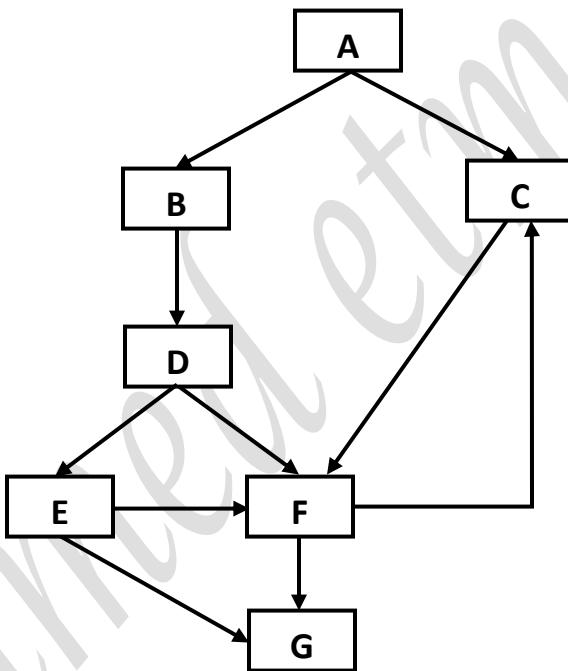
### Question 17

One of the test goals for your project is to have 100% decision coverage. The following three tests have been executed for the control flow graph shown below.

Test\_01 covers path: A, B, D, E, G

Test\_02 covers path: A, B, D, E, F, G

Test\_03 covers path: A, C, F, C, F, C, F, G



Which of the following statements related to the decision coverage goal is TRUE?

Answer Set:

- a) Decision D has not been tested completely.
- b) 100% decision coverage has been achieved.
- c) Decision E has not been tested completely.
- d) Decision F has not been tested completely.

### Question 18

A defect was found during testing:

While receiving customer data from a server the system crashed. The defect was fixed by correcting the code that checked the network availability during data transfer. The existing test cases covered 100% of all statements of the corresponding module. To verify the fix and to ensure more extensive coverage, some new tests were designed and added to the test suite and executed.

Which of the following test types should be used in this scenario?

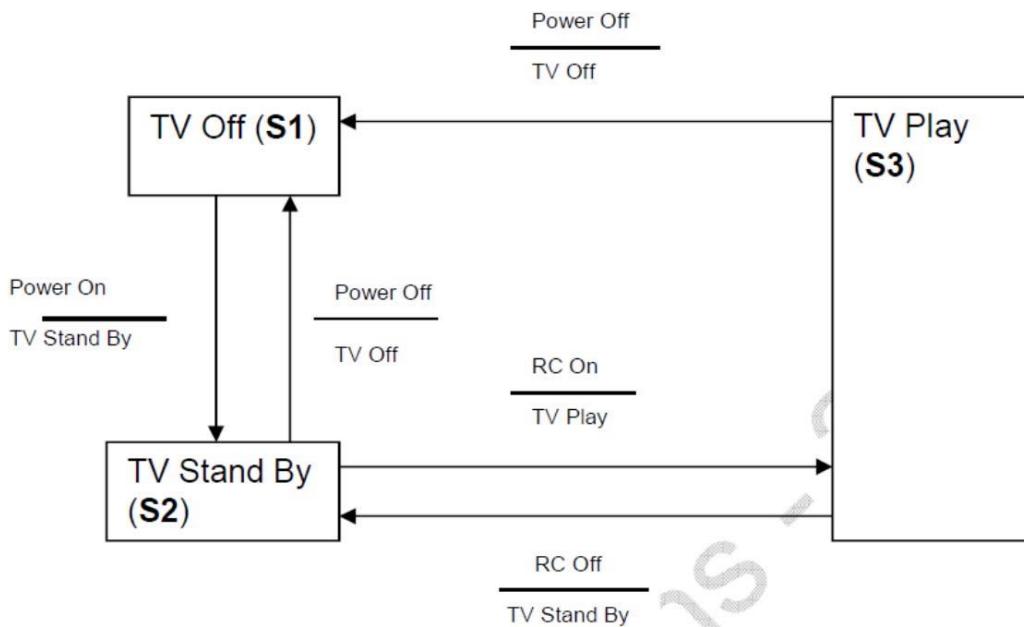
- I. Functional testing
- II. Structural testing
- III. Re-testing
- IV. Performance testing

Answer Set:

- a) I and II are used, but not III and IV
- b) I and III are used, but not II and IV
- c) I, II and III are used, but not IV
- d) II, III and IV are used, but not I

## Question 19

Which of the following statements about the given state transition diagram and the table of test cases is TRUE?



Test Case	1	2	3	4	5
Start State	S1	S2	S2	S3	S3
Input	Power On	Power Off	RC On	RC Off	Power Off
Expected output	TV Stand By	TV Off	TV play	TV Stand By	TV Off
Final State	S2	S1	S3	S2	S1

Answer Set:

- a) The given test cases can be used to derive both valid and invalid transitions in the state transition diagram.
- b) The given test cases represent all possible valid transitions in the state transition diagram.
- c) The given test cases represent only some of the valid transitions in the state transition diagram
- d) The given test cases represent sequential pairs of transitions in the state transition diagram

### Question 20

Which of the following statements for the equivalence partitioning test techniques are TRUE?

Equivalence partition testing ...

- I. Divides possible inputs into classes where all elements are expected to cause the same behavior.
- II. Uses both valid and invalid partitions.
- III. Must include at least two values from every equivalence partition.
- IV. Can be used only for testing equivalence partition inputs from a Graphical User Interface.

Answer Set:

- a) I, II, and IV are TRUE; III is FALSE
- b) I is TRUE; II, III and IV are FALSE
- c) II and III are TRUE; I and IV are FALSE
- d) I and II are TRUE; III and IV are FALSE

### Question 21

Which of the following options lists techniques categorized as Black Box design techniques?

Answer Set:

- a) Equivalence Partitioning, Decision Table testing, State Transition testing, and Boundary Value Analysis
- b) Equivalence Partitioning, Decision Table testing, Statement Coverage, Use Case Based testing
- c) Equivalence Partitioning, Decision Coverage testing, Use Case Based testing
- d) Equivalence Partitioning, Decision Coverage testing, Boundary Value Analysis

### Question 22

An employee's bonus is to be calculated. It cannot be negative, but it can be calculated down to zero. The bonus is based on the length of employment.

The categories are: less than or equal to 2 years, more than 2 years but less than 5 years, 5 or more years, but less than 10 years, 10 years or longer. Depending on the length of employment, an employee will get different levels of bonus.

How many test cases are necessary, if only valid equivalence partitions are needed to test the calculation of the bonus?

Answer Set:

- a) 3
- b) 5
- c) 2
- d) 4

### Question 23

Which of the following statements about the benefits of deriving test cases from use cases are true and which are false?

- I. Deriving test cases from use cases is helpful for system and acceptance testing.
- II. Deriving test cases from use cases is helpful only for automated testing.
- III. Deriving test cases from use cases is helpful for component testing.
- IV. Deriving test cases from use cases is helpful for integration testing.

Answer Set:

- a) I and IV are true; II and III are false
- b) I is true; II, III, and IV are false
- c) II and IV are true; I and III are false
- d) I, III and IV are true; II is false

### Question 24

Which of the options below would be the BEST basis for testing using fault attacks?

Answer Set

- a) Experience, defect and failure data; knowledge about software failures
- b) Risk identification performed at the beginning of the project
- c) Use Cases derived from business flows by domain experts
- d) Expected results from comparison with an existing system

### Question 25

You are working on a project that has poor specifications and time pressure.

Which of the following test techniques would be the most useful approach to use?

Answer Set:

- a) Use Case Testing
- b) Statement Testing
- c) Exploratory Testing
- d) Decision Testing

### Question 26

Which of the following test techniques is a white-box technique?

Answer Set:

- a) Decision Testing
- b) Boundary Value Analysis
- c) Equivalence Partitioning
- d) State Transition Testing

### Question 27

You are testing a system that calculates the greatest common divisor (GCD) of two integers (A and B) greater than zero.

clcGCD (A, B);

The following test inputs have been specified.

Test Case	A	B
1	1	1
2	INT_MAX	INT_MAX
3	1	0
4	0	1
5	INT_MAX+1	1
6	1	INT_MAX+1

Where INT\_MAX is the largest Integer.

Which specification-based test technique would you use for test cases 1 through 6?

Answer Set:

- a) Boundary Value Analysis
- b) State Transition Testing
- c) Use Case Testing
- d) Decision Table Testing

### Question 28

A company's employees are paid bonuses if they work more than a year in the company and achieve individually agreed targets.

The following decision table has been designed to test the system:

		T1	T2	T3	T4	T5	T6	T7	T8
<b>Conditions</b>									
Cond1	Employment for more than 1 year?	YES	NO	YES	NO	YES	NO	YES	NO
Cond2	Agreed target?	NO	NO	YES	YES	NO	NO	YES	YES
Cond3	Achieved target?	NO	NO	NO	NO	YES	YES	YES	YES
<b>Action</b>									
	Bonus payment?	NO	NO	NO	NO	NO	NO	YES	NO

Which test cases could be eliminated in the above decision table because the test case wouldn't occur in a real situation?

Answer Set:

- a) T1 and T2
- b) T3 and T4
- c) T7 and T8
- d) T5 and T6

### Question 29

Which of the following BEST describes how tasks are divided between the test manager and the tester?

Answer Set:

- a) The test manager plans testing activities and chooses the standards to be followed, while the tester chooses the tools and controls to be used.
- b) The test manager plans, organizes, and controls the testing activities, while the tester specifies and executes tests.
- c) The test manager plans, monitors, and controls the testing activities, while the tester designs tests and decides about the approval of the test object.
- d) The test manager plans and organizes the testing, and specifies the test cases, while the tester prioritizes and executes the tests.

### Question 30

Which of the following can be categorized as a product risk?

Answer Set:

- a) Low quality of requirements, design, code and tests.
- b) Political problems, and delays in especially complex areas in the product.
- c) Error-prone areas, potential harm to the user, poor product characteristics.
- d) Problems in defining the right requirements, potential failure areas in the software or system.

### Question 31

Which of the following are typical exit criteria from testing?

Answer Set:

- a) Test coverage measures, reliability measures, test cost, schedule, status of defect correction and residual risks
- b) Test coverage measures, reliability measures, degree of tester independence, and product completeness
- c) Test coverage measures, reliability measures, test cost, availability of testable code, time to market, and product completeness
- d) Time to market, residual defects, tester qualification, degree of tester independence, test coverage measures and test cost

### Question 32

As a Test Manager, you have the following requirements to test:

R1 - Process Anomalies

R2 - Synchronization

R3 - Confirmation

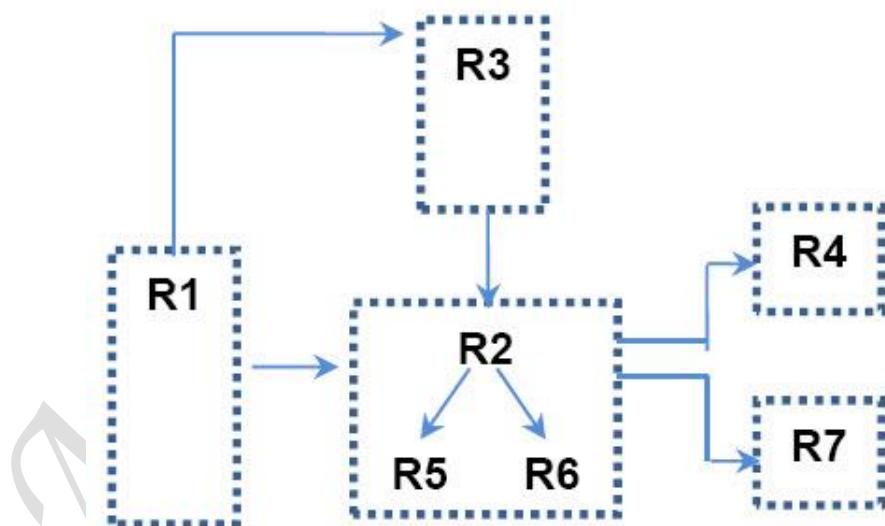
R4 - Issues

R5 - Financial Data

R6 - Diagram Data

R7 - Changes to the User Profile

The notation to indicate any requirement's logical dependencies is, for example, "R1 → R3" meaning that R3 is dependent on R1.



Which of the following options structures the test execution schedule according to the requirement dependencies?

Answer Set:

- a) R3 → R2 → R1 → R7 → R5 → R6 → R4
- b) R2 → R5 → R6 → R4 → R7 → R1 → R3
- c) R1 → R3 → R2 → R5 → R6 → R4 → R7
- d) R1 → R2 → R5 → R6 → R3 → R4 → R7

### Question 33

Which of the following is a possible benefit of independent testing?

Answer Set:

- a) More work gets done because testers do not disturb the developers all the time.
- b) Independent testers tend to be unbiased and find different defects than the developers.
- c) Independent testers do not need extra education and training.
- d) Independent testers reduce the bottleneck in the incident management process.

### Question 34

Which of the following is a project risk?

Answer Set:

- a) Skill and staff shortages
- b) Poor software characteristics (e.g. usability)
- c) Failure-prone software delivered
- d) Possible reliability defect (bug)

### Question 35

As a test manager, you are asked for a test summary report. Concerning test activities, and according to the IEEE 829 Standard, what should be the MOST important information to include in your report?

Answer Set:

- a) The number of test cases executed and their results.
- b) An overview of the major testing activities, events and the status with respect to meeting goals
- c) Overall evaluation of each development work item
- d) Training taken by members of the test team to support the test effort

### Question 36

You are a tester in a safety-critical software development project. During execution of a test, you find out that one of your test cases failed, causing you to write an incident report.

According to the IEEE Std. 829, what should you consider to be the MOST important information to include in your incident report in the context of a safety-critical development?

Answer Set:

- a) Impact, incident description, date and your name
- b) Unique ID for the report, special requirements needed and the person who caused the defect
- c) Transmitted items, your name and your feelings about the possible root cause of the defect
- d) Incident description, development environment and expected results of testing

### Question 37

From the list below, which are the recommended principles for introducing a test tool to an organization?

1. Roll out the tool to the entire organization at the same time
2. Start with a pilot project
3. Adapt and improve processes to fit the use of the tool
4. Provide training and coaching for new users
5. Let each team decide their own way of using the tool
6. Monitor that costs do not exceed initial acquisition cost
7. Gather lessons learned from all teams

Select ONE option.

Answer Set:

- a) 1, 3, 4, 5
- b) 2, 5, 6
- c) 2, 3, 4, 7
- d) 1, 6, 7

### Question 38

Which of the following BEST describes a characteristic of a keyword-driven test execution tool?

Answer Set:

- a) A table with test input data, action words, and expected results controls execution of the system under test.
- b) Tester actions are automated using a script that is rerun several times.
- c) Tester actions are automated using a script that is run with several sets of test input data.
- d) The ability to log test results, and compare them against the expected results stored in a text file.

### Question 39

Which of the following is NOT a goal of a pilot project for tool evaluation?

Answer Set:

- a) To evaluate how the tool fits with existing processes and practices.
- b) To determine use, management, storage, and maintenance of the tool and testware.
- c) To assess whether the benefits will be achieved at reasonable cost.
- d) To reduce the defect rate in the pilot project.

### Question 40

A software development and test organization would like to achieve the test efficiency improvement goals listed below.

Which would best be supported by a test management tool?

Answer Set:

- a) Enable traceability between requirements, tests, and defects (bugs)
- b) Optimize the ability of tests to identify failures
- c) Resolve defects faster
- d) Automate a selection of test cases for execution

وبكده يكون الامتحان خلص... حل الامتحان ف ورقة خارجية وتعالى راجع إجاباتك ف الصفحة الجاية...

## ISTQB Official sample exam Answers

### (Version 2.9)

هنا هنلوف الأسئلة بتتحل إزاي عشان نستفيد كلنا، وبعد كده ف الامتحانات الجاية ه تكون الإجابات موجودة  
علطول من غير شرح ... يلا بينا.

#### Question 1

ف السؤال ده بيقولك إيه أفضل جملة بتوصف واحدة من الـ 7 قواعد الأساسية للتيس، فالإختيارات اللي عندي:

- (a) غلط طبعاً لأن exhaustive testing is impossible سواء كان manual أو automated.
- (b) غلط طبعاً لنفس السبب بتاع a.
- (c) صح عشان نفس نفس الـ principle.
- (d) غلط عشان testing shows presence of defects يعني يقدر يثبت إن الـ defects موجودة لكن مايقدرش يثبت إن مافيش defects خالص، وف الاختيار قابل إنه يقدر يثبت إن مافيش defects خالص.

#### Question 2

بيقولك ف السؤال ده إيه أكثر test team لـ valid goal، فالإختيارات اللي عندي:

- (a) غلط عشان الـ component (unit) testing مش جزء من الـ system testing.
- (b) صح لأن ده واحد من الأهداف الرئيسية للـ testing team.
- (c) غلط عشان exhaustive testing is impossible.
- (d) لأن عشان تقول إن الـ failures الباقية مش هتسبب defects لازم الأول تحدد كل defects وده طبعاً مستحيل.

### Question 3

بيقولك ف السؤال إيه من التاسكات دى بتعملى أثناء مرحلة الـ **Test Analysis and Design** فالاختيارات اللي عندي:

- (a) غلط عشان تعريف الـ **test objectives** بيتم فى مرحلة الـ **test planning**.
- (b) صح عشان مراجعة الـ **test basis** بتتم أثناء مرحلة الـ **test analysis and design**.
- (c) غلط عشان عمل الـ **test suites** بيتم أثناء مرحلة الـ **test implementation and execution**.
- (d) غلط عشان تحليل الدروس المستفادة لتحسين العملية بيتم فى مرحلة الـ **test closure activities**.

الكلام ده كله موجود فى النقطة 1.4 ف المنهج.

### Question 4

مدلياك شوية مشاكل وبيقولك إيه منهم اللي نقول عليه إنه **failure**، فتعالي نشوف الاختيارات

- (a) صح لأن الـ **product** بتعنى ضرب لما اخترت اختيار ف الـ **dialog box**، وده بيتطابق تماما مع حقيقة الـ **failure** إنها بتطلع لليوزر نتيجة **defect**، وده اللي حصل بالظبط.
- (b) علط لأن ده مش شرط يسبب **failure** واضح وصريح، زي مثلا لو النسخة الجديدة من الكود فيها تغيير ف الـ **comments** اللي ع الكود بس، فكده ما فيهش أى حاجة إتأثرت معايا.
- (c) غلط برضه، مش شرط إنه يسبب **failure** ف العملية، بمعنى مثلا إن لو الـ **variable** بتعنى عملته بنوع **float** بدل **int** فكده مش هيأثر على عملية الجمع مثلا أو يطلعى نتيجة غلط، ممكن تحصل لكن ده مش أصح اختيار موجود عندي (مش أصح من a).
- (d) غلط برضه، النوع ده من الغلطات ممكن مايسببش **failure** لو الـ **algorithm** ده ماحدش استعمله مثلا، فده برضه مش أصح اختيار عندي.

افتكر إننا دايما بنختار أصح حاجة موجودة عندي... وف حالة الـ **a** الـ **failure** عندى واضح وصريح على عكس باقى الاختيارات اللي ممكن تحصل وممكن لا، فاحنا بنختار الأصح.

### Question 5

بيقولك إيه اللي ممكن يعمل مشكلة بين **team** عبارة عن مجموعة **developers** و **testers**، فالاختيارات اللي عندى:

- (a) غلط، الـ **action** ده مايعلمش مشاكل بين الديفلوبر والتيسير.
- (b) برضه غلط... كون إنك تبقى مؤهل إنك تطلع **defect** عمرها ماكانت مشكلة بين الديفلوبر والتيسير.
- (c) صح، لما التيسير والديفلوبير بيأخذوا الموضوع بشكل انتقاد شخصى بتحصل ساعتها مشاكل كتير بينهم.
- (d) غلط، لأن التيسير شغلاته إنه يلاقي الـ **defects** اللي الديفلوبير مااكتشفهاش.

### Question 6

بيقولك إيه اللي من الاختيارات دى صح وإيه اللي غلط، الاختيارات هى:

- I. الـ **software testing** ممكن يبقى مطلوب عشان يحقق **legal requirements** أو **contractual requirements**... الجملة دى صح طبقا للنقطة 2.2.4 ف المنهج، فنعلم عليهم ونشيلها على جنب.
- II. الـ **software testing** مطلوب أساسا عشان يحسن الـ **quality** بتاعة المنتج، الكلام ده صح برضه، نعلم ع الاختيار ده ونشيله على جنب.
- III. الـ **risk** حدوث **defects** وحل الـ **rigorous testing** المكتشفة ممكن يساعد فى تقليل المشاكل فى الـ **operational environment**... برضه الكلام ده صح طبقا للنقطة 1.2.
- IV. الـ **rigorous testing** أحيانا بيستخدم لإثبات إننا لاقيينا كل الـ **failures**... الكلام ده غلط لأن **testing show presence of defects**، فيقدر يثبت إن فيه **defects** ف السيستم، إنما مايقدرش يثبت إن السيستم خالى من الـ **defects**.

فكده الاختيارات الصح عندى هما I و II و III و IV بس هى اللي غلط، عشان كده الإجابة الصح هى

.a

معلش الأسئلة اللي من النوع ده بتبقى رخمة شوية ومحاجة تركيز، فحاول تركز معها على قد ماتقدر.

### Question 7

بيقولك إيه من الاختيارات اللي بيوصف الفرق بين الـ **testing** والـ **debugging** بشكل صحيح، فالاختيارات هي:

- (a) غلط، لأن التيسينج وظيفته يطلع الـ **defects** مش يعرف مصدرها.
- (b) صح، لأن الـ **dynamic testing** بيلاقى الـ **failures** الناتجة عن الـ **defects**، إنما الـ **debugging** بيلاقى أسباب الـ **failures** ويحللها ويعالجها.
- (c) غلط، التيسينج مش بيعالج الـ **faults** (هو بيلاقيها بس إنما مش بيعالجها).
- (d) غلط، لأن الـ **dynamic testing** مش بيمنع أسباب حدوث الـ **failures** (هو بيلاقيها بس).

### Question 8

بيقولك إيه أحسن اختيار بيوصف الـ **non-functional testing**، الاختيارات اللي عندى:

- (a) غلط لأن ده تعريف الـ **System testing**
- (b) غلط لأن ده وظيفة الـ **tool** بتاعة الـ **static analysis**
- (c) غلط برضه لأن ده الـ **black box testing**... ممكن يكون صح بس مش ده أصح اختيار.
- (d) صح، ده أدق حاجة توصف الـ **non-functional testing**

### Question 9

بيقولك إيه المهم اللي بيتعمل واحنا شغالين ف الـ **software development models**؟.. الاختيارات هي:

- (a) صح لأن الـ **models** بتدى خطوط عريضة مش خطوات دقيقة لازم تتبعها بالحرف.
- (b) غلط لأن الـ **waterfall model** هو الـ **model** من ضمن اكتر من ممكن الـ **team** يختاره وممكن لا.
- (c) غلط لأن الـ **V-model** مش **compatible** مع الـ **iterative models**
- (d) غلط طبعاً... هنغير المنظمة كلها لمجرد إن الـ **model** مش مناسب؟ مش منطقى خالص... الـ **model** هو اللي يتغير عشان يناسب شغل الـ **organization** مش العكس.

### Question 10

بيقولك إيه من الاختيارات يعتبر من خصائص الـ **good testing** ويتطلب في أي **SDLC model**؟  
الاختيارات اللي عندي:

- (a) غلط... الكلام ده صح بس ف حالة إن الـ **project** فيه **acceptance tests** ومش كل المشاريع فيها الـ **level** ده (راجع النقطة 2.1 ف المنهج).
- (b) غلط... فيه حالات بنكون فيها مش محتاجين بعض الـ **test levels** زى مثلا لو **3<sup>rd</sup> party** بيدىنى كود معين، ف الحالة دى مش محتاج **component testing**.
- (c) غلط برضه... الـ **testers** لازم يكونوا موجودين من قبل الكود كمان ف الأول خالص (من أول **requirements reviews**) (راجع النقطة 1.4.2 ف المنهج).
- (d) صح دى... طبقا للنقطة 2.1.3 ف المنهج، فمن ضمن خصائص الـ **good testing** إن كل **testing activity** يقابلها **development activity** موازى له.

### Question 11

بيقولك إيه من الاختيارات يعتبر مثال ع الـ **maintenance testing**، الاختيارات هي:

- (a) غلط... الـ **maintenance testing** ما بيتمش على سистем جديد، بيتم بعد ما بيتمل للعميل.
- (b) صح... الـ **maintenance testing** هو فعل بيتسيت التحسينات الجديدة على سистем موجود بالفعل (راجع النقطة 2.4 ف المنهج).
- (c) غلط... الـ **maintenance testing** ما بيتمش أثناء الـ **acceptance testing**... ده مرحلة بعده.
- (d) غلط برضه... عملية الـ **integration** بين الـ **components functions** (مش الـ **functions**) مش أصلًا **testing activity**

### Question 12

بيقولك إيه اللي من الاختيارات دي صح؟

I. غلط لأن الـ regression testing هو إننا بنتيست تأثير التعديل الجديد ده على المربطة بيها، أما الـ confirmation testing بتأكد منه إن الـ defect components اتحلت خلاص.

II. غلط برضه عشان ده تعريف الـ confirmation testing (re-testing) صح... الـ regression testing بيعمل على أجزاء اتهست قبل كده 100 مرة، عشان كده automation هيخدمني ف النقطة دي (راجع النقطة 2.3.4)  
III. الـ regression testing هى وظيفة الـ test levels ممكن يتعملى على كل الـ structural testing والـ non-functional testing والـ functional testing  
IV. صح... دى وظيفة الـ regression testing أساسا.  
V. غلط... الـ regression testing يمكن يتعملى على كل الـ test levels وبيشمل (راجع على النقطة 2.3.4) (white box)

فكله بقى عندنا III و IV بس هما اللي صح والباقي غلط

عشان كده الإجابة الصح هي C

### Question 13

بيقولك إيه الصح ف الاختيارات دي في المقارنة بين الـ component testing والـ system testing؟، الاختيارات هي:

(a) غلط... اللي بيتبيست الـ interactions بين الـ components والـ interfaces بينهم هو integration testing

(b) صح... راجع على النقطة 2.2.1 (component testing) والنقطة 2.2.3 (system ) (testing)

(c) غلط... الـ functional characteristics مش بيركز بس ع الـ component testing (راجع النقطة 2.2.1)

(d) غلط... الـ component testing مسئولية الديفلوبير والـ system تيسينج مسئولية الـ testers

### Question 14

بيقولك إيه الاختيار اللي بيوصف المراحل الأساسية من ال **formal review** ؟

- (a) غلط... مافيش ف ال **formal review** حاجة اسمها **initiation** ولا **backtracking**.
- (b) غلط برضه... العملية ناقصة ال **Kick-off** وال **closure** مش جزء من عملية ال **review** (راجع النقطة 3.2.1).
- (c) صح موجودين كلهم وبالترتيب الطبيعي بتاعهم (فاكر لما قلتاك احفظتهم بالترتيب عشان بيجوا بالحرف ف الامتحان؟)
- (d) غلط... ال **planning** وال **kick-off** وال **closure** ناقصين، كمان ال **root cause** مش جزء من ال **analysis** .

### Question 15

بيقولك إيه أفضل أوبشن ف ال **review types** الجايين نختاره في مراجعة **formal process** فى **software project components** وقائم على **checklists rules** ؟

الاختيارات اللي عندي: **technical review** و **informal review** و **inspection**، فأكتر واحد **formal** فيه هو ال **walkthrough**

عشان كده الإجابة الصح هي C

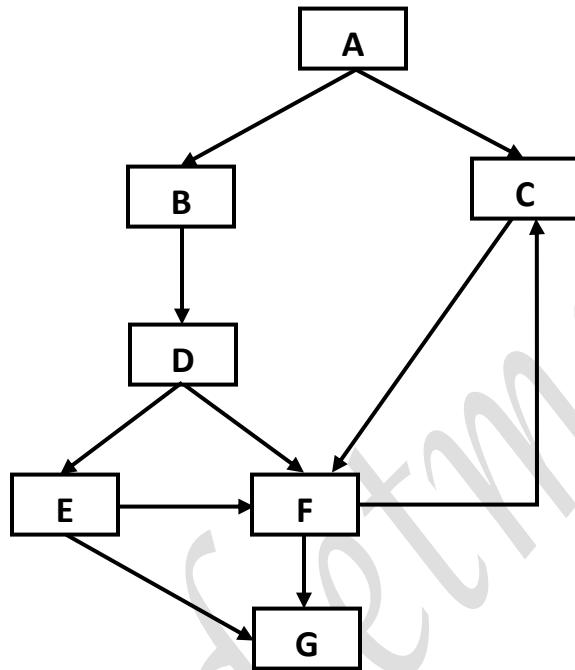
### Question 16

بيقولك إيه الجملة الغلط بخصوص ال **tool-supported static analysis** ف الاختيارات دي:

- (a) غلط... الجملة كلها صح (راجع النقطة 3.3).
- (b) غلط برضه... الجملة كلها صح برضه (احنا بندور ع الغلط مش الصح)
- (c) غلط... الجملة دى برضه صح.
- (d) صح... الجملة دى غلط، ال **static analysis** مافيهوش **failures** لأن الكود مايحصلوش **run** أصلًا.

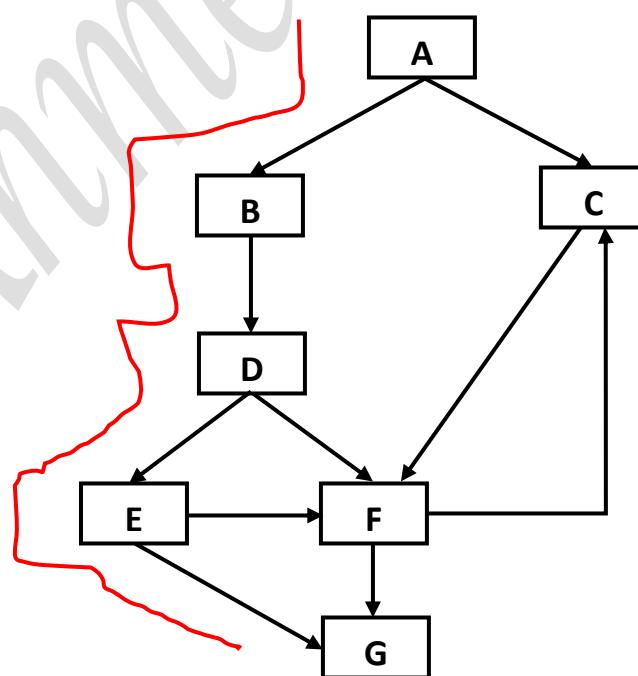
### Question 17

بيقولك ف الشكل ده عايز 100% decision coverage

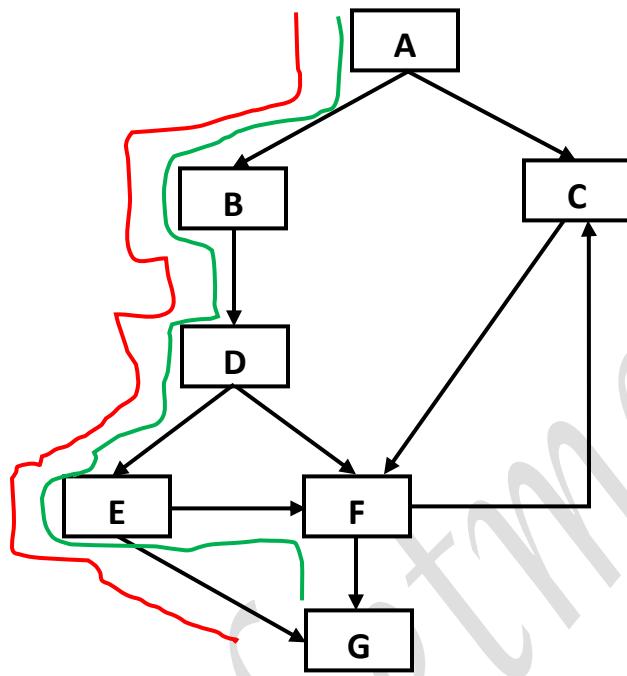


ومديلاك 3 tests

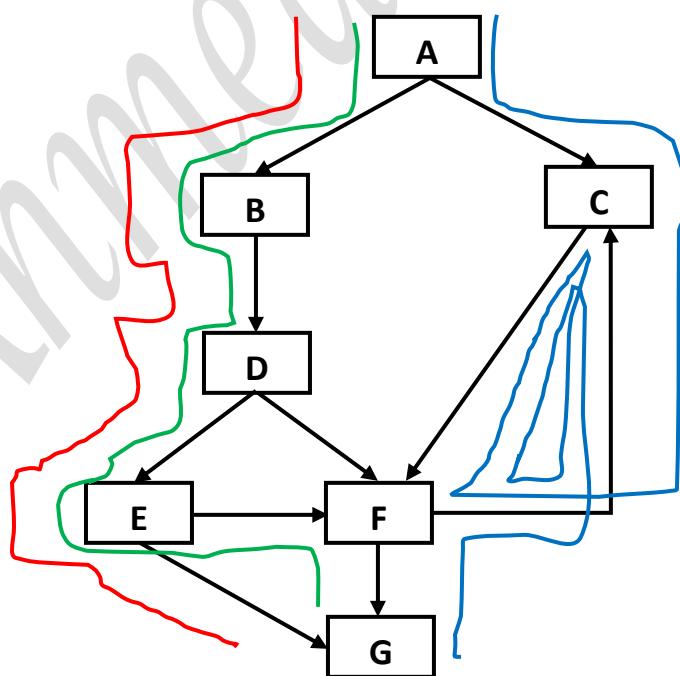
أول تيست ماشي بال path ...A, B, D, E, G ده: فرسمه بالقلم:



ثانى تيست ماشى بال path  $A, B, D, E, F, G$  فرسمه بالقلم:



ثالث تيست ماشى بال path  $A, C, F, C, F, C, F, G$  فرسمه بالقلم برضه:



فلو ركزنا شوية هنلاقى ال F → D هو ال Path الوحيد اللي ماراحش ليه أى test خالص.

عشان كده الإجابة الصحيحة هي a

### Question 18

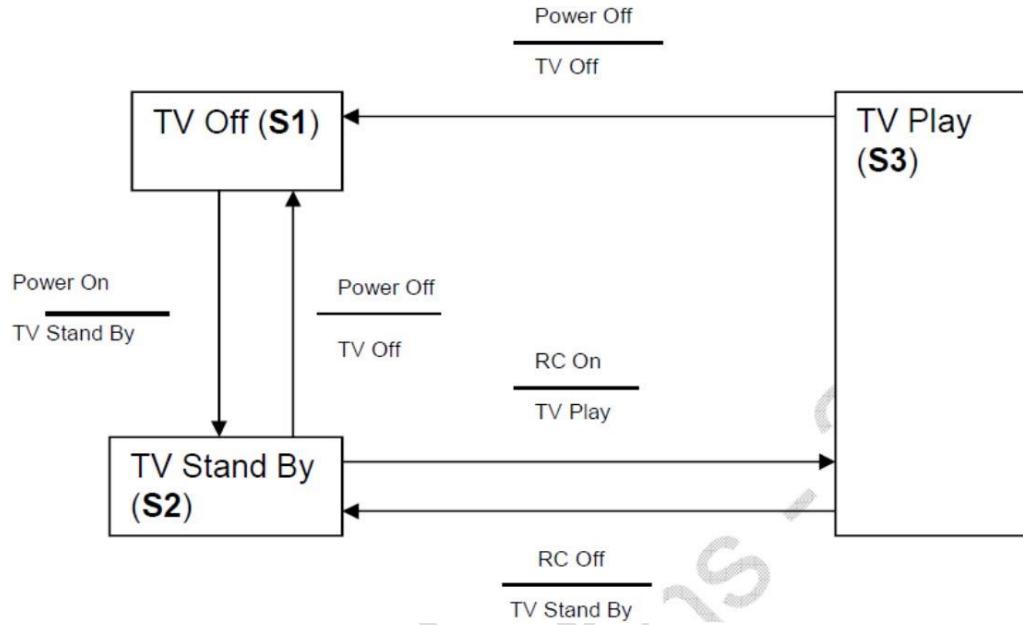
بيقولك إن أثناء عملية التيسينج حصل defect وهى إن السيستم بيقع لما بيتقبل ال data customer من سيرفر، فحلوا ال network defect دى عن طريق تصحيح الكود اللي بيسيك على ال availability أثناء نقل البيانات، وعشان تتأكد إن ال fix ده تمام ولزيادة ال coverage لازم نعمل شوية tests جديدة، فإيه اللي ممكن نستعمله عشان نستخدمه ف السيناريو ده؟

- I. Functional testing : بالتأكيد طبعا... كل اللي بيحصل ده جوه الكلام جزء منه.
- II. Structural testing : اللي هو ال white be ... أه طبعا لأنه عمال بيقول كود ونزوود ال coverage وكلام كله white box .
- III. Re-testing : بالتأكيد هنستخدمه عشان تتأكد إن ال fix ده تمام.
- IV. Performance testing : أنا هنا مش هستخدمه عشان ال bug ماكانتش خاصة بال performance خالص إنما كانت مشكلة ف الكود.

عشان كده الإجابة الصحيحة هي C

## Question 19

مدينى state transition diagram & table اللى هما دول:



Test Case	1	2	3	4	5
Start State	S1	S2	S2	S3	S3
Input	Power On	Power Off	RC On	RC Off	Power Off
Expected output	TV Stand By	TV Off	TV play	TV Stand By	TV Off
Final State	S2	S1	S3	S2	S1

الحالات ف الجدول:

Test case 1:  $S1 \rightarrow S2$ , Test case 2:  $S2 \rightarrow S1$ , Test case 3:  $S2 \rightarrow S3$ .

Test case 4:  $S3 \rightarrow S2$ , Test case 5:  $S3 \rightarrow S1$

تعالوا بقى نشوف الاختيارات اللى عندى بتقول إيه:

- (a) غلط... لأن الـ test cases اللى عندى بتغطى الـ valid transitions بس.
- (b) صح... لأن فعلا الـ test cases مغطية كل الـ transitions الممكنة.
- (c) غلط... لأن الـ test cases بتغطى كل الـ valid transitions مش بعضها.
- (d) غلط برضه لأن ما فيه transition حصل مرتين... كله حصل مرة واحدة بس.

### Question 20

بيقولك إيه من الاختيارات الجاية صح عن ال equivalence partitioning test technique

فال equivalence partitioning testing

- I. بيقسم ال inputs الممكنة إلى classes بحيث كل ال elements فيها متوقع منها أنها تسبب نفس ال behavior ... دى صح ... شيلها على جنب.
  - II. بيستخدم ال valid & invalid partitions ... دى صح برضه ... شيلها على جنب برضه
  - III. لازم يشمل ع الأقل قيمتين من كل equivalence partition ... دى غلط .. قيمة واحدة كافية ف التكنيك ده.
  - IV. ممكن يستخدم بس فى تيست ال equivalence partition inputs من ال GUI ... ده غلط ... أو ممكن أستقبل input من ال GUI ... بس ده مش الطريقة الوحيدة.
- وبالتالى عندنا I و II بس اللي صح والباقي غلط، فالاختيار الصح عندنا هو D.

### Question 21

بيقولك إيه اللي من الاختيارات يعتبر Black Box design techniques، فالاختيارات هي:

- (a) black box صح ... كلهم
- (b) Statement Coverage ... ال white box ده غلط ... black box مش
- (c) Decision Coverage ... ال white box برضه غلط ... black box
- (d) Decision Coverage ... ال white box برضه غلط ... black box

**Question 22**

بيقولك فيه نظام **bonus** بتحسب للموظفين بناء على المدة اللي قضوها في الشركة بمستويات مختلفة، ومش ممكن يكون الـ **bonus** ده بالسالب، بس ممكن يكون **zero**، فالـ **categories** هى: أقل من أو يساوى سنتين – أكثر من سنتين وأقل من 5 سنين – 5 سنين وأقل من 10 سنين – 10 سنين أو أكثر.

فبيقولك لو هتستخدم الـ **Equivalence Partitions** بس وانت شغال على حوار زى كده كام تيست كيس تحتاجها لو محتاجين الـ **valid equivalence partitions** بس.

مبئيا لو عايز تحل أى سؤال **Boundary Value Analysis** أو **Equivalence Partition** لازم الأول ترسم المجموعات بتاعة الـ **inputs** ع الورق، ففى السؤال ده هنرسم المجموعات بتاعتتنا فهتكون كده:



فعندنا هنا 4 مجموعات والسائلب مش معانا لأنه **EP** بناخد قيمة واحدة بس من كل مجموعة فيكون عدد الـ **test cases** اللي تحتاجها فالتكنيك ده .4 test cases عشان كده الإجابة الصح هي D.

**Question 23**

مدليك شوية اختيارات عن مزايا انك تطلع الـ **use cases** من الـ **test cases** وبيقولك مين فيهم صح ومين فيهم غلط، فالاختيارات هى:

- I. استخلاص الـ **test cases** من الـ **use cases** مفيد فى الـ **system & acceptance testing** ... دى صح ... شيلها على جنب.
- II. استخلاص الـ **test cases** من الـ **use cases** مفيد بس فى الـ **automated testing** ... وده طبعاً غلط لأن الـ **use cases** ممكن تتنفذ **manually** مش شرط **automation** بس.
- III. استخلاص الـ **test cases** من الـ **use cases** مفيد فى الـ **component testing** ... ده غلط لأن الـ **use case** بنستخدمها ف السيستم كله مش جزء منه.
- IV. استخلاص الـ **test cases** من الـ **use cases** مفيد فى الـ **integration testing** ... ده صح ... شيلها على جنب

وب kedde يكون الـ I والـ IV صح والباقي غلط... عشان كده الإجابة الصح هي a (راجع النقطة 4.3.5 فـ المنهج).

### Question 24

بيقولك إيه من الاختيارات دى هيكون أفضل basis ف التيستينج باستخدام ال fault attacks .

- (a) صح، كل دول هيساعدونى ف ال fault attack (راجع على 4.5 ف المنهج).
- (b) غلط... ال risk بيعرفنى ع المناطق اللي أركز عليها ف السيسنتم مش بيقولى أعمل تيست ع الجزء ده إزاى.
- (c) غلط برضه... ال business flows بيقولى ع السيناريو بناء السيسنتم كله ومش بيقولى على نقط الضعف اللي ف السيسنتم، فكده مش هيساعدنى.
- (d) غلط برضه... لما عمل مقارنة بين ال ER من سيسنتم موجود والسيسنتم بناعى فكده أنا بشوف التيست بناعى صح ولا مضرب، إنما مش هييفينى حاجة ف ال fault attack .

### Question 25



بيقولك إنك شغال على poor project فيه time pressure specifications و specifications تكينك أشتغل بيها.

طبعاً من غير تفكير ال Exploratory Testing ، ده غير إن كل ال techniques الثانية تحتاج وقت ف التنفيذ، فالإجابة الصح هي C .

### Question 26

بيقولك إيه ال white-box technique من الاختيارات، طبعاً ما فيه من الاختيارات إلا ال Decision Testing (Decision Coverage) هو الوحيدة ال black-box white-box والباقي ، عشان كده الإجابة الصح هي a .

### Question 27

مدليك جدول لحساب العامل المشترك الأكبر وبيقولك إيه ال specification-based technique اللي هنستخدمه معاه، فالاختيارات اللي عندنا:

- (a) صح... لأننا لو بصينا ف الجدول اللي ف السؤال هنلاقى ال inputs بتاعته الموجودة فى الجدول كالاتى (0, 1, INT\_MAX+1, INT\_MAX).
- (b) غلط... ده مش State Transition table خالص
- (c) غلط... مش راكب على Use Case خالص.
- (d) غلط برضه... ده مش Decision Table .

**Question 28**

بيقولك إن فيه شركة بتدفع **bonus** للموظفين بتوغها اللي اشتغلوا فيها أكثر من سنة وحققوا الـ **agreed** لكل واحد فيهم، ومدينى الـ **targets** ده:

		T1	T2	T3	T4	T5	T6	T7	T8
Conditions									
Cond1	Employment for more than 1 year?	YES	NO	YES	NO	YES	NO	YES	NO
Cond2	Agreed target?	NO	NO	YES	YES	NO	NO	YES	YES
Cond3	Achieved target?	NO	NO	NO	NO	YES	YES	YES	YES
Action									
	Bonus payment?	NO	NO	NO	NO	NO	NO	YES	NO

وبقولي إيه الـ **test cases** اللي هستبعدها عشان لايمكن تحصل ف الحقيقة.

تعالى نشوف إيه الدنيا واحدة واحدة... بيقولك إنك عشان تأخذ **bonus** لازم تحقق الشروط دي:

1- لازم تكون شغال أكثر من سنة.

2- لازم الـ **target** بتاعك الشركة توافق عليه

3- لازم تحقق الـ **target** اللي الشركة وافقت عليه ليك.

ففي الجدول ده لو ركزنا هنلاقى إن T5 و T6 لايمكن يتحققوا أبدا... إزاى حقق target الشركة أصلاً مش موافقة عليه؟ لازم يتوافق عليه الأول.

وبالتالي الإجابة الصحيحة هي d.

**Question 29**

بيقولك إيه أفضل اختيار بيوصف إزاى التاسكات بتتقسم بين الـ **test manager** والـ **tester**، فالاختيارات اللي عندي:

(a) غلط... لأن اختيار الـ **tools** من ضمن مهام الـ **test manager** برضه (كمـل قرایة الكلام)

(b) صح

(c) غلط... التـester مابيقررش حاجة بخصوص الـ **test object approval** ... ده من ضمن مهام الـ **test manager**

(d) غلط... الـ **test cases** مابيحددش الـ **test manager** ... ده من اختصاص التـester.

عشان تتأكد أكثر بصبره ع النقطة 5.1.2 ف المنهج

### Question 30

بيقولك إيه من الاختيارات ممكن تعتبره **product risk**؟  
الـ **product risk** هى أى حاجة تأثر على كفاءة الـ **product** اللي بطلعه من عندى لما يشتغل عند العميل، فتعالوا نشوف الاختيارات.

- .project risk ... low quality requirements (a)
- .project risks ... كل الـ items ف الاختيار ده هما (b)
- .product risks ... كل الـ items هما (c)
- .project risk requirements ... مشاكل الـ project تعتبر (d)

راجع تانى ع الفرق بين الـ **product risks** والـ **project risks** ف النقطة 5.5.1 و 5.5.2 ف المنهج.

### Question 31

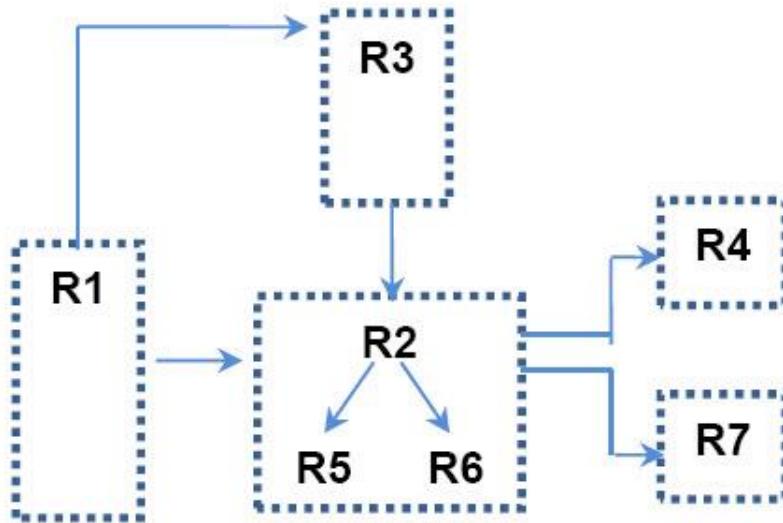
بيقولك إيه ف الاختيارات يعتبر **exit criteria**، فالاختيارات هى:

- .exit criteria ... كلهم يعتبروا (a)
- .exit criteria ... Degree of tester independence ... مش (b)
- exit entry criteria ... Availability of testable code ... مش (c)  
.criteria
- .tester qualification ... Degree of tester's independence ... مش (d)  
.exit criteria

عشان تتأكد راجع ع النقطة 5.2.4 ف المنهج.

## Question 32

مدينى الشكل ده:



وبيقولى إن السهم لو رايح على حاجة تبقى معتمدة على اللي طالع من عندها السهم، يعني لو قلت مثلاً إن  $R1 \rightarrow R3$  ده معناه إن  $R3$  معتمدة على  $R1$ ، وبيقولى إيه من الاختيارات أقدر أعمل عليه **test execution schedule**، فالاختيارات اللي عندي:

- (a) غلط... كلهم معتمدين على  $R1$  وعشان كده لازم التيسٍت يبدأ بال  $R1$ .
- (b) غلط برضه لأن التيسٍت مابيبدأش بال  $R1$ .
- (c) صح... كلهم ماشيين مظبوط والتسٍست بادئ بال  $R1$ .
- (d) غلط... ال  $R2$  مابتروحش لل  $R3$ ... ال  $R3$  معتمدة على ال  $R2$ ، عشان كده ال  $R3$  لازم يتعمل عليها تيسٍست قبل ال  $R2$ .

## Question 33

بيقولك إيه ال **benefit** من ال **independent testing** من الاختيارات دي:

- (a) غلط... ال **independence** مش معناه تقليل التعاون بين الديفلوبير والتيسٍست.
- (b) صح... ده سبب من أسباب ال **independence** (راجع النقطة 5.1.1 ف المنهج).
- (c) غلط... ال **testers** بيتحتاجوا تدريب وتعليم.
- (d) غلط... مافيش أى ترابط بين كونهم **independent testers** وعنق الزجاجة ف عملية **incident management**.

### Question 34

بيقولك إيه من الاختيارات يعتبر **project risk**، فالاختيارات هي:

- (a) صح... ده من الـ **project risks**.
- (b) غلط... ده **product risk** (راجع النقطة 5.5.2 فـ المنهج).
- (c) غلط... ده **product risk** (راجع النقطة 5.5.2 فـ المنهج).
- (d) غلط... ده **product risk** (راجع النقطة 5.5.2 فـ المنهج).

### Question 35

بيقولك إنك لك **test manager** مطلوب منك **test summary report** بتلخص فيه الـ **activities**، فطبقاً للـ **IEEE 829 Standard** إيه أكثر معلومة مهمة تكتب في التقرير ده؟  
الاختيارات عندى:

- (a) غلط... أه عدد الـ **test cases** اللي نفذتها ونتائجها بكتبهم في **test summary report** لكن ده مش أهم حاجة موجودة في الاختيارات اللي عندى (إحنا بنأخذ أهم حاجة).
- (b) صح... ده أهم حاجة موجودة عندى (راجع النقطة 5.3.2 فـ المنهج).
- (c) غلط... تقييم كل **development work item** مابكتوش في **test summary report**.
- (d) غلط... الـ **training** مالوش علاقة بالـ **test summary report**.

### Question 36

بيقولك إنك تيستر وعملت تيست على **safety-critical software**، وانت بتنفذ تيست كيس طلت **failed**، وبالتالي هتعمل **incident report** بالـ **bug** اللي لاقيتها، فيقولك إيه أهم حاجة ممكن تكتبها في الـ **report** ده من الاختيارات اللي هي:

- (a) صح... تأثير الـ **bug** أهم شئ ممكن يكتبه في **report**.
- (b) غلط... المعلومات دي لازم تكتب في **incident report** لكنها مش أهم من التأثير.
- (c) غلط... الـ **root cause** ماينفعش يكتبه فيه مشاعر التيستر تجاه الـ **incident report**.
- (d) غلط... المعلومات دي لازم تكتب في **incident report** لكنها مش أهم من التأثير.

### Question 37

مدیلک 7 اختیارات و بیقولک ایه ال **recommended principles** لما أحب أقدم **test tool** لمنظمة معینة، فالاختیارات هی:

1. غلط... ال **recommended deployment** ف الأول هو إننا نعمل على خفيف قبل مانقدمه للمنظمة كلها (راجع النقطة 6.3 ف المنهج)

2. صح... نفس السبب بناءً على النقطة اللي فانت.

3. صح... الكلام ده موجود بالنص ف النقطة 6.3 ف المنهج  
fits with existing processes and practices, and determine what would need to change"

4. صح... Provision of training is one of the success factors for deployment (النقطة 6.3).

5. غلط... لو سينينا كل واحد يقرر إزاي يستخدم ال **tool** هيحصلفوضى، "Defining usage guidelines" تبقى عامل من عوامل النجاح لل **deployment** (راجع النقطة 6.3).

6. غلط... تكلفة عمل **tool** **deploying** أكتر من مجرد تكلفة شراء ال **tool**، فعدم إدراك الجزء ده يصنف كواحد من ال **risks** المرتبطة بال **tool deployment** (راجع النقطة 6.2).

7. صح... استخلاص الدروس المستفادة من كل ال **teams** هو عامل من عوامل نجاح **deployment**.

عشان كده الاختيار الصح هو C.

### Question 38

بیقولک ایه أفضل اختیار بیوصف واحد من خصائص ال **keyword-driven test execution**؟، فالاختیارات عندي:

"In a keyword-driven testing approach, the spreadsheet ... (a) contains keywords describing the actions to be taken (also called action words), and test data" (Section 6.2.2).

(b) غلط... الكلام ده بیوصف ال **scripted test automation** (راجع على 6.2.2).

(c) غلط... الكلام ده بیوصف ال **data-driven test automation** (راجع على 6.2.2).

(d) غلط... ده وصف جزء من اللي بيعمله ال **test automation framework** (راجع على 6.1.6 ف المنهج).

### Question 39

بيقولك إيه من الاختيارات مش هدف لل pilot project اللي بنستخدمه ف تقييم ال tool ، فالاختيارات اللي عندي بتقول إن كلهم صح ماعدا D فتكون هي الإجابة (راجع ع النقطة 6.3).

### Question 40

بيقولك إن فيه software development and test organization عايزه تحقق أهداف تحسين test efficiency اللي موجودين ف الاختيارات، فإيه هيكون أفضل واحد فيهم ال management tool تدعمه؟

الاختيارات اللي عندي:

- (a) صح... لأن ال traceability بين ال requirements والتسينج ده وظيفة ال management tool (راجع ع النقطة 6.1.3 ف المنهج).
- (b) غلط... لأن ده مش ممكن مع ال test management tools (راجع على 6.1.6).
- (c) غلط... لأن ال defects أساساً مش بيتحل بال test management tools (راجع على النقطة 6.1.3 ف المنهج).
- (d) غلط... لأن ال test cases مش بتدعم اختيار ال test management tools (راجع ع النقطة 6.1.6 ف المنهج).

وبكله نكون خلصنا إجابات أول امتحان... يارب تكون دلوقتي فهمت الدنيا ماشية إزاي ف الامتحانات وفكر الناس ف ال ISTQB ماشى إزاي... خدت بالك إنه ساعات بيجيب الكلام حرفياً من المنهج الرسمي؟ عشان كده قلتلك لازم تقرأ المنهج الإنجليزى عشان بيجيب منه الأسئلة ساعات بالحرف.

دلوقتي هندخل ف 3 امتحانات تانيين، بس المرة دى ه تكون الإجابات من غير شرح... أظن دلوقتي انت فهمت الدنيا ماشية إزاي... فصحح لنفسك وشوف إنت واقع ف إيه وركز عليه أكثر.

امسك ال stop watch بقى وحدد لنفسك 60 دقيقة ف كل امتحان وحاول تحل كل الأسئلة ف المدة دى عشان تبقى متعددة على ضغط الوقت لما تدخل الامتحان الحقيقي فمايكونش فيه مشكلة معاك ان شاء الله.

## ISTQB Exam 2

1. Which factors contribute to humans making mistakes that can lead to faulty software?
  - I. Setting aggressive schedule
  - II. Integrating complex systems
  - III. Allocating adequate resources
  - IV. Failing to control changes
  - A. I and II are true; III and IV are false
  - B. II and IV are true; I and III are false
  - C. I, II and IV are true; III is false
  - D. I, II and III are true; IV is false
2. How can software defects in future projects be prevented from reoccurring?
  - A. Creating documentation procedures and allocating resource contingencies
  - B. Asking programmers to perform a thorough and independent testing
  - C. Combining levels of testing and mandating inspections of all documents
  - D. Documenting lessons learned and determining the root cause of problems

3. Which of the following are USUALLY stated as testing objectives?

- I. Finding defects in the software
- II. Reducing maintenance costs
- III. Confirming that the system works
- IV. Assessing the quality of the software
- V. Meeting schedule milestones

- A. I and II
- B. I, III, and IV
- C. II, IV, and V
- D. III and IV

4. Which test approach uses all combinations of input values and preconditions?

- A. Component testing
- B. Error guessing
- C. Keyword driven testing
- D. Exhaustive testing

5. Which of the following is a KEY control task?

- A. Initiating corrective actions
- B. Determining the scope
- C. Implementing the test policy
- D. Scheduling test implementation

6. Which of the following is a MAJOR task when evaluating the exit criteria?

- A. Creating test suites and cases for efficient execution
- B. Writing a test summary report for stakeholders
- C. Handing the testware to the maintenance organization
- D. Identifying any required infrastructure and tools

7. What principles do "avoiding author bias" and "communicating problems constructively" represent?
  - A. Preventive testing and reactive testing
  - B. Experience-based testing and interoperability testing
  - C. Independent testing and good interpersonal skills
  - D. Criticism avoidance and effective relationships
8. Which test may OPTIONALY be included in the common type of the V-model?
  - A. Component (unit) testing
  - B. Acceptance testing
  - C. System integration testing
  - D. Validation and verification
9. What is the difference between component testing and integration testing?
  - A. Component testing tests interfaces; integration testing searches for defects
  - B. Component testing searches for defects; integration testing tests interfaces
  - C. Developers perform component testing; testers perform integration testing
  - D. Testers perform component testing; users perform integration testing
10. Which test investigates both functional and non-functional system requirements?
  - A. Alpha testing
  - B. System testing
  - C. Acceptance testing
  - D. Confirmation testing

11. Which of the following can be tested as part of operational testing?

- A. Component interaction
- B. Probe effect
- C. State transition
- D. Disaster recovery

12. What is the KEY difference between black-box and white-box testing?

- A. Black-box is functional; white-box is structural
- B. Black-box is functional; white-box is non-functional
- C. Black-box has a wider statement coverage than white-box
- D. Black-box can only be performed after white-box

13. Which test is usually run many times and generally evolve slowly?

- A. Performance testing
- B. Stress testing
- C. Reliability testing
- D. Regression testing

14. Which defects are OFTEN much cheaper to remove?

- A. Usability defects found by customers
- B. Defects in infrequently used functionality
- C. Defects that were defected early
- D. Minor defects that were found by users

15. Which statements correctly describe certain phases of a formal review?

- A. Looking for defects occurs during kick-off phase  
Fixing defects found happens during rework phase
- B. Personnel selection occurs during planning phase  
Gathering metrics happens during the review meeting phase
- C. Distributing documents occurs during the planning phase  
Personal review happens during individual preparation phase
- D. Personnel selection occurs during planning phase  
Fixing defects found happens during rework phase

16. Which defect can typically be discovered using a static analysis tool?

- A. Inconsistencies in numerical calculations
- B. Programming standards violations
- C. Problems related to system usability
- D. Internal and external system reliability

17. What consists of a set of input values, execution preconditions and expected results?

- A. Test script
- B. Test procedure specification
- C. Test case
- D. Test data

18. Which documents specify features to be tested, approach, and pass/fail criteria?

- A. Test plan and test design specification
- B. Test plan and test case specification
- C. Test procedure specification and test design specification
- D. Test case specification and test procedure specification

19. Which technique is appropriate to test changes on old and undocumented functionalities of a system?

- A. Specification-based technique
- B. Black-box technique
- C. White-box technique
- D. Data driven testing technique

20. Which test design technique relies heavily on prior thorough knowledge of the system?

- A. Data driven testing technique
- B. Experience-based technique
- C. White-box technique
- D. Structure-based technique

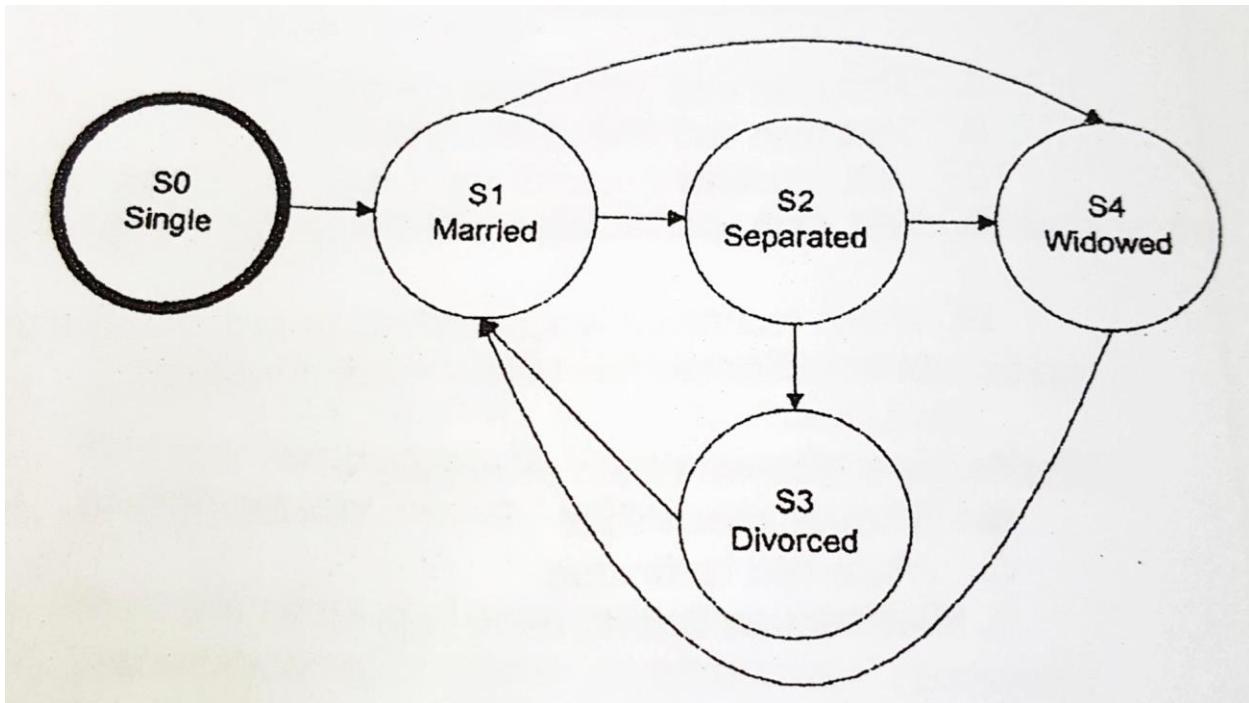
21. Which set of test data demonstrates equivalence partitioning to check whether a customer is a teenager or not?

- A. 10, 15, and 19 years
- B. 13, 19 and 25 years
- C. 13, 16 and 19 years
- D. 12, 13 and 20 years

22. What technique would be MOST appropriate to check status changes based on certain events?

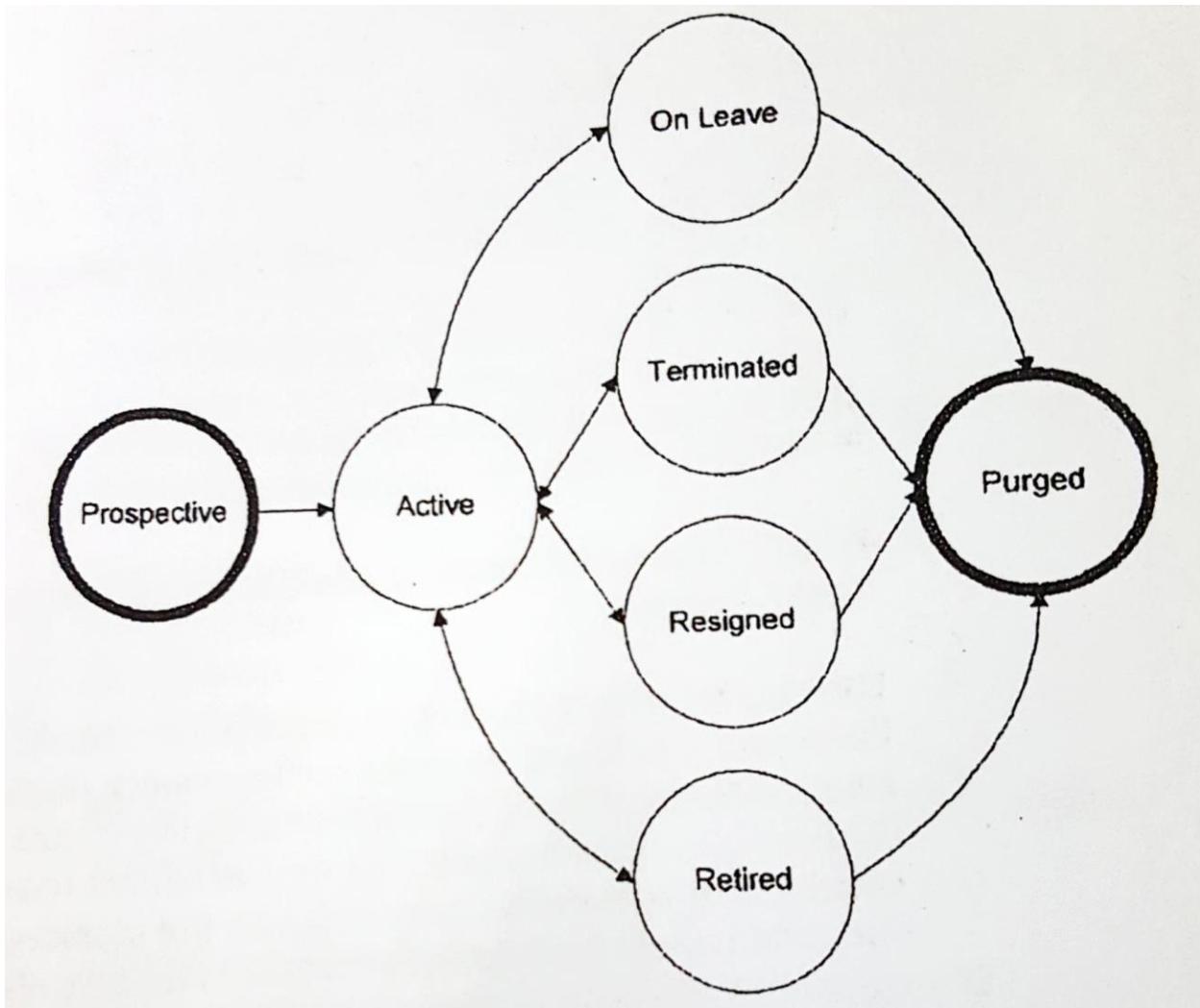
- A. State transition
- B. Equivalence partitioning
- C. Boundary value analysis
- D. Decision table

23. Using the diagram below, which test suite will check for ALL valid state transitions using the LEAST effort?



- A. S0 - S1 - S2 - S4 - S1 - S4 - S1 - S2 - S3 - S1
- B. S0 - S1 - S2 - S4 - S1 - S2 - S3 - S1
- C. S0 - S1 - S4 - S1 - S2 - S3 - S1
- D. S0 - S1 - S2 - S4 - S1 - S4 - S1 - S2 - S3

24. Using the diagram below, which test suite will uncover invalid state transitions for employee status reporting software?



- A. Prospective - Active - Resigned - Active - Terminated - Purged
- B. Prospective - Active - On Leave - Active - Resigned - Retired
- C. Prospective - Active - Retired - Active - On Leave - Purged
- D. Prospective - Active - On Leave - Active - Retired - Active

25. Which of the following assertions about code coverage are correct?

- A. Statement coverage usually requires more test case suites
- B. 100% statement coverage guarantees 100% decision coverage
- C. 100% decision coverage implies 100% statement coverage
- D. Decision tables cannot be used to list statement coverage values

26. How many test cases are needed to achieve 100% statement coverage?

```
if ( (temperature < 0) or  
     (temperature > 100) ) {  
    Alert ("DANGER");  
    if ( (speed > 100) and (load <= 50) ) {  
        speed = 50;  
    }  
} else {  
    Check = false;  
}
```

- A. 5
- B. 4
- C. 2
- D. 3

27. Using an error guessing test design technique to convert temperature (Celsius to Fahrenheit, and Fahrenheit to Celsius), experienced testers will MOST LIKELY use which set of test data?

- A. -1, 0, 89.6 and 212
- B. -40, 37.78, and 100
- C. -1, 0, 1 and 37.78
- D. -40, 0, 32 and 100

28. By creating future tests based on the results of previous tests, a tester is demonstrating what type of informal test design technique?

- A. Security testing
- B. Non-functional testing
- C. Exploratory testing
- D. Interoperability testing

29. Which of the following are potential drawbacks of independence in testing?

- 01. Independent testers may feel they are not part of the development team
  - 02. Developers may lose a sense of personal responsibility for quality
  - 03. Project managers will not have as much control on the project
  - 04. Customers may end up requesting features that are technically impossible
- A. 01 and 02
  - B. 01, 02 and 03
  - C. 03 and 04
  - D. 01, 02, 03 and 04

30. Which tasks are performed by a test leader versus a tester?

- S. Writing a project test strategy
- T. Selecting tools to support testing
- U. Preparing and acquiring data
- V. Scheduling tests

- A. Test leader: S and V; Tester: T and U
- B. Test leader: S, T and V; Tester: U
- C. Test leader: S, U and V; Tester: T
- D. Test leader: S; Tester: T, U and V

31. Which document specifies the sequence of test executions?

- A. Test procedure specification
- B. Test design specification
- C. Test case specification
- D. Test plan

32. Stochastic testing is an example of which test approach or strategy?

- A. Model-based
- B. Analytical
- C. Methodical
- D. Heuristic

33. Which sections are included as part of the test summary report?

- W. Variances
  - X. Comprehensive assessment
  - Y. Evaluation
  - Z. Summary of activities
- A. W, X and Y
  - B. W, X, Y and Z
  - C. W and X
  - D. W, X and Z
34. Which is a potential product risk factor?
- A. Failure of third party vendor
  - B. Training issues
  - C. Problems requirements definition
  - D. Poor software functionality

35. Based on the *IEEE Standard for Software Test Documentation* (IEEE Std 829- 1998), which sections of the test incident report should the following items be recorded?

Sections

- a) Test incident report identifier
- b) Summary
- c) Incident description
- d) Impact

Items

- 1. Impact on test plans
  - 2. Unique identifier
  - 3. Anomalies
  - 4. Procedure step
  - 5. Environment
  - 6. References to other relevant documents
- A. a: 2; b: 4; c: 1, 3 and 5; d: 6  
B. a: 2; b: 3; c: 4, 5 and 6; d: 1  
C. a: 2; b: 6; c: 3, 4 and 5; d: 1  
D. a: 2; b: 1; c: 3, 4 and 5; d: 6

36. Based on the IEEE Standard for Software Test Documentation (IEEE Std 829- 1998), which of the following sections are part of the test summary report?

- a) Test summary and report identifier
  - b) Comprehensive assessment
  - c) Summary of results
  - d) Evaluation
  - e) Observers
  - f) Approvals
- A. a, b, c, d and e  
B. a, b, c, e and f  
C. a, c, d, e and f  
D. a, b, c, d and f

37. What is the name of a temporary software component that is used to call another component for testing purposes?

- A. Domain
- B. Use case
- C. Stub
- D. Driver

38. Which of the following is a potential risk in using test support tools?

- A. Underestimating the effort needed to maintain the test assets
- B. Losing access to important testing information when needed
- C. Relying too much on qualitative and quantitative assessments
- D. Lowering the morale of the test team because of repetition

39. How are (a) static analysis tools and (b) performance testing tools different?

- A. (a) helps in enforcing coding standards; (b) tests system performance
- B. (a) analyzes security vulnerabilities; (b) measures the effectiveness of test cases
- C. (a) prepares codes prior to testing; (b) prepares codes prior to stress testing
- D. (a) highlights unreachable conditions; (b) improves system performance

40. Which of the following is a potential pilot project objective when introducing a test support tool into an organization?

- A. Measuring the satisfaction of management for staying within scope
- B. Assessing whether the benefits will be achieved at reasonable cost
- C. Receiving compliments from the users on the aesthetic aspects of the tool
- D. Reducing the amount of overtime need to finish the project on time

## ISTQB Exam 2 Answers

Question No.	Answer		Question No.	Answer
1	C		21	D
2	D		22	A
3	B		23	A
4	D		24	B
5	A		25	C
6	B		26	C
7	C		27	D
8	C		28	C
9	B		29	A
10	B		30	B
11	D		31	A
12	A		32	A
13	D		33	B
14	C		34	D
15	D		35	C
16	B		36	D
17	C		37	D
18	A		38	A
19	C		39	A
20	B		40	B

ها جبت كام؟ افتكـر ان النجاح من 26

ندخل بقى ع الامتحان الثالث...

## **ISTQB Exam 3**

1. Which approaches can help increase the quality of software?
  - I. Incorporating rigorous testing
  - II. Preventing change requests
  - III. Establishing defects metrics
  - IV. Allocating schedule contingencies
  - A. I and II are true; III and IV are false
  - B. III and IV are true; I and II are false
  - C. I and IV are true; II and III are false
  - D. I and III are true; II and IV are false
2. Which of the following can help testers understand the root causes of defects from previous projects?
  - A. Ishikawa diagram
  - B. Cause-and-defect diagram
  - C. Lessons learned
  - D. Fishbone diagram
3. What would USUALLY have a set of input values and execution conditions?
  - A. Test basis
  - B. Test case
  - C. Test objective
  - D. Test control

4. Which of the following general testing principles are true?

- I. Testing shows the presence of defects but not the absence of defects
  - II. Testing of combinations of inputs and outputs will find all defects
  - III. Testing should start after the completion of key development tasks
  - IV. Testing of safety-critical software is similar to testing web applications
- A. I is true; II, III and IV are false  
B. II is true; I, III and IV are false  
C. I and II are true; III and IV are false  
D. II and III are true; I and IV are false

5. Which of the following is a MAJOR test planning task?

- A. Determining the exit criteria
- B. Measuring and analyzing results
- C. Implementing corrective actions
- D. Monitoring and documenting progress

6. Which of the following is a KEY test closure task?

- A. Ensuring proper environment setup
- B. Writing a test summary report
- C. Assessing the need for additional tests
- D. Finalizing and archiving testware

7. Which statements BEST demonstrate the various degrees of independent testing?

- A. Tests are designed using exhaustive test design techniques
  - Tests are designed using exploratory test design techniques
- B. Tests are designed to generate most fundamental errors early
  - Tests are designed to uncover errors with the highest risk
- C. Tests are designed by the person who wrote the code
  - Tests are designed by a person from another department
- D. Tests are designed based on the documented requirements
  - Tests are designed based on the experience of the testers

8. Which of the following software work product can be used as a basis for testing?

- A. Incremental scenarios
- B. Design documents
- C. Undocumented features
- D. V-model specifications

9. Which tests are BEST described by the following characteristics?

W. Component testing

X. Integration testing

Y. Alpha testing

Z. Robustness testing

1. Testing the interaction between components
2. Fixing defects as soon as they are found
3. Automating test cases before coding
4. Testing separately testable components

A. W1, X4, Y3, and Z2

B. W2, W4, X1 and Z1

C. W2, W3, W4 and X1

D. W3, X1, X2 and X4

10. Which test is OFTEN the responsibility of the customers or users of the system?

A. Usability testing

B. Functional testing

C. Maintenance testing

D. Acceptance testing

11. Which acceptance test is USUALLY performed by system administrators?

A. Operational

B. Customer

C. Contractual

D. Regulatory

12. Which test can be performed at all test levels?

- A. System testing
- B. Operational testing
- C. Structural testing
- D. Integration testing

13. Regression testing can be applied to which of the following?

- I. Functional testing
  - II. Non-functional testing
  - III. Structural testing
- 
- A. I and II only
  - B. I and III only
  - C. II and III only
  - D. I, II and III

14. Which test requires the execution of the software component?

- A. Formal inspection
- B. Dynamic testing
- C. Code walkthrough
- D. Execution testing

15. As a moderator in a typical formal review, what can be one of your responsibilities?

- A. Deciding on the execution of reviews
- B. Documenting all the issues and problems
- C. Leading the review of the documents
- D. Identifying and describing the findings

16. What is the KEY difference in the usage of static analysis tools?
- A. Developers use static analysis tools before and during component testing  
Designers use static analysis tools during software modeling
  - B. Developers use static analysis tools to check the syntax of their codes  
Designers use static analysis tools to ensure adherence to programming standards
  - C. Developers use static analysis tools before and after integration testing  
Designers use static analysis tools to guarantee regulatory compliance
  - D. Developers use static analysis tools to check the syntax of their codes  
Designers use static analysis tools after software modelling
17. Features to be tested, approach refinements and feature pass/fail criteria BUT excluding environmental needs should be specified in which document?
- A. Test case specification
  - B. Test plan
  - C. Test procedure specification
  - D. Test design specification
18. Which document specifies the execution order of test cases?
- A. Test design specification
  - B. Test item
  - C. Test procedure specification
  - D. Test plan

19. Match the test design techniques to the correct descriptions

- S. Black-box technique
- T. White-box technique
- U. Structural-based technique
- V. Specification-based technique

1. Selecting test cases based on documentation

2. Ignoring the internal structure of the system

- A. S1, S2, U1 and U2
- B. T1, T2, U1 and U2
- C. S1, S2, V1 and V2
- D. T1, T2, V1 and V2

20. What techniques would be MOST appropriate if the specifications are outdated?

- A. Structure-based and experienced-based techniques
- B. Black-box and specification-based techniques
- C. Specification-based and structure-based techniques
- D. Structure-based techniques and exhaustive testing

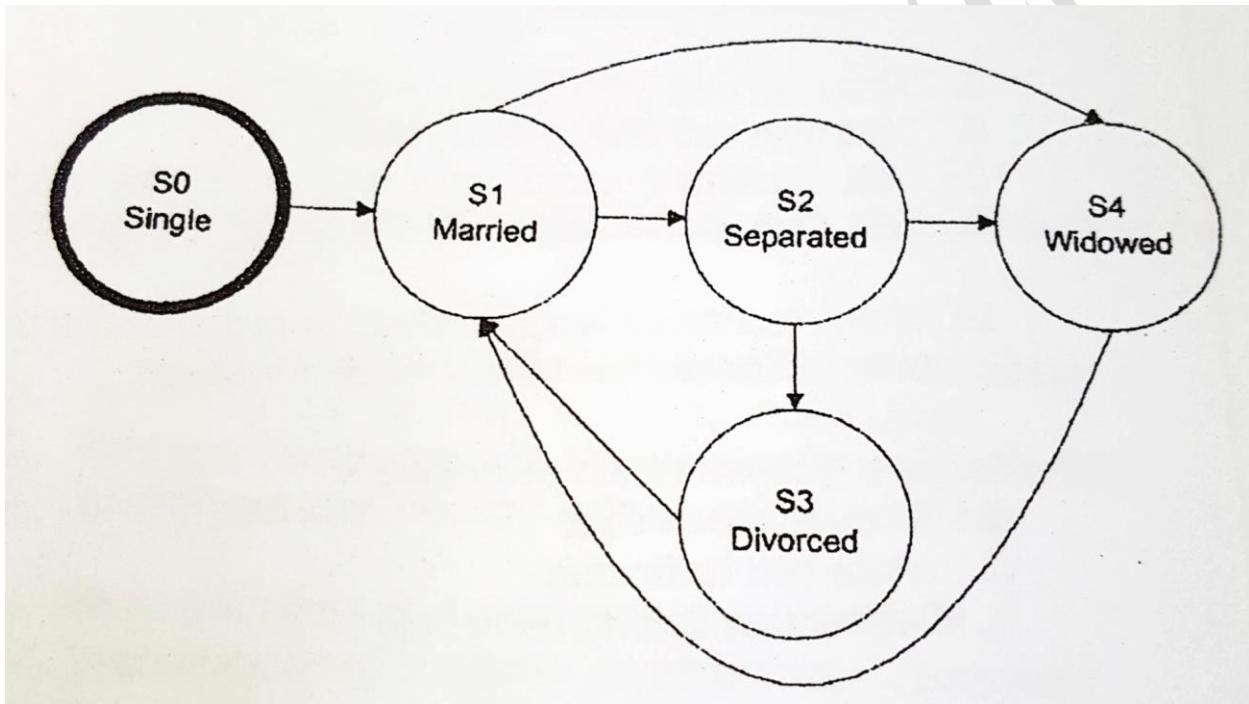
21. To test an input field that accepts a two-digit day based on a particular month which data set demonstrates boundary value analysis?

- A. 0, 1, 16, 31 and 100
- B. 1, 27, 28, 30 and 31
- C. 2, 26, 27, 29 and 30
- D. -1, 0, 15, 32 and 99

22. Which technique is OFTEN considered as an extension of equivalence partitioning?

- A. Decision table testing
- B. State transition testing
- C. Use case testing
- D. Boundary value analysis

23. Without testing all possible transitions, which test suite will test all marital statuses?



- A. S0 - S1 - S2 - S4 - S1 - S4
- B. S0 - S1 - S2 - S3 - S1 - S2
- C. S0 - S1 - S4 - S1 - S2 - S3
- D. S0 - S1 - S2 - S3 - S4 - S1

24. Which technique describes process flows through a system based on its likely usage?

- A. Data driven testing
- B. State transition testing
- C. Decision table testing
- D. Use case testing

25. Which combination of p, q and r values will ensure 100% statement coverage?

```
if (p = q) {  
    r = r + 1;  
    if (r < 5) {  
        s = 10;  
    }  
} else if (p > q) {  
    S = 5;  
}
```

- A. p = 5, q = 5, r = 5  
 p = 5, q = 4, r = -1
- B. p = 5, q = 1, r = 3  
 p = 4, q = 4, r = 5
- C. p = 3, q = 3, r = 3  
 p = -1, q = -2, r = 3
- D. p = -1, q = -1, r = 0  
 p = -2, q = -1, r = 0

26. How many test cases are needed to achieve 100% condition coverage?

```
if ( (temperature < 0) or  
     (temperature > 100) ) {  
    Alert ("DANGER");  
    if ( (speed > 100) and (load <= 50) ) {  
        speed = 50;  
    }  
} else {  
    Check = false;  
}
```

- A. 5
- B. 4
- C. 2
- D. 3

27. What is an informal test design technique where a tester uses information gained while testing to design new and better tests?

- A. Error guessing
- B. Exploratory testing
- C. Use case testing
- D. Decision table testing

28. How are error guessing and exploratory testing similar?

- A. Both are widely used formal techniques
- B. Both are white-box test design techniques
- C. Both are experience-based testing
- D. Both are effective at all testing levels

29. Who should be responsible for coordinating the test strategy with the project manager and others?

- A. Tester
- B. Developer
- C. Customer
- D. Test leader

30. Which of the following are KEY tasks of a test leader?

- i. Understanding the project risks
- ii. Measuring performance of components
- iii. Scheduling tests and other activities
- iv. Using monitoring tools as needed

- A. i and iii
- B. i and ii
- C. iii and iv
- D. ii and iii

31. What is a group of test activities that are organized and managed together?

- A. Test procedure specification
- B. Test level
- C. Test case specification
- D. Test plan

32. Which test approaches or strategies are characterized by the descriptions below?

- S. Process-compliant approaches
- T. Heuristic approaches
- U. Consultative approaches
- V. Regression-averse approaches
- 1. Includes reuse of existing test material
- 2. Listens to suggestions from technology experts
- 3. Adheres to industry-specific standards
- 4. Runs test execution and evaluation concurrently

- A. S4, T3, U2, V1
- B. S1, T2, U3, V4
- C. S2, T3, U1, V4
- D. S3, T4, U2, V1

33. What is the ratio of the number of failures relative to a category and a unit of measure?

- A. Failure rate
- B. Defect density
- C. Failure mode
- D. Fault tolerance

34. What are the main objectives of software project risk management?

- A. Increase focus on preventive processes and improve tester job satisfaction
- B. Reduce the probability of occurrence and decrease the potential impact
- C. Control contractor problems and minimize the impact of corporate politics
- D. Increase the probability of project success regardless of the cost involved

35. Based on the *IEEE Standard for Software Test Documentation* (IEEE Std 829-1998), which sections of the test incident report should the following details be recorded?

- a) Test incident report identifier
- b) Summary
- c) Incident description
- d) Impact

- 1. Expected results
- 2. Actual results
- 3. Procedure step
- 4. Environment
- 5. Revision level
- 6. Date and time

- A. a: 3; b: 5; c: 1, 2, 4 and 6
- B. b: 5; c: 1, 2, 3, 4 and 6
- C. b: 5 and 6; c: 1, 2, 3 and 4
- D. a: 5; c: 1, 2, 3, 4 and 6

36. Based on the IEEE Standard for Software Test Documentation (IEEE Std 829-1998), which of the following sections are part of the test summary report?

- a) Summary
  - b) Test incident report identifier
  - c) Test deliverables
  - d) Risks and contingencies
  - e) Variances
  - f) Approvals
  - g) Output specifications
- A. a, e and f  
B. a, c and d  
C. a, b and f  
D. a, d and e

37. Which tool can be used to support and control part of the test management process?

- A. Coverage management tool
- B. Test management tool
- C. Data preparation tool
- D. Performance testing tool

38. What is a scripting technique that uses data files to contain not only test data and expected results, but also keywords related to the application being tested?

- A. Automation technique
- B. Scripting language
- C. Process-driven testing
- D. Keyword-driven testing

39. Which tool needs to interface with other office automation software in order to generate reports in the format required by the organization?

- A. Progress tracking tools
- B. Test management tools
- C. Metrics management tools
- D. Test execution tools

40. Which of the following can be considered as success factors when deploying a new tool in an organization?

- A. Providing coaching to users and defining usage guidelines
- B. Monitoring tool usage and reducing the need for risk analysis
- C. Improving processes and focusing more on component testing
- D. Assessing testing completion and minimizing code reviews

## ISTQB Exam 3 Answers

Question No.	Answer		Question No.	Answer
1	D		21	B
2	C		22	D
3	B		23	C
4	A		24	D
5	A		25	C
6	D		26	A
7	C		27	B
8	B		28	C
9	C		29	D
10	D		30	A
11	A		31	B
12	C		32	D
13	D		33	A
14	B		34	B
15	C		35	B
16	A		36	A
17	D		37	B
18	C		38	D
19	C		39	B
20	A		40	A

ها جبت كام؟ افتكـر ان النـجـاح من 26 ... المـفـروض بـرضـه تكون حلـيـت الـامـتحـان دـه فـى خـلـال الـوقـت  
المـطـلـوب (60 دقـيقـة)

نـدخل بـقـى عـامـتحـان الـرـابـع وـالـأـخـير ...

## **ISTQB Exam 4**

1. Which of the problems below BEST characterize a result of software failure?
  - A. Damaged reputation
  - B. Lack of methodology
  - C. Inadequate training
  - D. Regulatory compliance
  
2. What should be taken into account to determine when to stop testing?
  - I. Technical risk
  - II. Business risk
  - III. Project constraints
  - IV. Product documentation  
  - A. I and II are true; III and IV are false
  - B. III is true; I, II and IV are false
  - C. I, II, and IV are true; III is false
  - D. I, II and III are true; IV is false
  
3. What is the process of analyzing and removing causes of failures in software?
  - A. Validation
  - B. Testing
  - C. Debugging
  - D. Verification

4. Which general testing principles are characterized by the descriptions below?

- W. Early testing
- X. Defect clustering
- Y. Pesticide paradox
- Z. Absence-of-errors fallacy

- 1. Testing should start at the beginning of the project
- 2. Conformance to requirements and fitness for use
- 3. Small number of modules contain the most defects
- 4. Test cases must be regularly reviewed and revised

- A. W1, X2, Y3 and Z4
- B. W1, X3, Y4 and Z2
- C. W2, X3, Y1 and Z4
- D. W1, X4, Y2 and Z3

5. Which of the following MAIN activity is part of the fundamental test process?

- A. Initiating and planning
- B. Documenting root-causes
- C. Capturing lessons learned
- D. Planning and control

6. Which of the following are MAJOR test implementation and execution tasks?

- I. Repeating test activities
  - II. Creating test suites
  - III. Reporting discrepancies
  - IV. Logging the outcome
  - V. Analyzing lessons learned
- 
- A. II, III and IV
  - B. I, III, IV and V
  - C. I, II, III and IV
  - D. III, IV and V

7. What principle is BEST described when test designs are written by a third-party?

- A. Exploratory testing
- B. Independent testing
- C. Integration testing
- D. Interoperability testing

8. Which test levels are USUALLY included in the common type of V-model?

- A. Integration testing, system testing, acceptance testing and regression testing
- B. Component testing, integration testing, system testing and acceptance testing
- C. Incremental testing, exhaustive testing, exploratory testing and data driven testing
- D. Alpha testing, beta testing, black-box testing and white-box testing

9. What test can be conducted for off-the-shelf software to get market feedback?

- A. Beta testing
- B. Usability testing
- C. Alpha testing
- D. COTS testing

10. Who OFTEN performs system testing and acceptance testing respectively?

- A. Senior programmers and professional testers
- B. Technical system testers and potential customers
- C. Independent test team and users of the system
- D. Development team and customers of the system

11. What is the key difference between (a) contract and regulation acceptance testing, and (b) alpha and beta testing?

- A. (a) are performed outside the company and (b) are conducted by the test team
- B. (a) are conducted by regulators and (b) are performed by system administrators
- C. (a) are mandatory test for government applications and (b) are usually optional
- D. (a) are for custom-developed software and (b) are for off-the-shelf software

12. Which test measures the system at or beyond the limits of its specified requirements?

- A. Structural testing
- B. Stress testing
- C. Error guessing
- D. Black-box testing

13. Which test ensures that modifications did not introduce new problems?

- A. Stress testing
- B. Black-box testing
- C. Structural testing
- D. Regression testing

14. Which typical defects are easier to find using static instead of dynamic testing?

- L. Deviation from standards
  - M. Requirements defects
  - N. Insufficient maintainability
  - O. Incorrect interface specifications
- 
- A. L, M, N and O
  - B. L and N
  - C. L, N and O
  - D. L, M and N

15. In a formal review, who is primarily responsible for the documents to be reviewed?

- A. Author
- B. Manager
- C. Moderator
- D. Reviewers

16. Who typically use static analysis tools?

- A. Customers and users
- B. Developers and designers
- C. Business and systems analysts
- D. System and acceptance testers

17. Which aspects of testing will establishing traceability help?

- A. Configuration management and test data generation
- B. Test case specification and change control
- C. Test condition and test procedure specification
- D. Impact analysis and requirements coverage

18. Features to be tested, approach, item pass/fail criteria and test deliverables should be specified in which document?

- A. Test case specification
- B. Test procedure specification
- C. Test plan
- D. Test design specification

19. Which test technique is based on requirements specifications?

- A. White-box technique
- B. Component testing
- C. Black-box technique
- D. Data driven testing

20. Which test design techniques should a tester use to respectively achieve the following: (a) check the documented features of the system, (b) ensure 100% decision coverage, and (c) defect likely defects and distribution?

- A. Specification-based, data driven testing, and defect density techniques
- B. Specification-based, branch coverage, and exploratory techniques
- C. Structure-based, equivalence partitioning, and exploratory techniques
- D. Specification-based, structure-based, and experience-based techniques

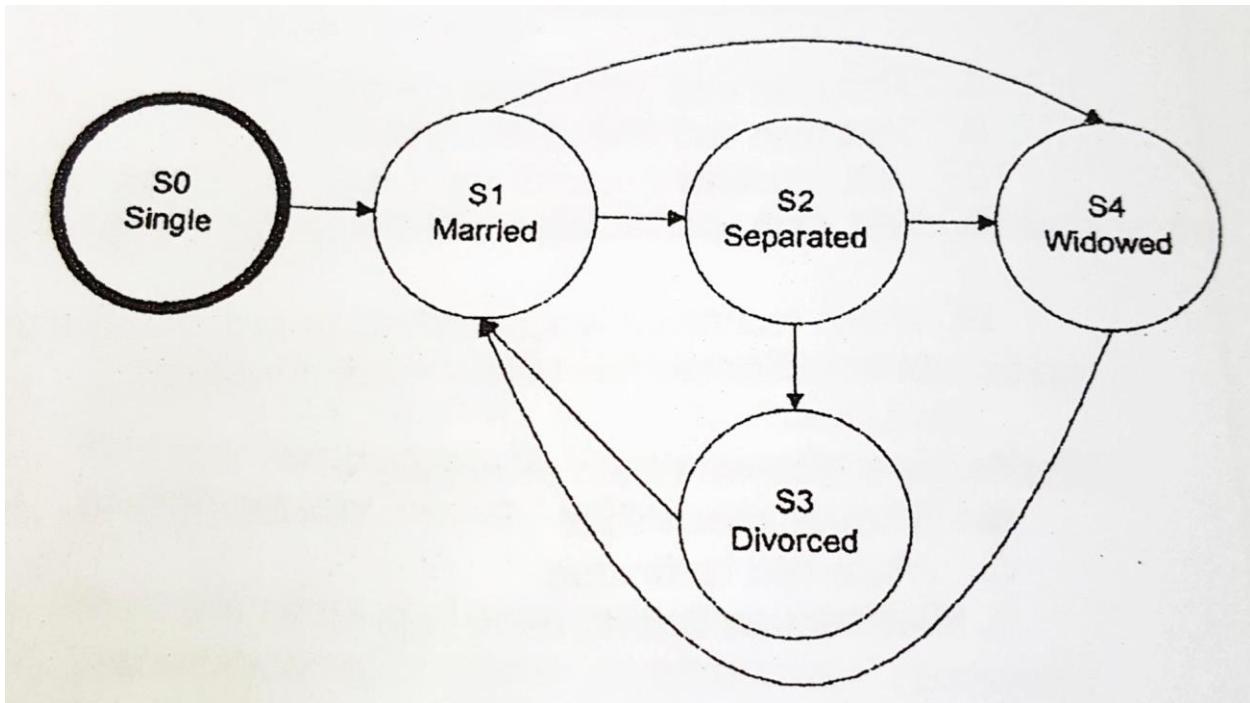
21. What technique captures system requirements that contain logical conditions?

- A. Boundary value
- B. Equivalence partitioning
- C. Decision table
- D. State transition

22. Input and output combinations that will be treated the same way by the system can be tested using which technique?

- A. Boundary value
- B. Equivalence partition
- E. Decision table
- F. State transition

23. Which test suite will check for an invalid transition using the diagram below?



- A. S0 - S1 - S2 - S3 - S1 - S4
- B. S0 - S1 - S4 - S1 - S2 - S3
- C. S0 - S1 - S3 - S1 - S2 - S1
- D. S0 - S1 - S2 - S3 - S1 - S2

24. How are integration testing and use case testing similar and dissimilar?

- A. Both checks for interactions: integration for components, use case for actors
- B. Both are black-box techniques: integration is low-level, use case is high-level
- C. Both are static testing: developers perform integration, users execute use case tests
- D. Both are V&V techniques: integration is for validation, use case is for verification

25. How many test cases are needed to achieve 100% decision coverage?

```
if (p = q) {  
    s = s + 1;  
    if (s < 5) {  
        t = 10;  
    }  
} else if (p > q) {  
    t = 5;  
}
```

- A. 3
- B. 6
- C. 5
- D. 4

26. What analysis determines which parts of the software have been executed?

- A. Impact analysis
- B. Code coverage
- C. Gap analysis
- D. Cyclomatic complexity

27. Based on the error guessing test design technique, which of the following will an experienced tester MOST LIKELY test in calendar software?

- i. First two letters of the month, e.g., MA can represent March or May
  - ii. First letter of the day, e.g., T can mean Tuesday or Thursday
  - iii. Leap year
  - iv. Number of days in a month
  - v. Three-digit days and months
- A. i, ii, iv and v  
B. iii and iv  
C. i, ii, iii and iv  
D. i, ii and v
28. Which input combinations will a knowledgeable tester MOST LIKELY use to uncover potential errors when testing a surname field?
- A. Johnson, de la Cruz and Morgan
  - B. Go, Stephanopoulos and Venkatsewaran
  - C. Smit, Smyth and Smithsonian
  - D. O'Brien, Zeta-Jones and Young Pów

29. Which of the following demonstrates independence in testing?
- J. Independent testers are external to the organization
  - K. Independent testers are part of the development team
  - L. Independent testers are from the user community
  - M. Programmers who wrote the code serve as independent testers
  - N. Customers who wrote the requirements serve as independent testers
- A. J, L and N
- B. J, K, L and N
- C. K, M and N
- D. J, L, M and N
30. Which of the following is a KEY task of a tester?
- A. Reviewing tests developed by others
  - B. Writing a test strategy for the project
  - C. Deciding what should be automated
  - D. Writing test summary reports
31. In software testing, what is the MAIN purpose of exit criteria?
- A. To enhance the security of the system
  - B. To prevent endless loops in codes
  - C. To serve as an alternative or "Plan B"
  - D. To define when to stop testing

32. Which test approaches or strategies are characterized by the descriptions below?

S. Analytical approaches

T. Model-based approaches

U. Methodical approaches

V. Consultative approaches

1. Relies on guidelines from domain experts

2. Includes error guessing and fault-attacks

3. Use statistical information about failure rates

4. Focuses on areas of greatest risk

A. S4, T3, U2, V1

B. S1, T2, U3, V4

C. S2, T3, U1, V4

D. S3, T4, U2, V1

33. Which of the following can be used to measure progress against the exit criteria?

W. Number of test cases that passed or failed

X. Number of defects found in a unit of code

Y. Dates for milestones and deliverables

Z. Subjective confidence of testers in the product

A. W, X, Y and Z

B. W, X and Y

C. W and X

D. W, X and Z

34. What type of risk includes potential failure areas in the software?

- A. Project risks
- B. Product risks
- C. Economic risks
- D. Requirements risks

35. Based on the IEEE Standard for Software Test Documentation (IEEE Std 829-1998), which sections of the test incident report should the following details be recorded?

Sections

- a) Test incident report identifier
- b) Summary
- c) Incident description
- d) Impact

Details

1. Unique identifier
2. Version level of the test items
3. Inputs
4. Expected results
5. Actual results
6. Anomalies
7. Date and time

- A. a: 1; b: 2 and 7; c: 3, 4 and 5; d: 6
- B. a: 1; b: 6 and 7; c: 3, 4 and 5; d: 7
- C. a: 1; b: 2; c: 3, 4, 5, 6 and 7
- D. a: 1; b: 6 and 7; c: 3, 4 and 5

36. Based on the IEEE Standard for Software Test Documentation (IEEE Std 829-1998), which of the following sections are part of the test summary report?

- a) Test summary and report identifier
  - b) Summary
  - c) Variances
  - d) Anomalies
  - e) Comprehensive assessment
  - f) Approvals
- A. a, b, e and f  
B. a, b, c, d and f  
C. a, b, c, e and f  
D. a, b, c and f

37. What is the name of a skeletal implementation of a software component that is used for testing?

- A. Use case
- B. Domain
- C. Driver
- D. Stub

38. Which of the following are potential benefits of using test support tools?

- A. Ensuring greater consistency and minimizing software project risks
- B. Reducing repetitive work and gaining easy access to test information
- C. Performing objective assessment and reducing the need for training
- D. Allowing for greater reliance on the tool to automate the test process

39. Which test support tool can be used to enforce coding standards?

- A. Static analysis tools
- B. Performance testing tool
- C. Test comparator
- D. Test management tool

40. What should be considered when introducing a tool into an organization?

- A. Assessing the organizational maturity
- B. Counting the number of systems to be tested
- C. Calculating the ratio between programmers and testers
- D. Reviewing the exit criteria of previous projects

## ISTQB Exam 4 Answers

Question No.	Answer		Question No.	Answer
1	A		21	C
2	D		22	B
3	C		23	C
4	B		24	A
5	D		25	D
6	C		26	B
7	B		27	C
8	B		28	D
9	A		29	B
10	C		30	A
11	D		31	D
12	B		32	A
13	D		33	A
14	A		34	B
15	A		35	C
16	B		36	C
17	D		37	D
18	C		38	B
19	C		39	A
20	D		40	A

ها جبت كام؟ افتكـر ان النـجـاح من 26 ... المـفـروض بـرضـه تكون حلـيـت الـامـتحـان دـه فـى خـلـال الـوقـت المـطـلـوب (60 دقـيقـة).

## خاتمة

دلوقتى احنا وصلنا لنهاية الكتاب... بکده أقدر أقولك إنك دلوقتى تقدر تدخل امتحان ال ISTQB وتمتحن الشهادة وتنجح إن شاء الله، وكمان تقدر تشتعل زى أى professional tester وتدخل أى interview وانت واثق من نفسك كمان زى ما وعدتك فى أول الكتاب، فالاهم من ال certificate إنك تكون فاهم كل تفاصيل الشغل بتاعك سواء business أو technical.

طب هل ده آخر المطاف؟

لأ... شهادة ال ISTQB Foundation Level اللي احنا اتكلمنا عن منهجهما عن الكتاب دى بداية لحالات تانية كتير منزلاها ال ISTQB Agile زى مثلا ال ISTQB Agile اللي بتساعدك إنك تكون تيستر شغال جوة Agile environment وده حاليا مطلوب فى شركات كتير لإنأغلبهم حاليا بيتجه لطريقة ال Agile ف شغفهم، ممكن كمان تاخذ شهادة ال ISTQB Automation، وده بيعملك إزاي تعمل automation testing و تكون automation tester الوظيفة دى مافيهاش ناس كتير ومرتباتها أعلى من ال manual tester ... فحاول تفكر فيها الفترة الجاية... لكن كل دول عشان توصلهم لازم تعدى الأول ال foundation level اللي احنا نقشناه فى الكتاب، فدوس ف الطريق ده يامعلم وربنا يعينك ويوافقك، وفيه صفحات وجروبات كتير ع ال LinkedIn وال facebook مهمته بالتيستينج ووظائفها... دور انت واكتشف بنفسك... كل ماتدور أكثر كل ماذراتك بتزيد ومعرفتك بتزيد.

حابب بس أشكر كل أصحابي وزميلى اللي مابخلوش بالدعم والتشجيع من ساعة ما عرفوا إنى بعمل الكتاب ده، كمان بشكر إدارة (ACT) Advanced Computer Technology على كامل الدعم اللي قدموه ليا عشان يخرج الكتاب ده فى أسرع وقت ممكن...

ف الآخر خالص بعتذر إن كان فيه أى خطأ سواء لغوی أو technical، الحقيقة أنا اشتغلت ع الكتاب ده حوالي 4 شهور ليل نهار... بعتذر لو فيه حاجة سقطت منى بالغلط ولا حاجة، وبرجوك لو لاقيت أى bug (تيستر بقى) ف الكتاب تبعتنى عشان أقدر أصححه، ودى الحسابات بتاعتى لو فيه أى حاجة شفتها غلط أو فيه استفسار معين حابب تعرفه... وأكيد طبعاً أشرف بمعرفتك من غير أى حاجة خالص.

Email: [ahmed.etman24@gmail.com](mailto:ahmed.etman24@gmail.com)

LinkedIn: <https://www.linkedin.com/in/ahmedetman24/>

Facebook: <https://www.facebook.com/ahmed.etman.24>

شكرا جزيلا

تم بحمد الله

## References

- [https://en.wikipedia.org/wiki/Samsung\\_Galaxy\\_Note\\_7](https://en.wikipedia.org/wiki/Samsung_Galaxy_Note_7)
- [https://en.wikipedia.org/wiki/Software\\_development\\_process](https://en.wikipedia.org/wiki/Software_development_process)
- [https://en.wikipedia.org/wiki/Waterfall\\_model](https://en.wikipedia.org/wiki/Waterfall_model)
- [https://en.wikipedia.org/wiki/V-Model\\_\(software\\_development\)](https://en.wikipedia.org/wiki/V-Model_(software_development))
- [https://en.wikipedia.org/wiki/Website\\_wireframe](https://en.wikipedia.org/wiki/Website_wireframe)
- [https://www.tutorialspoint.com/sdlc/sdlc\\_waterfall\\_model.htm](https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm)
- [https://en.wikipedia.org/wiki/Systems\\_development\\_life\\_cycle](https://en.wikipedia.org/wiki/Systems_development_life_cycle)
- <https://softwareengineering.stackexchange.com/questions/228282/can-someone-explain-the-v-model-process-why-is-it-different-than-the-waterfall>
- <https://airbrake.io/blog/sdlc/iterative-model>
- [https://www.tutorialspoint.com/sdlc/sdlc\\_iterative\\_model.htm](https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm)
- [https://en.wikipedia.org/wiki/Agile\\_software\\_development](https://en.wikipedia.org/wiki/Agile_software_development)
- [https://www.tutorialspoint.com/software\\_testing/software\\_testing\\_qa\\_qc\\_testing.htm](https://www.tutorialspoint.com/software_testing/software_testing_qa_qc_testing.htm)
- <https://www.differenc.com/difference/Quality Assurance vs Quality Control>
- <https://en.wikipedia.org/wiki/QA/QC>
- [https://en.wikipedia.org/wiki/Software\\_bug#History](https://en.wikipedia.org/wiki/Software_bug#History)
- <https://www.akamai.com/>
- <http://www.photobucket.com/>
- [https://en.wikipedia.org/wiki/System\\_testing](https://en.wikipedia.org/wiki/System_testing)
- <http://softwaretestingfundamentals.com/system-testing/>
- <https://en.wikipedia.org/wiki/Flowchart>
- <https://www.facebook.com/StepByStepTesting/photos/a.302241196879539.1073741828.298200600616932/365408233896168/?type=3&theater>
- <http://istqb-glossary-explanations.blogspot.com.eg/2013/08/testware.html>
- <https://www.guru99.com/what-is-test-harness-comparison.html>
- [https://en.wikipedia.org/wiki/Test\\_harness](https://en.wikipedia.org/wiki/Test_harness)
- <http://istqb-glossary-explanations.blogspot.com.eg/search?q=test+harness>