UNIT-II

प्रश्न 1 – Black-box टेस्टिंग से किस तरह कि गलतियां ढूंढी जाती हैं?

What kinds of errors are found by Black-box testing?

उत्तर–Black-box Testing–Black-box Testing को behavioural Testing भी कहते है। यह सॉफ्टवेयर की फंक्शनल आवश्यकताओं पर फोकस करती है; अर्थात् Black-box टेस्टिंग सॉफ्टवेयर इंजीनियर के लिये इनपुट शर्तों के लिये derive सेट करने में समय बताती है जो एक प्रोग्राम के लिये सभी फंक्शनल आवश्यकताओं का पूर्ण रूप से अभ्यास करेगा। Black-box टेस्टिंग, White-box तकनीकी का alternative नहीं है। यह एक Complementary एप्रोच है जो White-box मैथड्स की अपेक्षा errors की विभिन्न classes को uncover करता है। Black-box टेस्टिंग errors की निम्नलिखित श्रेणियों को find करने के लिये चुनती है–

1. Incorrect or missing information
2. Interface errors
3. Errors in data structures or external database
4. Behaviour or Performance errors
5. Initialization and termination errors.

Black-box टेस्टिंग, टेस्टिंग के later stage के समय लागू होती है, क्योंकि Black-box टेस्टिंग इनफॉरमेशन डोमेन के कण्ट्रोल स्ट्रक्चर पर ध्यान केन्द्रित करती है।

Tests निम्नलिखित प्रश्नों के उत्तर के लिये डिजायन किया जाता है–

1. Functional वैधता कैसे टेस्ट होती है?
2. सिस्टम Behaviour तथा परफॉरमेन्स कैसे टेस्ट होते हैं?
3. कौन-सी Classes के इनपुट अच्छे Test case को बनायेंगे?
4. क्या सिस्टम निश्चित इनपुट वैल्यूज के लिये संवेदनशील है?
5. डाटा क्लास की boundaries कैसी है?
6. कौन-से data rates तथा data volume सिस्टम को Tolerant कर सकते हैं?
7. सिस्टम ऑपरेशन पर डाटा के विशिष्ट Combinations का क्या प्रभाव पड़ेगा?

Black-box तकनीकी को लागू करने से हम Test cases के एक Set को Derive करते हैं, जो निम्नलिखित Criteria की संतुष्ट करता है–

1. Test case, जो परिवर्तित होते हैं।
2. Test case जो हमें Errors की Classes की उपस्थिति या अनुपस्थिति के बारे में बताता है। ❑

प्रश्न 2 – Black-box testing तथा Structural testing के बीच अन्तर बताइये।

Differentiate between Black-box testing and Structural testing.

उत्तर–Black-box Testing तथा Structural Testing के बीच मुख्य अन्तर निम्नलिखित हैं–

Black-box Testing	Structural Testing
1. Black-box टेस्टिंग सिस्टम, फंक्शनल, या behavioural टेस्टिंग के नाम से भी जानी जाती है।	1. Structural Testing को कंट्रोल स्ट्रक्चर टेस्टिंग के नाम से भी जाना जाता है।
2. Black-box टेस्टिंग, टेस्ट किये जाने वाले इन्टरनल वर्किंग के knowledge के बिना carried out होता है।	2. Structural Testing के अंतर्गत कंडीशन टेस्टिंग, डाटा फ्लो टेस्टिंग, लूप टेस्टिंग इत्यादि आते हैं।
3. Black-box टेस्टिंग में टेस्टर को केवल legal Inputs तथा आशान्वित आउटपुट के बारे में जानना होगा।	3. कंडीशन टेस्टिंग, **प्रोग्राम मॉड्यूल में समाहित लॉजिकल कंडीशनस् के अभ्यास के अनुरूप टेस्ट case डिजाइन मैथड् है।**

Scanned by CamScanner

4. Black-box टेस्टिंग में टेस्टर को प्रोग्राम कोड में एक्सेस उपलब्ध नहीं करायी जानी चाहिये।	4. कंडीशन टेस्टिंग का रिलेशनल एक्सप्रेशन निम्नलिखित रूप में है। E_1 < relational-operator > E_2 यहां पर E_1 तथा E_2 अर्थमेटिक एक्सप्रेशन है।
5. Black-box टेस्टिंग में, सिस्टम बड़े Black-box की तरह treat किया जाता है तथा टेस्टर इसके Inside नहीं देख सकता है।	5. लूप टेस्टिंग, सॉफ्टवेयर में इम्प्लीमेन्ट किये गये सभी एल्गोरिद्म्स होते हैं।
6. Black-box टेस्टिंग, Specification के respect में टेस्टिंग के रूप में विचार की जाती है।	6. लूप टेस्टिंग White-box टेस्टिंग तकनीकी है जो लूप Constructs की वैधता का फोकस होते हैं।
7. Black-box टेस्टिंग requirement specification तथा उच्च लेवल डिजाइन के आधार पर प्लान की जा सकती है।	7. डाटा फ्लो टेस्टिंग मैथड्स प्रोग्राम के test paths को सलेक्ट करते हैं जो प्रोग्राम में वेरियेबल्स के प्रयोगों तथा परिभाषाओं की लोकेशन के अनुसार होते हैं।
8. Black-box टेस्टिंग में smoke Testing, correctness Testing, stress Testing, load Testing, performance Testing इत्यादि सम्मिलित हैं।	8. डाटा फ्लो टेस्टिंग एप्रोच में एक प्रोग्राम में प्रत्येक स्टेटमेण्ट को Unique स्टेटमेण्ट नम्बर एसाइन किया जाता है।

❑

प्रश्न 3 – सिस्टम टेस्टिंग क्यों आवश्यक है? *User acceptance testing* क्या है?

Why is system testing necessary ? What is user acceptance testing?

उत्तर–सिस्टम टेस्टिंग (System Testing)–जैसा कि हम जानते हैं कि कम्प्यूटर पर आधारित सिस्टम का सबसे बड़ा एलीमेन्ट सॉफ्टवेयर है। वास्तविक रूप में सिस्टम टेस्टिंग विभिन्न टेस्टों की एक सीरीज है, जिसका प्रारम्भिक उद्देश्य कम्प्यूटर पर आधारित सिस्टम का पूर्ण रूप से अभ्यास करना है। सॉफ्टवेयर पर आधारित सिस्टमुस की प्रमुख सिस्टम टेस्टिंग निम्नलिखित हैं–

1. रिकवरी टेस्टिंग (Recovery Testing)–बहुत-से कम्प्यूटर पर आधारित सिस्टमों को एक निर्धारित समय में Fault तथा Resume प्रोसेसिंग से recover अवश्य करना चाहिये। कुछ Cases में सिस्टम Fault tolerant होना चाहिये; अर्थात् प्रोसेसिंग दोष को पूरे सिस्टम फंक्शन को Cause करने का कारण नहीं होना चाहिये। दूसरे Cases में सिस्टम असफलता एक निश्चित समय सीमा में सही होनी चाहिये। रिकवरी टेस्टिंग एक सिस्टम टेस्ट है जो सॉफ्टवेयर को विभिन्न तरीकों में असफल होने के लिये दबाव बनाती है तथा पुष्टि करती है कि रिकवरी सही तरीके से पूरी की गयी है।

2. सिक्योरिटी टेस्टिंग (Security Testing)–कोई भी कम्प्यूटर पर आधारित सिस्टम जो नाजुक सूचना को मैनेज करता है या Cause actions जो असामान्य तरीके से हानि पहुंचा सकता है, वह अवैध छेदन या गलत तरीके से Target होता है। छेदन (Penetration), गलती गतिविधियों की रेन्ज का एक बोर्ड है, जैसे हैकर्स जो सिस्टमों को Sporss के लिये Penetrate करने के लिये चुनते हैं, गैर जिम्मेदार Employees जो बदले के लिये Penetrate को चुनते हैं। सिक्योरिटी टेस्टिंग पुष्टि करती है कि प्रोटेक्शन विधि, एक सिस्टम will को बनाती है, वास्तव में यह असामान्य Panetration से प्रोटेक्ट करती है।

सिक्योरिटी टेस्टिंग के समय, टेस्टर का role व्यक्तिगत रूप से होता है जिसकी इच्छा सिस्टम को Penetrate करती है। कुछ भी हो, टेस्टर पासवर्ड को प्राप्त कर लेता है तथा सिस्टम पर attack कर सकता है।

3. स्ट्रेस टेस्टिंग (Stress Testing)–Stress Testing को असामान्य स्थितियों में प्रोग्राम को Comfront करने के लिये डिजाइन किया गया है।

Stress Testing एक सिस्टम को एक तरीके से क्रियान्वित करती है, जो संसाधनों की मांग असामान्य मात्रा, Frequency या Volume में करती है।

Stress Testing का Variation, Sensitive Testing कहलाता है। Sensitive Testing, बिना Cover किये गये data combinations को वैध इनपुट क्लासेस के साथ चुनती है जो अस्थिरता या असामान्य प्रोसेसिंग की वजह से कारक हो सकता है।

User acceptance Testing–User Acceptance Testing, एप्लीकेशन से पहले फाइनल Step है। सामान्यत: end users एप्लीकेशन को स्वीकार करने से पहले एप्लीकेशन टेस्ट का प्रयोग करेंगे। इस Type की टेस्टिंग end users को आत्मविश्वास देती है तथा एप्लीकेशन उनकी आवश्यकताओं के अनुरूप डिलीवर होते हैं।

यह टेस्टिंग एप्लीकेशन के यूजेबिलिटी से सम्बन्धित nail bugs में भी सहायता करती है। ❑

Scanned by CamScanner

प्रश्न 4 – *Functional तथा Non-functional Testing के महत्व को समझाइये।*

Explain the importance of Functional & Non-functional testing?

उत्तर–Functional तथा Non-functional Testing–टेस्टिंग, बिजनेस आवश्यकताओं के against एप्लीकेशन है। Functional Testing, Functional specifications के प्रयोग से की जाती है जो Client या design specifications के द्वारा उपलब्ध करायी जाती है। Functional Testing निम्नलिखित बिन्दुओं को कवर करती है–

1. यूनिट टेस्टिंग
2. स्मोक टेस्टिंग
3. इंटीग्रेशन टेस्टिंग (टॉप डाउन, बॉटम अप टेस्टिंग)
4. इंटरफेस तथा यूजेबिलिटी टेस्टिंग
5. सिस्टम टेस्टिंग
6. रिग्रेशन टेस्टिंग
7. प्री एक्सेप्टेन्स टेस्टिंग
8. यूजर एक्सेप्टेंस टेस्टिंग
9. White box तथा Black-box टेस्टिंग
10. ग्लोबजाईशेन तथा लोकलाईजेशन टेस्टिंग

Non-functional Testing, Client तथा परफॉरमेन्स आवश्यकताओं के against एप्लीकेशन है। Non-functional टेस्टिंग Client के द्वारा परिभाषित आवश्यकताओं तथा Test scenarios पर आधारित की जाती है। Non-functional Testing निम्नलिखित बिन्दुओं को कवर करती है–

1. लोड तथा परफॉरमेन्स टेस्टिंग
2. Erogonomics टेस्टिंग
3. स्ट्रेस तथा वॉल्यूम टेस्टिंग
4. Compatibility तथा Migration टेस्टिंग
5. डाटा कनवर्जन टेस्टिंग
6. सिक्योरिटी/पेनीटरेशन टेस्टिंग
7. ऑपरेशनल Readiness टेस्टिंग
8. Installation Testing

Functional टेस्टिंग उस टेस्टिंग को रेफर करती है जिसमें निश्चित इनपुट वैल्यूज के लिये आउटपुट के केवल प्रेक्षण समाहित होते हैं। इसमें कोड का कोई विश्लेषण नहीं होता है जो आउटपुट को निर्मित करते हैं। हम कोड के आन्तरिक स्ट्रक्चर पर ध्यान नहीं देते हैं इसलिये फंक्शनल टेस्टिंग को Black-box टेस्टिंग की तरह भी रेफर किया जाता है, जिसमें Black-box के Contents पता नहीं होते हैं। जैसा कि निम्नांकित Fig. में Functional Testing को Black-box टैस्टिंग के रूप में दर्शाया गया है–

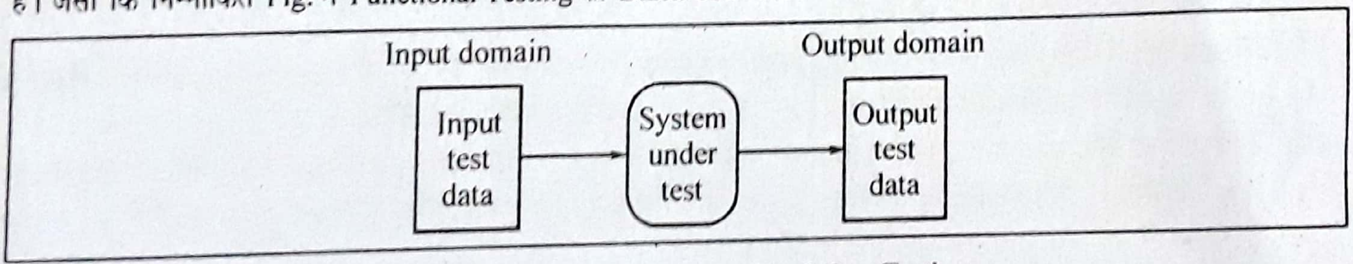

Fig. : Functional Testing as Black Box Testing

यहां पर हम कोड के इंटरनल स्ट्रक्चर के बजाय फंक्शनलिटी में रुचि रखते हैं। काफी हद तक हम Black-box Knowledge को अधिक प्रभावी रूप से ऑपरेट करते हैं। ❑

प्रश्न 5 – *Alpha तथा Beta testing की व्याख्या कीजिये।*

Explain Alpha & Beta testing.

उत्तर–Alpha Testing–Alpha Testing, प्रथम वैधानिकता परीक्षण को कहा जाता है। Alpha Testing में प्रथम वातावरण के तहत सॉफ्टवेयर को प्रणाली के उपयोगकर्ता के द्वारा बताई गयी स्थितियों व आवश्यकताओं के आधार पर परीक्षित कर उसकी अशुद्धियों की जांच व प्रणाली अध्ययन किया जाता है।

Alpha Testing में प्रणाली की गुणवत्ता की जांच की जाती है। Alpha Testing में प्रणाली की अशुद्धि का पता लगता है। Alpha Testing में प्रणाली के उपयोगकर्ता की आवश्यकता तथा प्रणाली का अध्ययन किया जाता है।

यदि यह सॉफ्टवेयर प्रोडक्ट के रूप में डेवलप किया जाता है जिसे बहुत सारे Customers के द्वारा प्रयोग किया जाता है तो प्रत्येक के Formal acceptance tests को परफॉर्म करना प्रेक्टिकल संभव नहीं है। अधिकतर सॉफ्टवेयर प्रोडक्ट बिल्डर्स एक प्रोसेस को प्रयोग करते हैं जिसे Alpha तथा Beta टेस्टिंग कहते हैं।

Scanned by CamScanner

Alpha test, end-users के द्वारा डेवलपर के साइट पर Conduct होती है। Alpha tests, नियंत्रित वातावरण में Conduct की जाती है।

Beta Testing–Beta Testing द्वितीय परीक्षण कहा जाता है। Beta Testing में अशुद्धियों के सॉफ्टवेयर को द्वितीय वातावरण अर्थात् असली वातावरण व उपयोग के साथ चलाया जाता है। कुछ समय या दिनों तक वास्तविक वातावरण में कार्य सम्पन्न होने के पश्चात् प्राप्त अशुद्धियों को सुधार कर व एक बार पुनः परीक्षित कर प्रणाली को उपयोगकर्त्ता के उपयोग के लिये जारी किया जाता है।

Beta Testing में प्रणाली की गुणवत्ता रहती है। Beta Testing में प्रणाली के उपयोगकर्त्ता की आवश्यकता पूर्ण कर दी जाती है।

Beta test, end-user site पर Conduct की जाती है। Alpha Testing के विपरीत इसमें डेवलपर सामान्यतः उपस्थित नहीं होता है। इसलिये Beta test, एक वातावरण में सॉफ्टवेयर का live एप्लीकेशन है जो डेवलपर के द्वारा नियंत्रित नहीं किया जाता है। end-user सभी Problems को रिकॉर्ड करता है जो Beta Testing के दौरान उत्पन्न होती है तथा एक रेगुलर इंटरवल के दौरान डेवलपर को रिपोर्ट करता है। Beta test के दौरान रिपोर्ट की गयी समस्याओं को सॉफ्टवेयर इंजीनियर्स के द्वारा संशोधित करके इन्हें Customer base पर सॉफ्टवेयर प्रोडक्ट के रूप में तैयार करते हैं।

Alpha तथा Beta टेस्टिंग तब प्रयोग की जाती हैं, जब Customers के लिये सॉफ्टवेयर एक प्रोडक्ट के रूप में डेवलप किया जाता है। Alpha tests, Customer के द्वारा developer site पर Conduct किया जाता है। ये Tests नियंत्रित वातावरण में Conduct किये जाते हैं। Formal Testing प्रोसेस के पूरा होने के लगभग Alpha Testing को शुरू किया जा सकता है। अधिकतर Companies इस Practice को सर्वप्रथम करती है। ये अपने प्रोडक्ट के Beta को कुछ महीनों के लिये भेजते हैं। बहुत-से Potential Customers, प्रोडक्ट को प्रयोग करेंगे तथा Product के बारे में अपने Views भेजते हैं। ❑

प्रश्न 6 – *लोड टेस्टिंग क्या है? Load runner अवयवों को समझाइये।*

What is Load testing ? Explain the components of Load Runner.

उत्तर–लोड टेस्टिंग (Load Testing)–लोड टेस्टिंग एक सिस्टम पर डिमाण्ड को रखने की प्रक्रिया को या डिवाइस तथा इसके रेस्पांस को मापने की प्रक्रिया है। जब लोड सिस्टम पर Place किया जाता है तो System का रेस्पांस high या peak load के रूप में होता है तो इसे Stress Testing कहते हैं। सामान्यतः load इतना ज्यादा होता है कि error conditions आशान्वित रिजल्ट को प्रभावित करती है, यद्यपि जब load test की एक्टिविटी होती है तथा stress test होता है तो इसकी कोई सीमा स्पष्ट नहीं है।

Load टेस्टिंग के Specific goals के लिये थोड़े-से एग्रीमेन्ट है।

Load टेस्टिंग को प्रोफेशनल टेस्टिंग कम्यूनिटी में विभिन्न रूपों में प्रयोग किया जाता है। Load Testing सामान्यतः एक सॉफ्टवेयर प्रोग्राम की आशाओं की प्रेक्टिस को रेफर करती है जो प्रोग्राम में Concurrently मल्टीपल यूजर्स के द्वारा उत्पन्न होती है। यह टेस्टिंग मल्टीयूजर सिस्टम्स के लिये Client/Server मॉडल, वेब सर्वर के अनुरूप है। हालांकि दूसरे Types के सॉफ्टवेयर सिस्टम्स भी लोड टेस्ट कर सकते हैं; जैसे–

Word Processor या ग्राफिक्स एडिटर अत्यधिक बड़े डॉक्यूमेण्टों को पढ़ने के लिये Force कर सकते हैं।

यदि एक वेब साइट 100 users को एक्सेस करती है, जो निम्नलिखित फंक्शनों को करती है–

1. 25 वर्चुअल यूजर्स items तथा logging off के माध्यम से ब्राउज कर रहे हैं।
2. 25 वर्चुअल यूजर्स shopping cart में आइटमों को add कर रहे हैं तथा logging off को चेकआउट कर रहे हैं।
3. 25 वर्चुअल यूजर्स आइटमों की खरीददारी करके वापिस आ रहे हैं तथा लॉग-ऑफ कर रहे हैं।
4. 25 वर्चुअल यूजर्स बिना किसी एक्टिविटी के अभी लॉग किया है।

कभी-कभी इसे Non-Functional टेस्टिंग की कहते हैं।

इस वर्चुअल यूजर्स को जनरेट करने के लिये विभिन्न प्रकार के टूल्स उपलब्ध हैं। एप्लीकेशन 100 वर्चुअल यूजर्स लोड के लिये जिन्हें ऊपर दिखाया गया है, Subjected हैं तथा इनकी परफॉरमेन्स को मॉनीटर किया जाता है।

Load Runner, सिस्टम व्यवहार तथा परफॉरमेन्स के लिये इण्डस्ट्री स्टैण्डर्ड परफॉरमेन्स तथा लोड टेस्टिंग प्रोडक्ट है। Load Runner, वेब सर्विस, J2EE तथा .NET को सपोर्ट करता है। ERP/CRP एप्लीकेशनस के लिये केवल यही प्रमाणित टेस्टिंग प्रोडक्ट है।

Load Runner के द्वारा हम सिस्टम परफॉरमेन्स की सही पिक्चर को प्राप्त कर सकते हैं तथा Specified परफॉरमेन्स आवश्यकताओं को अपग्रेड या नयी एप्लीकेशनस प्राप्त कर सकते हैं। ❑

प्रश्न 7 – *Spiral Model से आप क्या समझते हो? इसके लाभ तथा हानियां क्या हैं?*

What is spiral model? Explain its advantages and disadvantages.

उत्तर–Spiral Model–Spiral Model को 1985 में Barry Boehm के द्वारा विकसित किया गया था। Spiral Model एक Coiled Curve के रूप में ग्राफीकल रूप में प्रदर्शित किया जाता है। Spiral Model, evolutionary सॉफ्टवेयर डेवलपमेन्ट मॉडल्स के

Scanned by CamScanner

टाईप का एक मॉडल है जो सॉफ्टवेयर डेवलपमेन्ट के लिये कार्य करता है। यह मॉडल बड़े, कीमती तथा Complicated प्रोजेक्ट्स के अनुरूप होता है। यह मॉडल एक महत्वपूर्ण मैनेजमेन्ट कंट्रोल को उपलब्ध कराता है। यह प्रोटोटाइपिंग, Multiple iterations में सॉफ्टवेयर को रिलीज करना, मॉड्यूल्स तथा सिस्टम की Stepwise टेस्टिंग, रिस्क एनालिसिस तथा कंट्रोल एक्टिविटीज, कस्टमर के द्वारा लगातार review, डेवलपमेन्ट प्रोसेस के दौरान Specifications का संशोधन करने के विकल्प के फायदे को उपलब्ध कराता है।

यह सॉफ्टवेयर डेवलपमेन्ट की Stepwise एप्रोच को संभालता है लेकिन इसे एक Iterative फ्रेमवर्क में संयोजित करता है। Spiral SDLC मॉडल में प्रत्येक Spiral Cycle चार मुख्य Phases को Cover करता है–

(i) Planning, (ii) Reviewing, (iii) Risk analysis (iv) Construction/Engineering.

प्रत्येक Spiral cycles, iteration के लिये Planning के साथ start करता है। Iteration तथा Road map के उद्देश्यों को Plan किया जाता है। निम्नांकित Fig. में Spiral Model को प्रदर्शित किया गया है–

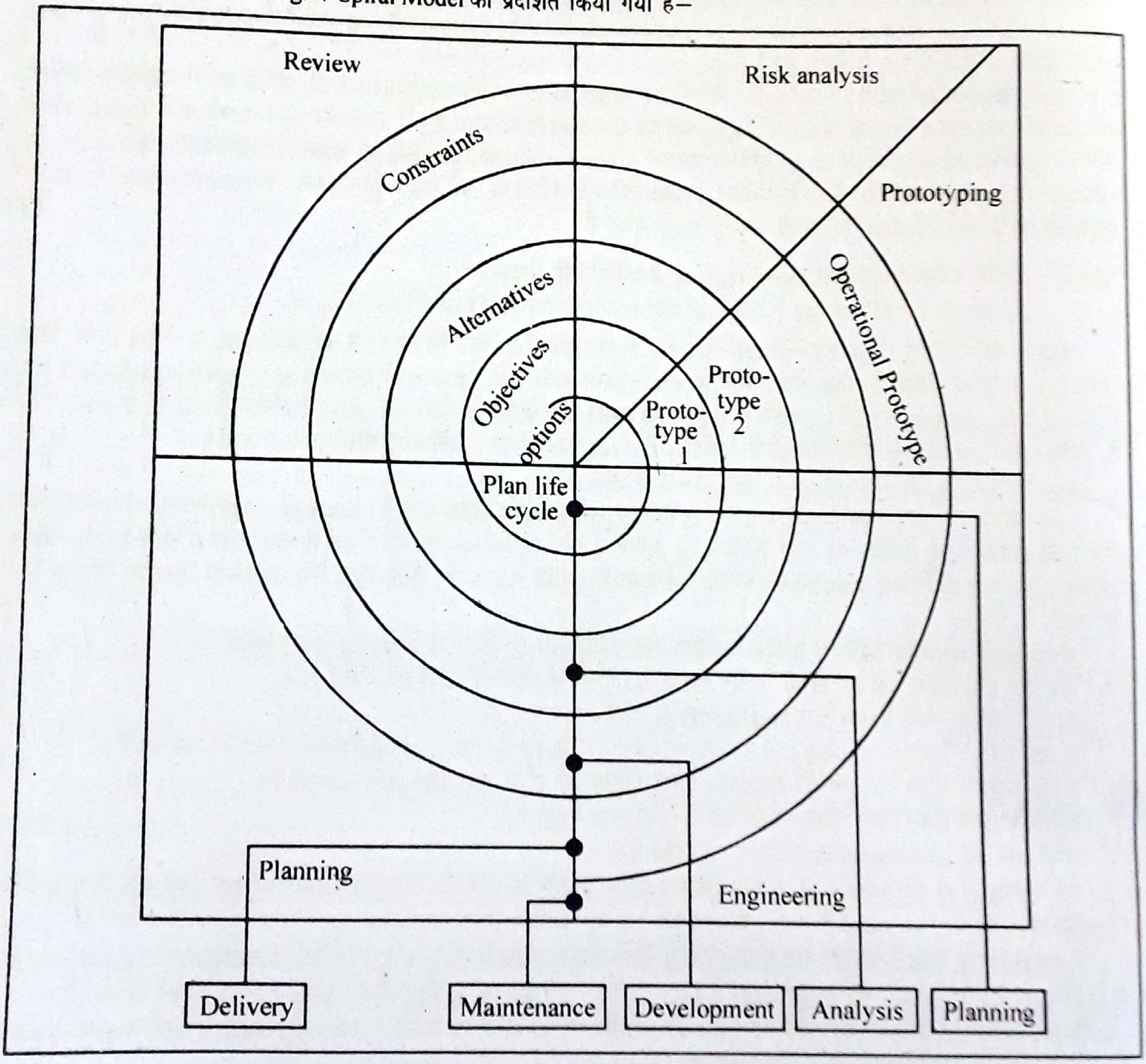

Fig. : Spiral SDLC Model

Spiral Model के लाभ–Spiral Model के प्रमुख लाभ निम्नलिखित हैं–

1. Complicated तथा Large सॉफ्टवेयर को बनाने के लिये Spiral model एक अच्छी पद्धति है।

2. Spiral Model में Complex सॉफ्टवेयर की प्रोटोटाइपिंग सम्मिलित होती है। अतः इसके लिये अलग से प्रोटोटाइप मॉडल की आवश्यकता नहीं होती है।

Scanned by CamScanner

3. Spiral Model में Classical life cycle पद्धति का प्रयोग किया गया है।
4. Spiral Model में Risk analysis तथा Control को आसानी से हल किया गया है।

Spiral Model की हानियां–Spiral Model की प्रमुख हानियां निम्नलिखित हैं–

1. Customer को यह समझना मुश्किल होता है कि इसमें Risk Factor को कंट्रोल किया गया है।
2. इसमें Risk analysis को हल करने की विश्वसनीयता यूजर की कुशलता पर निर्भर है।
3. यह Waterfall model या Prototype model की तरह Multiple module में प्रयुक्त किया जाने वाला मॉडल नहीं है। ❑

प्रश्न 8 – *Software की quality assurance से आप क्या समझते हैं? इसके विभिन्न चरणों को समझाइये।*

What do you understand by software quality assurance. Explain its various steps.

उत्तर–वर्तमान में प्रणाली को कम्प्यूटरीकृत करने के लिये विकसित किये जाने के लिये सामान्यतः प्रोग्रामर के पास अत्यन्त कम समय रहता है। प्रोग्रामों की जटिलता भी दिन-प्रतिदिन बढ़ती जा रही है। इसके अतिरिक्त भी अनेक कई कारणों से सॉफ्टवेयरों को विकसित करने में अपनाई जाने वाली सामान्य व्यूह रचना नहीं अपनाई जाती है।

प्रणाली की गुणवत्ता निर्धारक तत्व (Quality Factor Specification)

(i) शुद्धता (Correctness)–सॉफ्टवेयर द्वारा उपलब्ध कराया गया आउटपुट पूर्ण शुद्ध होना चाहिये।

(ii) विश्वसनीयता (Reliability)–सॉफ्टवेयर की कार्यप्रणाली व कार्यक्षमता विश्वास योग्य होनी चाहिये।

(iii) उपयोगिता (Usability)–सॉफ्टवेयर में उसकी क्षमताओं के अनुरूप कार्य करने की योग्यता होनी चाहिये, जिससे अधिक से अधिक उपयोग किया जा सके।

(iv) रखरखाव (Maintenance)–सॉफ्टवेयर का रखरखाव आसान व कम खर्चीला होना चाहिये।

(v) लाना-लेजाना (Portability)–सॉफ्टवेयर को एक कम्प्यूटर से दूसरे कम्प्यूटर पर ले जाने में किसी कठिनाई से स्थापित करने की सुविधा होनी चाहिये।

(vi) मित्र व्यवहार (User friendly)–सॉफ्टवेयर चलाने में सरल होने चाहियें, जिससे बिना अधिक प्रशिक्षण के कर्मचारी स्वयं ही सॉफ्टवेयर द्वारा दिये जा रहे निर्देशों के अनुसार उसे चला सके।

(vii) डाटा एकत्रीकरण (Compilation of Data)–डाटा एकत्रीकरण की विधि व वातावरण प्रणाली के लिये आवश्यक रूप में उचित होने चाहिये।

(viii) डाटा वैधता (Validity of Data)–डाटा संसाधन प्रक्रिया के पूर्व समस्त इनपुट डाटा की वैधता की जांच भली-भाँति होनी चाहिये।

(ix) अशुद्धि रिपोर्ट (Error Report)–डाटा वैधता व डाटा संसाधन के समय प्राप्त होने वाली अशुद्धियों की रिपोर्ट प्रणाली को स्वयं तैयार करने में सक्षम होनी चाहिये।

(x) डाटा सुरक्षा (Data Safety)–प्रणाली में इनपुट किये गये डाटा को किसी प्रकार से अवांछित व्यक्ति नुकसान न पहुंचा पाये, इस बात का इंतजाम किया जाना चाहिये।

(xi) विस्तार सुविधा (Expandability)–आवश्यकता पड़ने पर प्रणाली की कार्यक्षमता व उसके डाटाबेस का विस्तार आसानी से किया जा सके, इस बात की सुविधा होनी चाहिये। ❑

प्रश्न 9 – *Software maintenance से आप क्या समझते हैं?*

What is software Maintenance?

उत्तर–Software maintenance का सबसे कठिन प्रश्न है जिसने सॉफ्टवेयर उद्योग को अपने बंधन में जकड़ रखा है और उपलब्ध संसाधनों के एक बड़े भाग को अपने से जोड़कर रखा है। वर्तमान में प्रणाली विश्लेषक व प्रोग्रामर, प्रोग्रामों के रख-रखाव, पर प्रोग्राम लिखने में लगाये गये समय की अपेक्षा कहीं अधिक समय लगा रहे हैं। सॉफ्टवेयर विकास का लगभग 50-80 प्रतिशत भाग सॉफ्टवेयर रख-रखाव के खाते में सामान्यतः चला जाता है।

सॉफ्टवेयर रख-रखाव की प्रक्रिया में आने वाली कुछ समस्यायें हैं, जिनके कारण यह प्रक्रिया कठिन मानी जाती है, निम्नलिखित हैं–

1. सॉफ्टवेयर विकसित करने की अपेक्षा सॉफ्टवेयर रखरखाव प्रक्रिया में प्रोग्रामर को लाभ कम रहते हैं।
2. सॉफ्टवेयर रख-रखाव के लिये अत्यंत कम साधन व तकनीकी उपलब्ध हैं।
3. सॉफ्टवेयर रख-रखाव की ऊंची लागत होने के कारण उपयोगकर्त्ता उस पर कम ध्यान देते हैं।
4. सॉफ्टवेयर अच्छे प्रकार से दस्तावेजित नहीं होता है।

अनेक बार प्रोग्रामर भी सॉफ्टवेयर रख-रखाव को निम्न स्तर का कार्य मानते हैं और जूनियर प्रोग्रामरों को यह कार्य सौंप देते हैं।

Scanned by CamScanner

सबसे बेहतर तो यह होता है कि शुरूआत से ही प्रणाली पूरी एकाग्रता व समस्त पूर्ण निर्धारित आवश्यक तत्वों को ध्यान में रखकर विकसित की जाये, जिससे रख-रखाव की आवश्यकता भी कम से कम पड़े।

रखरखाव तथा उन्नति (Maintenance or Enhancement)–सॉफ्टवेयर रख-रखाव को तीन भागों में वर्गीकृत किया जा सकता है–सुधारात्मक, अनुकूलात्मक तथा परिपूर्णात्मक। सुधारात्मक रख-रखाव से तात्पर्य डाटा संसाधन के कार्य या प्रणाली के प्रदर्शन में आ रही असफलता को दूर करने से है। अनुकूलात्मक रख-रखाव से तात्पर्य किसी प्रोग्राम की प्रक्रिया को परिवर्तित करने से है तथा परिपूर्णात्मक रख-रखाव से तात्पर्य प्रोग्राम की क्षमता में वृद्धि करने व उपयोगकर्त्ता को अधिक सुविधायें प्रदान करने से है।

रख-रखाव खर्च न्यूनतम करना (Reducing the Maintenance Cost)–सॉफ्टवेयर रख-रखाव की कठिनाई का सफलतापूर्वक सामना करने के लिये एक योजना का होना आवश्यक है।

कई संस्थान यह कार्य रख-रखाव की आवृत्ति न्यूनतम रखकर करते हैं, इस प्रक्रिया में निम्नलिखित तत्व शामिल हैं–

1. रख-रखाव प्रबंध जांच (Maintenance Management test)–इसके अंतर्गत साक्षात्कार व प्रश्नोत्तरी रख-रखाव की कठिनाई का सफलतापूर्वक सामना करने के लिये एक योजना का होना आवश्यक है। कई संस्थान यह कार्य, रख-रखाव की आवृत्ति न्यूनतम रखकर करते हैं। इस प्रक्रिया में निम्न तत्व शामिल होते हैं–

(a) क्या प्रणाली रख-रखाव के सम्बंध में किसी रजिस्टर का निर्माण किया गया है?
(b) कुल समय का कितने प्रतिशत समय प्रणाली रख-रखाव पर व्यय होता है?
(c) क्या प्रणाली रख-रखाव को न्यूनतम रखने के लिये किसी तरह की योजना वर्तमान में कार्यरत है?

इस प्रकार एकत्रित किया गया डाटा प्रणाली रख-रखाव की प्रबंध योजना बनाने में सहायक होता है।

2. सॉफ्टवेयर प्रणाली जांच (Software System Test)–इसके अंतर्गत सम्मिलित तत्व निम्न हैं–

(a) प्रणाली के दस्तावेजों, डाटा फाइलों की गुणवत्ता, डाटाबेस, प्रणाली विश्वसनीयता व कार्यक्षमता का पूर्ण निरीक्षण
(b) समस्त प्रोग्रामों की अलग-अलग जांच, यह ज्ञात करने के लिये कि वे किस तरह व किसी सीमा तक अपना कार्य कर रहे हैं?

3. सॉफ्टवेयर सुधार (Software Modification)–इसके अंतर्गत निम्न तत्व सम्मिलित रहते हैं–

(a) आवश्यकतानुसार प्रोग्राम पुनः लिखना।
(b) प्रणाली दस्तावेजों में सुधार।
(c) प्रोग्रामों की पुनः जांच।

प्रश्न 10 – इंक्रीमेन्ट प्रोसेस मॉडल क्या है? इसके लाभ तथा हानियां लिखिये।

What is increment model ? Write its advantages and disadvantages.

उत्तर–इंक्रीमेन्ट प्रोसेस मॉडल्स (Increment Process Models)–इंक्रीमेन्ट प्रोसेस मॉडल, उस स्थित में प्रभावित होते हैं जहां आवश्यकतायें सही तरीके से परिभाषित होती हैं तथा फाईनल प्रोडक्ट की functionality के बारे में कोई भी कनफ्यूजन नहीं होता है। प्रत्येक Cycle के पश्चात् एक प्रयोग होने योग्य प्रोडक्ट कस्टमर को दिया जाता है; जैसे–यूनिवर्सिटी ऑटोमेशन सॉफ्टवेयर लाइब्रेरी ऑटोमेशन मॉड्यूल प्रथम फेज में डिलीवर हो सकता है तथा एक्जामिनेशन ऑटोमेशन मॉड्यूल द्वितीय फेज में डिलीवर हो सकता है, इसी तरह प्रत्येक मॉड्यूल अलग-अलग फेज में दिया जा सकता है।

इंक्रीमेन्ट प्रोसेस मॉड्ल्स विशेष रूप से तब पॉपुलर हैं जब हमें एक लिमिटेड functionality सिस्टम की डिलीवरी की जल्दी है। निम्नांकित Fig. में इस मॉडल के प्रारूप को दर्शाया गया है–

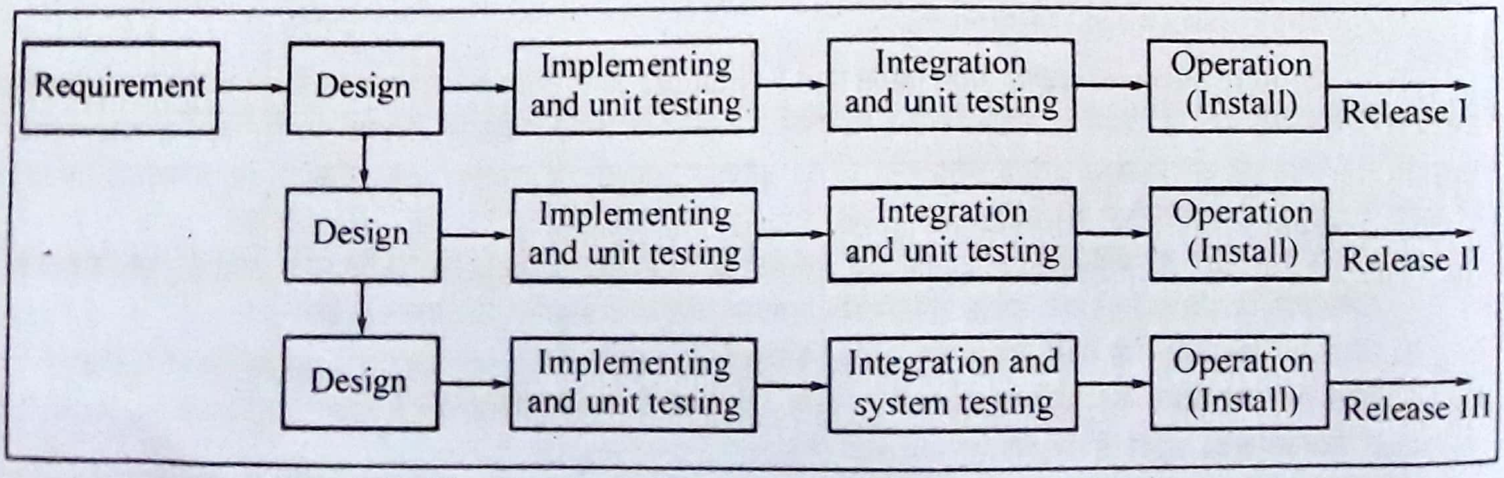

Fig. : Increment Process Model (Iterative)

Scanned by CamScanner

Increment Model के लाभ—Increment Model के प्रमुख लाभ निम्नलिखित हैं—

1. Increment Model में प्रारम्भ में ही सॉफ्टवेयर के मॉड्यूल्स उपलब्ध होते हैं।
2. Increment Model में सॉफ्टवेयर डेवलपमेन्ट के मॉड्यूल में कोई कनफ्यूजन नहीं होता है।
3. इस मॉडल में प्रत्येक मॉड्यूल की डिलीवरी के साथ ऑपरेशनल प्रोडक्ट की डिलीवरी पर ध्यान केन्द्रित होता है।
4. यह मॉडल शुरू में यूजर फीडबैक के आधार पर System Error को कंट्रोल करता है।

Increment Model की हानियां—Incremental Model की प्रमुख हानियां निम्नलिखित हैं—

1. यह मॉडल सॉफ्टवेयर्स डेवलपर्स के लिये स्थापित किये Contractual Model के साथ उपयुक्त नहीं होता है।
2. यह मॉडल विशेषतः उस स्थिति में सही होता है जब प्रोजेक्ट के लिये निर्धारित की गयी सीमा तक बिजनेस के लिये किये जाने वाले सभी कार्यों के लिये Employee उपलब्ध न हो।
3. इस मॉडल में आवश्यकतायें पूर्ण होने के पहले सिस्टम आर्कीटेक्चर को स्थापित करना अनिवार्य होता है। ❑

प्रश्न 11 – सॉफ्टवेयर टेस्टिंग की स्ट्रेटजिक एप्रोच (Strategic approach) के बारे में समझाइये।

Explain strategic approach of Software testing?

उत्तर—टेस्टिंग, एक्टिविटीज (Activities) का एक सेट है जो एडवांस में प्लान की जाती है तथा सही तरीके से पूर्ण की जाती है। अतः हम कह सकते हैं कि सॉफ्टवेयर टेस्टिंग, Steps का एक set है जिसमें हम विशिष्ट Test Case डिजाइन तकनीकियों को स्थापित कर सकते हैं। विभिन्न प्रकार की सॉफ्टवेयर Testing strategy का विकास हो चुका है। सभी Testing strategy के सामान्य प्रमुख लक्षण निम्नलिखित हैं—

1. प्रभावी टेस्टिंग को पूरा करने के लिये, सॉफ्टवेयर टीम को प्रभावी फॉर्मल टेक्निकल रिव्यूज को Conduct करना चाहिये। ऐसा करने से बहुत सारी गलतियां टेस्टिंग के शुरू होने से पूर्व ही समाप्त हो जायेंगी।
2. टेस्टिंग, कम्पोनेन्ट लेवल पर शुरू होती है, तथा पूरे कम्प्यूटर बेस्ड सिस्टम के इंटीग्रेशन के चारों ओर कार्य करती है।
3. विभिन्न टेस्टिंग तकनीकियां, समय के विभिन्न बिन्दुओं पर appropriate होती हैं।
4. टेस्टिंग, सॉफ्टवेयर के डेवलपर के द्वारा Conduct की जाती है।
5. टेस्टिंग तथा डिबगगिंग भिन्न एक्टिविटीज हैं, लेकिन डिबगगिंग किसी भी Testing strategy में अवश्य होनी चाहिये।

सॉफ्टवेयर टेस्टिंग के लिये स्ट्रेटजी (Strategy) को Low-level tests को रखना चाहिये जो यह संतुष्टि करने के लिये आवश्यक है कि छोटे सोर्स कोड समूह सही तरीके से इम्प्लीमेन्ट हुये हैं। मैनेजर के लिए Milestones के set तथा अभ्यासकर्त्ताओं के लिये दिशा निर्देश, Strategy को अवश्य उपलब्ध कराने चाहिये। क्योंकि Strategy test के steps उस समय लागू होते हैं जब deadline का दबाव उत्पन्न होता है।

सॉफ्टवेयर टेस्टिंग को Strategic approach के प्रमुख बिन्दु निम्नलिखित हैं—

(a) Verification and Validation—सॉफ्टवेयर टेस्टिंग, broader topic का एक एलीमेन्ट है जो अक्सर Verification and validation के रूप में रेफर किया जाता है। Verification, एक्टिविटीज के सेट को रेफर करता है जो सुनिश्चित करता है कि सॉफ्टवेयर एक Specific function को सही तरीके से इम्प्लीमेन्ट करता है। Validation, एक्टिविटीज के भिन्न-भिन्न सेट को रेफर करता है, जो सुनिश्चित करता है कि जो सॉफ्टवेयर बनाया गया है वह कस्टमर की आवश्यकतानुसार Traceable है या नहीं है।

(b) Organizing for Software Testing— टेस्टिंग के शुरू होने से पहले प्रत्येक सॉफ्टवेयर प्रोजक्ट के लिये एक आन्तरिक कलह होती है। व्यक्ति, जिसने सॉफ्टवेयर को बनाया है उससे सॉफ्टवेयर टेस्ट के लिये पूछा जाता है। यह स्वयं को harmless प्रतीत होता है।

सॉफ्टवेयर इंजीनियर, Models को Analyzes करता है। इसके पश्चात् कम्प्यूटर प्रोग्राम तथा इसके दस्तावेजों को तैयार करता है।

(c) Conventional Software Architecture के लिये सॉफ्टवेयर टेस्टिंग स्ट्रेटजी (Strategy)— सॉफ्टवेयर प्रोसेस को Spiral के रूप में देखा जा सकता है, जैसा कि संलग्न Fig. में प्रदर्शित किया गया है।

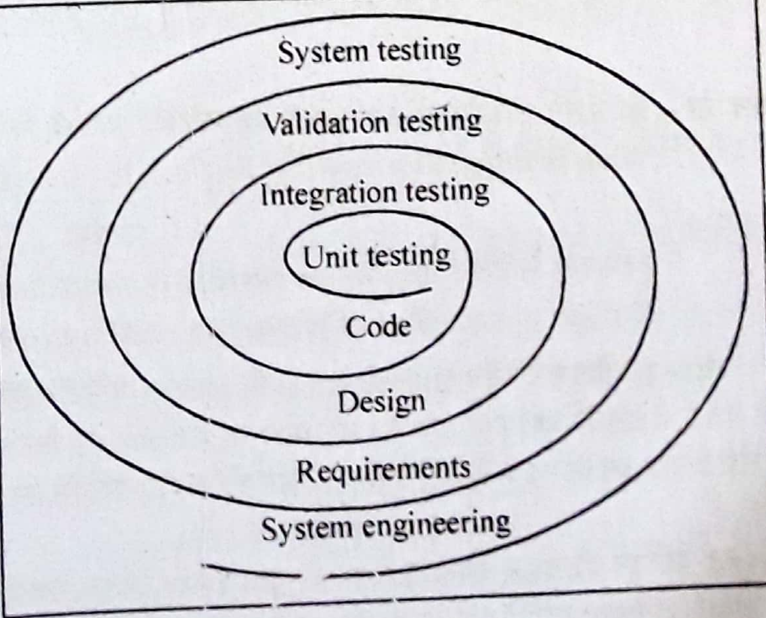

Fig. : Testing Strategy

Scanned by CamScanner

प्रारम्भ में सिस्टम इंजीनियरिंग, सॉफ्टवेयर के रोल को परिभाषित करता है तथा सॉफ्टवेयर आवश्यकताओं विश्लेषण को लीड करता है, जहां सॉफ्टवेयर के लिये इनफॉरमेशन डोमेन, फंक्शन, व्यवहार, परफॉरमेन्स, Constraints, Validation Criteria इत्यादि स्थापित किये जाते हैं। Spiral के अनुसार हम डिजाइन पर आते हैं तथा अंततः कोडिंग पर पहुंचते हैं। ❑

प्रश्न 12 – सॉफ्टवेयर टेस्टिंग में सम्मिलित Steps के बारे में विस्तारपूर्वक समझाइये।

Explain in detail the steps included in Software testing?

उत्तर–सॉफ्टवेयर इंजीनियरिंग के कॉन्टेक्स्ट के अनुसार वास्तविक रूप में टेस्टिंग के चार प्रमुख Steps होते हैं जो क्रमिक रूप में इम्प्लीमेन्ट किये जाते हैं। निम्नलिखित Fig. में ये चारों Steps स्पष्ट रूप से दिखायी देते हैं–

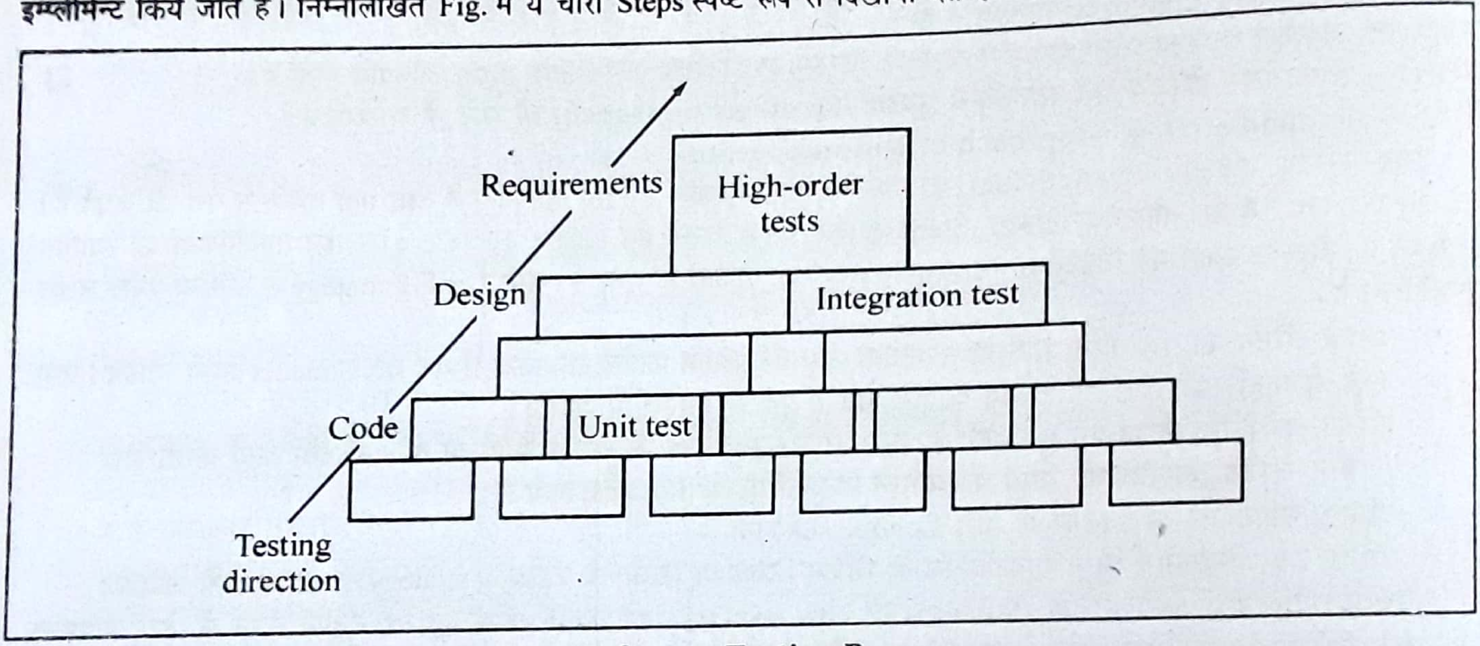

Fig. : Software Testing Process

प्रारम्भ में Test प्रत्येक कम्पोनेन्ट पर व्यक्तिगत रूप से फोकस करता है तथा सुनिश्चित करता है कि ये फंक्शनूस एक यूनिट के रूप में है। अतः इसे Unit testing कहते हैं। आगे कम्पोनेन्ट्स को कम्पलीट सॉफ्टवेयर पैकेज के रूप में असेम्बल या इंटीग्रेट करना चाहिये। Test Case डिजाइन तकनीकी Inputs तथा Outputs पर फोकस करती है। सॉफ्टवेयर के Construct हो जाने के पश्चात् High-order tests के सेट को Conduct किया जाता है। Validation Criteria को अवश्य Evaluate करना चाहिये। Validation testing, अन्तिम विश्वसनीयता को उपलब्ध कराती है जो सभी फंक्शनल, व्यावहारिक तथा परफॉरमेन्स आवश्यकताओं को सॉफ्टवेयर से मिलाती है।

अन्तिम Step, high-order टेस्टिंग सॉफ्टवेयर इंजीनियरिंग की Boundary को पूर्ण करता है। ❑

प्रश्न 13 – इंटीग्रेशन टेस्टिंग से आप क्या समझते हैं? इसकी विभिन्न विधियाँ बताइये।

What is Integration testing. Explain it various methods.

अथवा *(Or)*

उदाहरण सहित Integration testing approaches को समझाइये।

Explain approaches of Integration testing with example.

उत्तर–इंटीग्रेशन टेस्टिंग (Integration Testing)–यूनिट टेस्टिंग का उद्देश्य यह निर्धारित करना है कि प्रत्येक स्वतन्त्र मॉड्यूल सही तरीके से इम्प्लीमेन्ट किया गया है। यह थोड़ा-सा Chance यह डिटरमाइन करने के लिये देता है कि मॉड्यूलों के मध्य का इंटरफेस भी सही है तथा इस कारण के लिये इंटीग्रेशन टेस्टिंग को अवश्य परफॉर्म करना चाहिये। इंटीग्रेशन टेस्टिंग का एक Specific टॉरगेट इंटरफेस है।

कई वर्गीकृत इंटीग्रेशन रणनीति हैं जो वास्तव में एक रेशनल मैथडोलॉजी में थोड़े basics को रखते हैं।

Scanned by CamScanner

निम्नांकित Fig. में तीन विभिन्न इंटीग्रेशन एप्रोच को दर्शाया गया है–

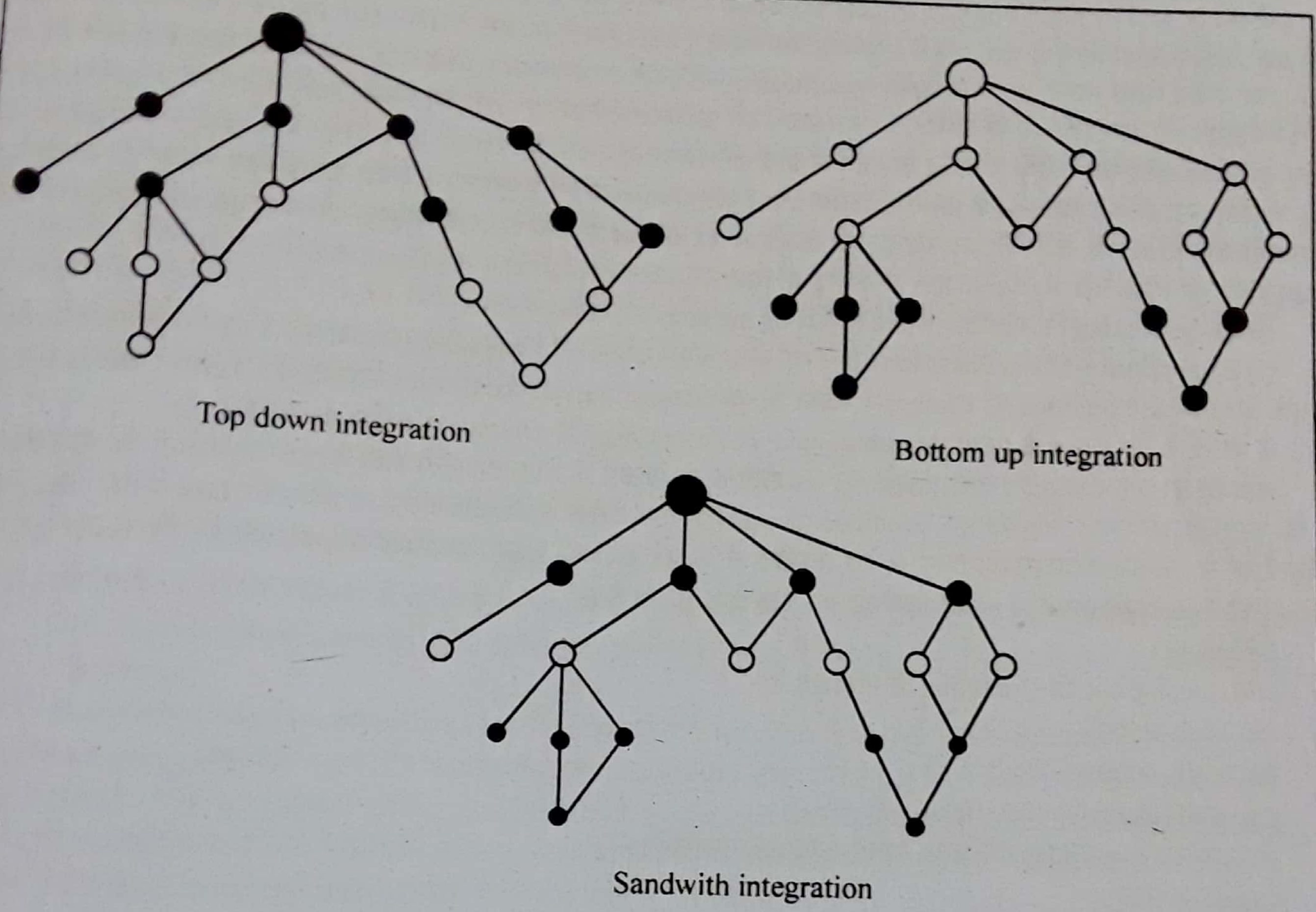

Fig. : Three different integration approaches

इंटीग्रेशन टेस्टिंग में हम स्ट्रक्चरल टेस्टिंग से धीरे-धीरे मूव करते हैं तथा फंक्शनल टेस्टिंग के चारों ओर जो एक मॉड्यूल को, एक फंक्शन को पूरा करने के लिये एक विधि की तरह Treat करती हैं।

क्योंकि ऑब्जेक्ट ओरियेन्टेड सॉफ्टवेयर एक स्पष्ट हायरैरिकल कण्ट्रोल स्ट्रक्चर को नहीं रखता है। इंटीग्रेशन टेस्टिंग के लिये दो विभिन्न रणनीतियां हैं–

(i) Thread-based Testing (ii) Use-based Testing

(i) Thread-based Testing–Thread-based Testing सिस्टम के लिये इवैंट या इनपुट को रेस्पांस के लिये आवश्यक क्लासों के सेट को इंटीग्रेट करता है। प्रत्येक Thread, व्यक्तिगत रूप से इंटीग्रेट किया जाता है।

(ii) Use-based Testing–Use-based Testing, उन classes की टेस्टिंग के द्वारा सिस्टम के Construction से शुरू होती हैं जो बहुत कम प्रयोग की जाती है।

स्वतंत्र Classes के टेस्ट हो जाने के पश्चात् Classes की अगली लेयर dependent classes कहलाती हैं, जो स्वतंत्र क्लासों को प्रयोग करती है, टेस्ट की जाती है। डिपेन्डेन्ट क्लासों की टेस्टिंग लेयर्स का Sequence तब तक Continue रहता है जब तक पूरा सिस्टम Construct नहीं हो जाता है। ☐

प्रश्न 14– Black box testing की निम्न तकनीकों को उपयुक्त उदाहरणों के साथ समझाइये :

Explain about the following methods of Black box testing with suitable examples :

(i) Equivalence class partitioning (ii) Boundary value analysis

उत्तर–Black-Box Testing–यह एक सॉफ्टवेयर टेस्टिंग तकनीक है जहां पर आइटम की आन्तरिक क्रियाओं के होने वाले test

Scanned by CamScanner

के बारे में, tester के द्वारा पता नहीं चलता है। इसमें कोड पर ध्यान नहीं दिया जाता है, test cases की डिजाइनिंग में केवल इनफॉरमेशन का प्रयोग किया जाता है। Black-box test, System तथा Black box के व्यवहार के test design करता है। Black-box टेस्टिंग सॉफ्टवेयर को एक "black-box" की तरह treat करती है अर्थात् बिना किसी इंटरनल इम्पलीमेन्टेशन के यह टेस्टिंग इम्पलीमेन्ट होती है। इसमें Test cases का प्रयोग केवल सॉफ्टवेयर के फंक्शनल specification को प्रयोग करके डिजाइन किये जाते हैं अर्थात् सॉफ्टवेयर के बिना किसी इंटरनल स्ट्रक्चर के द्वारा डिजाइन किये जाते हैं। इस कारण इसे फंक्शनल टेस्टिंग भी कहते हैं। इसका Test case केवल Input/Output वैल्यू को examine करता है तथा डिजाइन या कोड के ज्ञान की जरूरत नहीं होती है। Black-box टेस्टिंग में, black-box के contents की जानकारी नहीं होती है तथा box के function इसके Input और Output के रूप में पूर्ण रूप से समझे जाते है। हम मॉड्यूल की आन्तरिक जानकारी को check नहीं करते हैं। हम मॉड्यूल की outer detail को check करते हैं। यदि मॉड्यूल की आउटपुट सही नहीं है तब हम मॉड्यूल को आन्तरिक रूप से check करते हैं अर्थात् मॉड्यूल की इंटरनल रिलेशनशिप को check करते हैं।

Black-box testing निम्नलिखित types के testing methods को cover करती है–

(i) Equivalence Class Partitioning– यह ब्लैक बॉक्स टेस्टिंग की Case selection तकनीक है। इस तकनीक में Tester अलग-अलग Input Condition को Classes में बाँटता है, यह Clases Equivalence Classes कहलाती है। Tester Classes को इस तरह से बाँटता है कि Class के प्रत्येक Member Input की Processing और Output का तरीका same हो।

आमतौर पर एक Class उन सभी Inputs का Set होती है जो सिस्टम के लिए एक जैसी होती है। इस तकनीक में यह माना जाता है कि अगर सिस्टम एक Class की एक condition या case को गलत तरीके से handle करेगा तो वह उस Class के सभी cases को गलत तरह से handle करेगा। इसी कारण से इस तकनीक में Test Cases की संख्या अत्यधिक कम हो जाती है और Tester कुछ ही Cases को Test करके ज्यादा से ज्यादा गलतियों का पता लगा सकता है।

Example :

अगर Income tax की Condition निम्नलिखित है–

Rs. 1000–2500 → No Tax

Rs. 2501–4000 → Tax 5%

Rs. 4001–above → Tax 10%

अब हम इन conditions को अलग-अलग Classes में बाँटेंगे।

Class1–0-999

Class2–1000-2500

Class3–2501-4000

Class4–>40001

जैसा की हम जानते है कि यह मुमकिन नहीं है कि प्रत्येक Value को Test करके देखे। इसलिए हम यह Test करते है कि एक ही Class के Inputs का Output Similar हो। अब Input की गयी Values को Class के अनुसार बाँट ले और प्रत्येक Class से एक या दो Input को Check करे। अगर Outputs हमारी अपेक्षा के अनुसार है तो उस Class के सभी Inputs का Output Similar होगा। इस तरह हर Class के Input को Test किया जाता है।

(ii) Boundary value Analysis– इस टेस्टिंग मैथड् में test cases को equivalance class से एक इनपुट के रूप में सलेक्ट करते हैं। इस तरीके में इनपुट equivalence class की edge के रूप में lie करते हैं। दूसरों शब्दों में हम यह कह सकते हैं कि इस तकनीक में Tester Extreme Values का चयन करता है, जैसे– Maximum value, Minmum value, Just inside/Outside boundries, Typical values & Error values. इस तकनीक का आधार यह है कि यदि System इन चुनी हुई extreme values के साथ सही कार्य करता है तो बाकी सभी values के साथ भी सही कार्य करेगा।

Example :

यदि हम User से Month के लिए एक Integer enter कराना चाहते है तो इस case में possible boundaries होगी–

1---- – 2, – 1, 0	1, 2, 3 ---------12	13, 14-----------
invalid partition1	valid partition	invalid partition2

अब Tester हर Partition से दो Cases (Values) Select करके Test करेगा। अगर हर Value का Result partition के अनुरूप आता है तो System सही है अन्यथा नहीं। ❑

Scanned by CamScanner

प्रश्न 15– Integration testing की आवश्यकता और लक्ष्य क्या हैं ?

What are the need and goals of Integration testing ?

उत्तर–Testing एक Phase है जहां पर पहले Phases से बची सभी गलतियों का पता लगाया जाता है। Testing, सॉफ्टवेयर की विश्वसनीयता को सुनिश्चित करने के लिये तथा Quality assurance के लिये महत्त्वपूर्ण role को पूरा करती है। Integration testing की जरूरत व्यक्तिगत प्रोग्राम घटकों को व्यक्तिगत रूप में टेस्ट करके आपस में इंटीग्रेट करने के लिये होती है। Integration testing में एक बार व्यक्तिगत प्रोग्राम घटकों की टेस्टिंग हो जाने के पश्चात् इन्हें आंशिक या पूर्ण सिस्टम को क्रिएट करने के लिये इंटीग्रेट करना चाहिये। इस इंटीग्रेशन प्रोसेस में सिस्टम को बनाना भी सम्मिलित होता है। Integration test, system specification से डेवलप होना चाहिये तथा Integration testing उतनी जल्दी शुरू होनी चाहिये जितनी जल्दी सिस्टम कम्पोनेन्ट्स के प्रयोग किये जाने वाले संस्करण उपलब्ध होते हैं। Integration testing में top-down तथा bottom-up approaches प्रयोग की जाती हैं। पहले हम Minimal configuration को इंटीग्रेट करते हैं तब इस सिस्टम को test करते हैं। इसके पश्चात् हम इस Minimal configuration में कम्पोनेन्ट को add करते हैं तथा test करते हैं। Integration testing प्रोग्राम स्ट्रक्चर के निर्माण के लिये एक System तकनीक है।

Integration testing के प्रमुख Goals निम्नलिखित हैं–

1. Transaction Management
2. Spring IOC Container Caching
3. Dependency Injection of test fixture instances
4. Spring-specific Support classes. ❑

प्रश्न 16– सॉफ्टवेयर स्वीकृति परीक्षण से आप क्या समझते हैं ? यह कब किया जाता है ? इसके महत्व का वर्णन कीजिये।

What do you understand by Software acceptance testing ? When is it performed ? Illustrate its importance.

उत्तर–Software Acceptance Testing–सॉफ्टवेयर डेवलपमेन्ट में acceptance testing को end user testing भी कहा जाता है। Acceptance testing सॉफ्टवेयर डेवलपमेन्ट का एक फेज (Phase) है जिसमें सॉफ्टवेयर Intended audience के द्वारा real world में test किया जाता है। User acceptance testing, new systems या processes को इम्पलीमेन्ट करने के लिये Projects का एक key feature है। यह Formal mean है जिसके द्वारा हम सुनिश्चित करते हैं कि नये सिस्टम या प्रोसेस आवश्यक यूजर आवश्यकताओं से वास्तविक रूप से मिलती है। कुछ User acceptance को यहां पर बताया गया है, जो अग्रलिखित है–

1. A chance to completely test software.
2. A chance to completely test business processes.
3. A condensed version of a system.
4. A comparison of actual test results against expected results.
5. A discussion forum to evaluate the process.

Acceptance test को Project team के द्वारा attend किया जाता है। तथा System testing में अच्छा योगदान देने वाले लोगों को Identify किया जाता है। Acceptance test की प्रक्रिया को सावधानी पूर्वक मैनेज करना चाहिये तथा यह सुनिश्चित करना चाहिये कि यह निम्नलिखित उद्देश्यों के अनुसार है–

1. ट्रांजेक्शनस तथा यूजर एक्सेस के लिये वेलीडेट सिस्टम सेटअप का होना।
2. बिजनेस प्रोसेस को पूरा करने के लिये सिस्टम के प्रयोग की पुष्टि होना।
3. बिजनेस क्रिटिकल फंक्शनस के परफॉरमेन्स की संस्तुति करना।
4. कनवर्टड् तथा अतिरिक्त डाटा की इंटीग्रिटी की पुष्टि करना।

प्रोजेक्ट टीम सभी test cases की तैयारी को को-ऑरडिनेट करने के लिये जिम्मेदार होगी तथा acceptance test ग्रुप सभी test cases के क्रियान्वित होने के लिये जिम्मेदार होगा। User acceptance test ग्रुप Sample सोर्स दस्तावेजों को इनपुट के रूप में प्रयोग करके test cases को क्रियान्वित करता है तथा सुनिश्चित करता है कि tests की फाइनल outcomes संतोषजनक है।

Acceptance testing का मुख्य उद्देश्य end-users तथा stakeholders को खुश रखना है। यह वेबसाइट डिजाइन तथा डेवलपमेन्ट पर apply होती है। अधिकांशतः यह Automated सिस्टम पर इम्पलीमेन्ट होती है। Acceptance testing सुनिश्चित करती है कि नयी या सुधारी गयी वेबसाइट उसी रूप में हो जिसमें हम देखना चाहते हैं। Acceptance testing, किसी सिस्टम या सॉफ्टवेयर के सफलतापूर्वक इम्पलीमेन्टेशन के लिये आवश्यक है। ❑

Scanned by CamScanner

प्रश्न 17– परिदृश्य परीक्षण (Scenario testing) को समझाइये। इसके उद्देश्य और आवश्यकता का वर्णन कीजिये।

Explain Scenario testing. What is the purpose and need of Scenario testing ?

उत्तर–Scenario Testing–Scenario testing एक सॉफ्टवेयर टेस्टिंग एक्टिविटी है जो Scenario tests का प्रयोग करती है। यह Hypothetical story पर आधारित होती है जो एक व्यक्ति को जटिल समस्या या सिस्टम के लिये टेस्टिंग वातावरण के लिये सहायता करती है। Scenario testing के निम्नलिखित 5 key factors होते हैं–

(i) A story

(ii) That is Motivating

(iii) Credible (i.e. can happen real world)

(iv) Complex

(v) Easy to evaluate

सॉफ्टवेयर टेस्टिंग में Scenario testing का अर्थ संसार में विभिन्न लोगों के लिये विभिन्न things है।

Scenario testing बड़े Test cases को क्रिएट करने के लिये प्रयोग की जाती है। ये Tests, सिस्टम में Bugs को पता करने के लिये प्रयोग किये जाते हैं। प्रोडक्ट को सीखने तथा आवश्यकताओं को पूरा करने के लिये और आने वाली समस्याओं का पता करने के लिये इन Test का प्रयोग किया जाता है। Scenario testing कोड कवरेज को उपलब्ध नहीं कराती है तथा यह एक Black-box testing एप्रोच है जिसे डेवलपमेन्ट में प्रयोग किया जाना चाहिये।

Scenario testing, सॉफ्टवेयर की end to end फंक्शनिंग को सही तरीके से पूरा करने के लिये की जाती है। इसके माध्यम से यह सुनिश्चित किया जाता है कि सॉफ्टवेयर के business process flows सही तरीके से कार्य कर रहे हैं या नहीं। Scenario testing में testers, clients stakeholders तथा developer से Test scenarios को क्रिएट करने के लिये सहायता लेते हैं।

Scenario testing, testers के लिये यह खोजने में सहायता करती है कि कैसे सॉफ्टवेयर एक end user के हाथों में कार्य करेगा। चूंकि Scenario testing सॉफ्टवेयर के business flow को Test करती है अतः यह काफी खराबियों (defects) को ढूंढने में सहायता करती हैं जो दूसरे प्रकार की टेस्टिंग में ढूंढना आसान नहीं है।

एक Test scenario, एक स्वतंत्र Test Case या Test Cases की एक सीरीज हो सकता है जो एक दूसरे को follow करते हैं। Test scenario एक कहानी की तरह है जो किसी end user के द्वारा सॉफ्टवेयर के usage की व्याख्या करते हैं।

Scenario testing के द्वारा हम अत्यधिक महत्वपूर्ण end-to-end transactions या सॉफ्टवेयर एप्लीकेशन्स के वास्तविक प्रयोग को Figure out कर सकते हैं जो सुनिश्चित करता है कि सॉफ्टवेयर Common user scenario के लिये कार्य कर रहा है। Scenario testing, जटिल ट्रांजेक्शन्स तथा प्रोगाम की end-to-end फंक्शनिंग की study करने के लिये बहुत महत्वपूर्ण है।

जैसे–हॉस्पिटल मैनेजमेन्ट सिस्टम, Scenario testing का उदाहरण है। आजकल जब हम कुछ हॉस्पिटलस में जाते है तो वहां पर सभी Patients का रिकॉर्ड्स HMS (Hospital Management System) सॉफ्टवेयर में होते हैं। इसमें Inpatient तथा Outpatient दोनों तरह के Patients के रिकॉर्ड्स होते हैं। Inpatient वह होते हैं जो Hospital में Admit हैं तथा Outpatients वह होते हैं जो वहां से उसी दिन इलाज करा कर जा चुके हैं। यह सॉफ्टवेयर सिस्टम Outpatients तथा Inpatients records दोनों को मैनेज करता है।

अब एक Scenario लेते हैं जहां पर एक outpatient हॉस्पिटल में डॉक्टर के पास अपनी टांग के दर्द के लिये परामर्श के लिये आता है। उसकी entry, outpatient के रूप में HMS में होती है तथा उसकी सभी पहली Medical history, HMS में होती है। कुछ समय पश्चात् वह अपनी छाती में दर्द महसूस करता है तथा हॉस्पिटल में एडमिट होने की जरूरत होती है। हॉस्पिटल में उसकी entry अब outpatient से Inpatient की हो जाती है लेकिन इस बदलाव के बाद डॉक्टर को पता लगता है कि वह इस Patient की कोई Previous history को प्राप्त करने में able नहीं है। यद्यपि इस Patient की Outpatients status के Inpatient status में बदलाव होने पर Outpateints की history उपलब्ध होनी चाहिये थी। ऐसा इसलिये होता है कि HMS सिस्टम इस Scenario में असफल हो जाता है। इसलिये यदि scenario testing इस test scenario के लिये होगी तो यह bug एडवांस में पता होगी। अतः हम देख सकते हैं कि सॉफ्टवेयर टेस्टिंग में scenario testing कितनी महत्त्वपूर्ण है। ❑

प्रश्न 18– Integration testing के विभिन्न प्रकारों को संक्षिप्त में समझाइये।

Explain the different types of Integration Testing in Short.

उत्तर–Integration testing का उद्देश्य यह सुनिश्चित करता है कि दो या दो से अधिक components के Interaction से प्रोड्यूस किया गया परिणाम फंक्शनल आवश्यकताओं को संतुष्ट करता है। Integration testing में test cases, घटकों (Components) के बीच के Interface के उद्देश्य के साथ डेवलप किये जाते हैं। Integration testing के प्रमुख types अग्रलिखित हैं–

Scanned by CamScanner

1. Big bang Integration testing
2. Top Down Integration testing
3. Bottom up Integration testing
4. Hybrid Integration testing.

Integration testing को fellow programmer की तरह treat किया जाता है। Big bang Integration testing में अधिकतर modules एक complete सॉफ्टवेयर सिस्टम के रूप में आपस में जुड़े होते हैं। Bottom up Integration testing में low level modules, procedures या functions इंटीग्रेट होते हैं इसके पश्चात् इनको test किया जाता है। Top down integration testing में top modules या high level modules पहले test होते हैं इसके पश्चात् lower modules में जाते हैं। Hybrid integration testing, तीनों types की testing का मिश्रण हैं। ❑

प्रश्न 19 – ब्लैक बॉक्स टेस्टिंग की तकनीक क्या है?

What BlackBox Testing Techniques?

उत्तर–ब्लैक बॉक्स टेस्टिंग की तकनीक (Techniques of Black Box Testing)–ब्लैक बॉक्स टेस्टिंग (Balck Box Testing) निम्नलिखित तकनीकों का प्रयोग करके की जाती है—

(1) **रिग्रेशन टेस्टिंग** (Regression Testing)—इस टेस्टिंग में टेस्टर (Tester) यह जाँचता है कि एप्लीकेशन के किसी भाग में किया गया छोटा-सा भी परिवर्तन, एप्लीकेशन के अपरिवर्तित भागों को तो प्रभावित नहीं कर रहा है। यह टेस्टिंग एप्लीकेशन के पिछले संस्करणों (Previous Versions) को पुनः एग्जिक्यूट (Re-Execute) करके की जाती है।

(2) **फंक्शनल टेस्टिंग** (Funtional Testing)—इस टेस्टिंग में सॉफ्टवेयर को फंक्शनल रिक्वायरमेन्ट्स (Functional Requirements) के लिये टेस्ट किया जाता है। इस टेस्टिंग में यह जाँचा जाता है कि क्या मॉड्यूल (Module) स्पेसिफिकेशन्स (Specifications) के अनुरूप कार्य कर रहा है।

(3) **सिस्टम टेस्टिंग** (System Testing)—सिस्टम टेस्टिंग यह पुष्टि करती है कि सॉफ्टवेयर की फंक्शनल (Functional) और नॉन-फंक्शनल (Non-Functional) रिक्वायरमेन्ट्स (Requirements) पूरी हो रही हैं और यह टेस्टिंग सॉफ्टवेयर/हार्डवेयर रिक्वायरमेन्ट्स में परिभाषित सीमाओं से परे जाकर भी टेस्ट करने का इरादा रखती है। सिस्टम टेस्टिंग वास्तव में विभिन्न टेस्ट्स की एक श्रेणी (Series) है, जिसका मुख्य उद्देश्य कम्प्यूटर आधारित सिस्टम को पूर्णतः प्रयोग करना है। सॉफ्टवेयर सिस्टम के लिये आवश्यक विभिन्न सॉफ्टवेयर टैस्टिंग्स निम्नलिखित हैं—

(a) **रिकवरी टेस्टिंग** (Recovery Testing)—इस टेस्टिंग द्वारा यह जाँचा जाता है कि सॉफ्टवेयर किसी हार्डवेयर फेलियर (Hardware Failure), विनाशकारी समस्याओं (Catastrophic Problems) और किसी प्रकार के सिस्टम क्रैश (System Crash) से रिकवर होने की योग्यता कितनी तीव्र है।

(b) **वॉल्यूम टेस्टिंग** (Volume Testing)—इस टेस्टिंग में सिस्टम की चरम सीमाओं को जाँचने के लिये बड़ी मात्रा में डेटा का प्रोसेस किया जाता है।

(c) **सिक्योरिटी टेस्टिंग** (Security Testing)—अनधिकृत आन्तरिक अथवा बाह्य डेटा से सॉफ्टवेयर अपनी रक्षा कितनी अच्छी तरह कर सकता है, इसकी पुष्टि, इस टेस्टिंग द्वारा की जाती है।

(d) **यूजेबिलिटी टेस्टिंग** (Usability Testing)—इस टेस्टिंग द्वारा एप्लीकेशन के प्रयोग में सरलता को जाँचा जाता है। यदि यूजर इन्टरफेस (User Interface) एक महत्वपूर्ण मुद्दा है और विशिष्ट प्रकार के यूजर के लिये विशिष्ट आवश्यकता है, तो यूजेबिलिटी टेस्टिंग (Usability Testing) का प्रयोग किया जाता है। इस टेस्टिंग को यूजर फ्रेन्डलीनेस के लिये टेस्टिंग (Testing for User Friendliness) भी कहा जाता है।

(e) **परफॉरमेन्स टेस्टिंग** (Performance Testing)—इस टेस्टिंग द्वारा यह जाँचा जाता है, कि यूजर्स की रिक्वायरमेन्ट्स के अनुरूप सिस्टम उचित रूप से प्रदर्शन (Perform) कर रहा है या नहीं? परफॉरमेन्स टेस्ट निम्नलिखित दो प्रकार का होता है—

(i) **लोड टेस्टिंग** (Load Testing)—इस टेस्टिंग में सॉफ्टवेयर के प्रदर्शन (Performance) की जाँच करने के लिये, उसकी सीमाओं से ऊपर लोड डालकर टेस्ट किया जाता है।

(ii) **स्ट्रैस टेस्टिंग** (Stress Testing)—इस टेस्टिंग में सॉफ्टवेयर को उसकी सामान्य अपेक्षाओं (Expectations) और ऑपरेशनल क्षमता (Operational Capacity) से परे टेस्ट किया जाता है।

(f) **एक्सेप्टेन्स टेस्टिंग** (Acceptance Testing)—इस टेस्टिंग द्वारा यह पुष्टि की जाती है कि क्या सॉफ्टवेयर कस्टमर को स्वीकार्य (Acceptable) है और क्या यह निर्धारित रिक्वायरमेन्ट्स की पूर्ति कर रहा है। एक्सेप्टेन्स टेस्ट निम्नलिखित दो प्रकार का होता है—

Scanned by CamScanner

(i) **अल्फा टेस्टिंग** (Alpha Testing)—यह टेस्टिंग कस्टमर द्वारा डेवलपर के स्थान पर एक बन्द वातावरण में की जाती है।

(ii) **बीटा टेस्टिंग** (Beta Testing)—यह टेस्टिंग कस्टमर द्वारा अपने ही स्थान पर एक खुले वातावरण में की जाती है। इस प्रकार की टेस्टिंग में Developer की उपस्थिति अनिवार्य नहीं होती है।

(4) **विविध टेस्टिंग रणनीतियाँ** (Miscellaneous Testing Strategies)—विविध टेस्टिंग रणनीतियाँ (Miscellaneous Testing Strategies) निम्नलिखित हैं—

(a) **कॉन्फीगरेशन टेस्टिंग** (Configuration Testing)—इस टेस्टिंग को अनुकूलता मुद्दों को टेस्ट करने के लिये प्रयोग किया जाता है। इस टेस्टिंग में सॉफ्टवेयर और सॉफ्टवेयर के न्यूनतम (Minimum) और इष्टतम (Optimal) कॉन्फीगरेशन (Configuration) को टेस्ट किया जाता है और रिसोर्सेज (Resources) को जोड़ने (Adding) और संशोधित (Modifying) किये जाने के प्रभावों का निर्धारण किया जाता है।

(b) **सिनारियो टेस्टिंग** (Scenario Testing)—इस प्रकार की टेस्टिंग, डोमेन (Domain) अथवा Combinatonial टेस्ट डिजाइन के माध्यम से प्राप्त कृत्रिम कॉम्बीनेशन्स (Artificial Combinations) के बजाय, फंक्शन्स का अधिक यर्थाथवादी (Realistic) और सार्थक (Meaningful) कॉम्बीनेशन उपलब्ध कराती है।

(c) **सैनिटी टेस्टिंग** (Sanity Testing)—यह टेस्टिंग सिस्टम के व्यवहार (Behaviour) की जाँच करने के लिये प्रयोग की जाती है। इसे नैरो रिग्रेशन टेस्टिंग (Narrow Regression Testing) भी कहा जाता है।

(d) **एक्सप्लोरेटोरी टेस्टिंग** (Exploratory Testing)—यह टेस्टिंग सॉफ्टवेयर फीचर्स (Software Features) को एक्सप्लोर (Explore) करने के लिये की जाती है।

(e) **कॉम्पेटिबिलिटी टेस्टिंग** (Compatibility Testing)—यह टेस्टिंग निर्धारित करती है कि सॉफ्टवेयर पैकेज और हार्डवेयर के विभिन्न कॉम्बीनेशन्स के साथ समर्थित कॉन्फीगरेशन्स के अन्तर्गत सॉफ्टवेयर अपेक्षानुरूप प्रदर्शन (Perform) कर रहा है।

(f) **इन्सटालेशन टेस्टिंग** (Installation Testing)—यह टेस्टिंग उन तरीकों की पहचान करती है, जिनमें इन्सटालेशन प्रोसीजर गलत परिणाम प्रस्तुत करता है।

(g) **इन्टर-सिस्टम टेस्टिंग** (Inter-System Testing)—यह टेस्टिंग दो अथवा अधिक एप्लीकेशन सिस्टम्स के मध्य इन्टरफेस की जाँच करती है। ❑

प्रश्न 20 – बाउन्ड्री वैल्यू एनालिसिस क्या है?

What is Boundary Value Analysis?

उत्तर–बाउन्ड्री वैल्यू एनालिसिस (Boundary Value Analysis) – बाउन्ड्री वैल्यू एनालिसिस (Boundary Value Analysis–BVA) का आधारभूत विचार यह है कि इनपुट वैरिएबल्स (Input Variables) के सबसे कम (Minimum), सबसे कम से बिल्कुल ऊपर (Just Above Minimum), सामान्य (Normal), सबसे अधिक से थोड़ा कम (Just Below Maximum) और सबसे अधिक (Maximum) मान का प्रयोग करना चाहिए अर्थात् {Min, Min+ , Nom, Max–, Max}। इसे निम्नांकित Fig. में दर्शाया गया है—

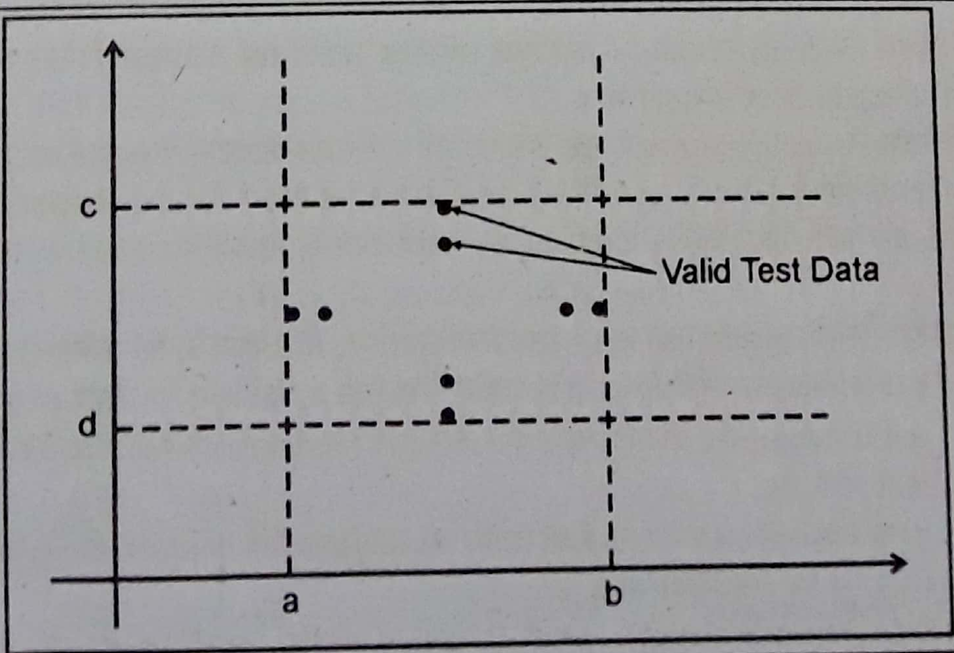

Fig. : BVA Test Cases

Scanned by CamScanner

बाउन्ड्री वैल्यू एनालिसिस (Boundary Value Analysis–BVA), एक जटिल धारणा (Assumption) पर आधारित है जिसे सिंगल फॉल्ट अजेम्पशन थ्योरी (Single Fault Assumption Theory) के नाम से जाना जाता है।

इस धारणा के अनुसार, हम इस तथ्य के आधार पर टेस्ट केसेज़ (Test Cases) बनाते हैं कि दो अथवा अधिक लगातार फॉल्ट्स (Faults) के कारण विफलता (Failure) नहीं होती। अतः हम टेस्ट केसेज़ (Test Cases) बनाते समय सभी मान लेते हैं परन्तु उनमें से एक वैरिएबल (Variable) के सांकेतिक मानों (Nominal Values) पर कार्य करता है और शेष समस्त वैरिएबल्स (Variables), प्रत्येक वैरिएबल के लिये min, min+, nom, max–, max मान को धारण किया जाता है।

अतः n वैरिएबल्स (Variables) के फंक्शन (Function) के लिये BVA को (4n + 1) टेस्ट केसेज़ (Test Cases) की आवश्यकता होती है। ❑

प्रश्न 21 – डिजीजन टेबल क्या है? इसकी लाभ-हानि भी बताइये।

What is Decision Tabel? Explain its Advantages and Disadvange.

उत्तर–डिसीजन टेबल का स्ट्रक्चर (Structure of Decision Table)–डिसीजन टेबल (Decision Table) पंक्तियों अर्थात् रो (Rows) और स्तम्भों अर्थात् कॉलम्स (Columns) से मिलकर बनी होती है, यह टेबल (Table) चार क्वाड्रेन्ट्स (Quadrants) में विभाजित होती है—

Rules ⟶

Condition	Condition Entry
Action	Action Entries

Fig. : Structure of Decision Table

डिसीजन टेबल (Decision Table) के एन्ट्री भाग (Entry Portion) में कॉलम (Column) नियम (Rule) होता है। ये नियम (Rules) यह स्पष्ट करते है कि कन्डीशन भाग (Condition Portion) में दिये गये कन्डीशनल अवस्थाओं (Conditional Circumstances) के लिये कौन-से एक्शन (Action) लिये जाने चाहिये। वे डिसीजन टेबल्स, जिसमें सभी कन्डीशन्स, बायनरी (Binary) होती हैं, लिमिटेड एन्ट्री डिसीजन टेबल्स (Limited Entry Decision Tables) कहलाती हैं। यदि कन्डीशन्स के अनेक मानों की अनुमति प्रदान की जाती है, तो परिणामतः बनने वाली टेबल्स एक्सटेन्डेड एन्ट्री डिसीजन टेबल्स (Eextended Entry Decision Tables) कहलाती हैं।

डिसीजन टेबल्स के साथ टेस्ट केसेज़ की पहचान करने के लिये निम्नलिखित steps का अनुपालन करना होता है—

Step-1 : एक मॉड्यूल के लिये इनपुट कन्डीशन्स (Input Conditions) अर्थात् कारणों (Causes) और एक्शन्स (Actions) अर्थात् प्रभावों (Effects) की पहचान की जाती है।

Step-2 : कॉज़-इफैक्ट ग्राफ (Cause-Effect Graph) को विकसित किया जाता है।

Step-3 : दूसरे चरण से प्राप्त कॉज़-इफैक्ट ग्राफ (Cause-Effect Graph) को डिसीजन टेबल में परिवर्तित किया जाता है।

Step-4 : डिसीजन टेबल नियमों को टेस्ट केस (Test Case) में परिवर्तित किया जाता है।

डिसीजन टेबल का प्रत्येक कॉलम टेस्ट केस (Test Case) को दर्शाता है अर्थात्

Number of Test Cases = Number of Rules

एक लिमिटेड एन्ट्री डिसीजन टेबल्स (Limited Entry Decision Tables) के लिये यदि n केसेज़ हैं, तो रूल्स (Rules) की संख्या 2^n होगी।

डिसीजन टेबल के लाभ (Advantages of Decision Table)

डिसीजन टेबल्स के निम्नलिखित प्रमुख लाभ हैं—

(1) इस प्रकार की टेस्टिंग पुनरावृत्तीय रूप से कार्य करती है। यह प्रारम्भिक टेबल असंतोषजनक है, तो पहली पुनरावृत्ति में तैयार की गयी टेबल नयी डिसीजन टेबल्स को व्युत्पन्न करने के लिये Guide Line के समान कार्य करती है।

(2) ये टेबल्स इसका विश्वास दिलाती हैं कि हमने कन्डीशन वैल्यूज़ (Condition Values) के हर सम्भव संयोजन (Combination) पर ध्यान दिया है। इसे कम्पलीटनेस प्रॉपर्टी (Completeness Property) के रूप में जाना जाता है। यह प्रॉपर्टी अन्य टेस्टिंग तकनीक

Scanned by CamScanner

की तुलना में कम्पलीट टेस्टिंग प्रदान करती है।

(3) डिसीजन टेबल्स घोषणात्मक (Declarative) होती हैं। इसमें कन्डीशन्स (Conditions) और एक्शन्स (Actions) के घटित होने के लिये कोई निश्चित क्रम नहीं होता है।

डिसीजन टेबल की हानियाँ (Disdvantages of Decision Table)

डिसीजन टेबल्स की निम्नलिखित प्रमुख हानियाँ हैं—

(1) टेस्टिंग की प्रक्रिया में यह एक अतिरिक्त जिम्मेदारी है।

(2) इसमें दी गयी समस्या के हल के लॉजिक द्वारा प्रवाह को नहीं दर्शाया जाता है।

(3) इसे सोर्स प्रोग्राम में अनूदित करना सरल नहीं है।

डिसीजन टेबल्स को भली-भाँति स्केल-अप (Scale-Up) नहीं किया गया है। निरर्थकता को हटाने के लिये बड़ी टेबल्स को छोटी-छोटी टेबल्स में खण्डित करने की आवश्यकता होती है। ❑

प्रश्न 22– डोमेन टेस्टिंग क्या है?

What is Domain Testing?

उत्तर–डोमेन टेस्टिंग (Domain Testing) डोमेन (Domain), इनपुट वैल्यूज़ का वह समूह (Set) होता है, जिस पर सिस्टम/एप्लीकेशन (System/Application) को परिभाषित किया जाता है।

डोमेन टेस्टिंग (Domain Testing) को ब्लैक-बॉक्स टेस्टिंग (Black Box Testing) के विस्तार (Extension) के रूप में देखा जा सकता है। यह टेस्टिंग (Testing) मुख्यतः सॉफ्टवेयर के व्यवहार (Software Behaviour) को टेस्ट (Test) करने पर केन्द्रित होती है। इस डोमेन टेस्टिंग (Domain Testing) के लिये उन टेस्टर्स (Testers) का प्रयोग किया जाता है जिन्हें विशेषतः व्यापार-क्षेत्र का ज्ञान हो। इन विशेषज्ञों का डोमेन ज्ञान (Domain Knowledge) टेस्टिंग (Testing) की लागत और समय को कम करता है और प्रोडक्ट (Product) की प्रभावकारिता बढ़ाता है। डोमेन टेस्टिंग (Domain Testing) ब्लैक-बॉक्स टेस्टिंग (Black Box Testing) की एक विधि है इसलिये इसमें प्रोग्राम अर्थात् प्रोडक्ट (Product) के लॉजिक (Logic) को नहीं बल्कि प्रोडक्ट (Product) के व्यवहार को टेस्ट (Test) किया जाता है।

डोमेन टेस्टिंग (Domain Testing) एक ऐसी टेस्टिंग तकनीक (Testing Technique) है जिसमें टेस्टिंग (Testing) डोमेन (Domain) के ज्ञान और एप्लीकेशन (Application) के डोमेन (Domain) में विशेषता (Expertize) के आधार पर की जाती है। डोमेन टेस्टिंग (Domain Testing) में सॉफ्टवेयर (Software) की टेस्टिंग (Testing) के लिये प्रोडक्ट स्पेसीफिकेशन (Product Spesification) की सहायता नहीं ली जाती है। डोमेन टेस्टिंग (Domain Testing) के लिये उस व्यापारिक गतिविधि के दैनिक कार्यों के विषय में पूर्ण ज्ञान होना आवश्यक है जिसके लिये सॉफ्टवेयर विकसित किया जा रहा है। अतः सॉफ्टवेयर स्पेसीफिकेशन (Software Specification) या सॉफ्टवेयर कोडिंग (Software Coding) के स्थान पर डोमेन (Domain) का ज्ञान होना आवश्यक है। ❑

प्रश्न 23– इंटीग्रेशन टेस्टिंग में स्टब्स एवं ड्राइवर्स क्या है?

What is stubs and Drivers of Integration Testing?

उत्तर–स्टब्स और ड्राइवर्स (Stubs and Drivers)—स्टब्स (Stubs) और ड्राइवर्स (Drivers), इन्टीग्रेशन टेस्टिंग (Integration Testing) के महत्त्वपूर्ण भाग हैं।

किसी फंक्शन (Function) को टेस्ट (Test) करने के लिये अन्य फंक्शन (Function) की भी आवश्यकता होती है। मेन प्रोग्राम (Main Program) के अतिरिक्त किसी भी अन्द फंक्शन (Function) को कॉल (Call) किया जाना अनिवार्य है अथवा इसके द्वारा किसी ऐसे मॉड्यूल (Module) को कॉल (Call) करने की आवश्यकता हो सकती है जो अभी तक विकसित न किया गया हो। यदि फंक्शन (Function) में किसी अन्य अविकसित मॉड्यूल (Module) को कॉल (Call) करने के लिये कोड (Code) सम्मिलित हो तो, ऐसे कोड (Code) को सन्तुष्ट करने के लिये कुछ दिखावटी प्रोसीजर्स अर्थात् डमी प्रोसीजर्स (Dummy Procedures) बनाये जा सकते हैं जिन्हें स्टब (Stub) कहा जाता है।

स्टब (Stub) का प्रयोग ऐसे मॉड्यूल (Module) के स्थान पर किया जाता है जो अभी तक पूर्णतः क्रियान्वित (Implement) नहीं हुआ हो और जिन्हें टेस्ट (Test) किये जा रहे कम्पोनेन्ट (Component) द्वारा कॉल (Call) किया गया हो। स्टब (Stub), अन्य मॉड्यूल (Module) के व्यवहार की नकल करता है।

उदाहरण की सहायता से स्पष्ट करने के लिये मान लेते हैं कि हमारे पास तीन मॉड्यूल्स हैं—मॉड्यूल A, मॉड्यूल B और मॉड्यूल C। मॉड्यूल A टेस्ट किये जाने के लिये तैयार है, परन्तु मॉड्यूल A, मॉड्यूल B और मॉड्यूल C के फंक्शन्स को कॉल करता है एवं मॉड्यूल B और

Scanned by CamScanner

मॉड्यूल C तैयार नहीं हैं। इस परिस्थिति में सॉफ्टवेयर डेवलपर (Software Developer), एक डमी मॉड्यूल (Dummy Module) लिखता है, जो मॉड्यूल B और मॉड्यूल C के अनुरूप होता है और मॉड्यूल A को वैल्यूज़ रिटर्न करता है। इस डमी मॉड्यूल को भी स्टब (Stub) कहा जाता है।

स्टब्स (Stubs), डमी मॉड्यूल्स (Dummy Modules) हैं, जिनको कॉल्ड प्रोग्राम्स (Called Programs) कहा जाता है और जिनका प्रयोग इन्ट्रीग्रेशन टेस्टिंग (Integration Testing) के टॉप-डाउन एप्रोच (Top Down Approach) में उस समय किया जाता है, जब उप प्रोग्राम्स (Sub Programs) बनने की स्थिति में होते हैं।

टेस्ट ड्राइवर (Test Driver) विशेष रूप से बनाया गया इन्टरफेस (Interface) है जो टेस्ट (Test) किये जा रहे सब-सिस्टम (Sub-System) के लिये टेस्ट डेटा (Test Data) भेजने में सक्षम बनाता है। टेस्ट ड्राइवर (Test Driver) का प्रयोग केवल टेस्टिंग (Testing) में किया जाता है और अन्तिम सिस्टम (Final System) में इसकी कोई भूमिका नहीं होती। ड्राइवर (Driver) एक सरल प्रोग्राम है, जिसका उद्देश्य केवल टेस्ट (Test) किये जा रहे प्रोसीज़र (Procedure) अथवा फंक्शन (Function) को कॉल (Call) करना है।

उदाहरण की सहायता से स्पष्ट करने के लिये मान लेते हैं कि हमारे पास तीन मॉड्यूल्स हैं—मॉड्यूल A, मॉड्यूल B और मॉड्यूल C। मॉड्यूल B और मॉड्यूल C टेस्ट किये जाने के लिये तैयार हैं, परन्तु मॉड्यूल A, जोकि मॉड्यूल B और मॉड्यूल C से फंक्शन्स को कॉल करता है, अभी तैयार नहीं हैं। इस परिस्थिति में सॉफ्टवेयर डेवलपर (Software Developer), एक डमी मॉड्यूल (Dummy Module) लिखता है, जो मॉड्यूल A के अनुरूप होता है और मॉड्यूल B और मॉड्यूल C को वैल्यूज़ रिटर्न करता है। इस डमी मॉड्यूल को ड्राइवर (Driver) कहा जाता है।

ड्राइवर्स (Drivers), डमी मॉड्यूल्स (Dummy Modules) हैं, जिनको कॉलिंग प्रोग्राम्स (Calling Programs) कहा जाता है और जिनका प्रयोग इन्ट्रीग्रेशन टेस्टिंग (Integration Testing) के बॉटम-अप एप्रोच (Bottom-Up Approach) में उस समय किया जाता है, जब मुख्य प्रोग्राम्स (Main Programs) अभी बनने की स्थिति में होते हैं। ❑

प्रश्न 24 – कॉलग्राफ बेस्ड इंटीग्रेशन क्या है?

What is Call graph Based Integration?

उत्तर–कॉल ग्राफ बेस्ड इन्टीग्रेशन (Call Graph Based Integration)–डिकम्पोजिशन बेस्ड इन्टीग्रेशन (Decomposition Based Integration) की एक कमी यह है कि यह फंक्शनल डिकम्पोजिशन ट्री (Functional Decomposition Tree) पर आधारित है। कॉल ग्राफ बेस्ड इन्टीग्रेशन (Call Graph Based Integration) का प्रयोग करके इस समस्या को हल किया जा सकता है। इससे हम स्ट्रक्चरल टेस्टिंग (Structrural Testing) की ओर अग्रसित होते हैं। चूँकि कॉल ग्राफ (Call Graph) एक डायरेक्टेड ग्राफ (Directed Graph) है इसलिये इसे प्रोग्राम ग्राफ (Program Graph) के रूप में भी प्रयोग किया जा सकता है। कॉल ग्राफ बेस्ड इन्टीग्रेशन (Call Graph Based Integration) को निम्नलिखित दो वर्गों में वर्गीकृत किया जा सकता है—

(1) **पेअरवाइज़ इन्टीग्रेशन** (Pairwise Integration)—पेअरवाइज़ इन्टीग्रेशन (Pairwise Integration) का मुख्य आधार स्टब/ड्राइवर (Stub/Driver) को विकसित करने में किये जाने वाले प्रयासों को हटाना है। इसमें स्ट्ब्स (Stubs) और ड्राइवर्स (Drivers) को विकसित करने के स्थान पर वास्तविक कोड (Actual Code) को प्रयोग किया जाता है। यह बिग-बैंग इन्टीग्रेशन (Big-Bang Integration) की भांति लगता है, परन्तु कॉल ग्राफ (Call Graph) में केवल एक जोड़े (Pair) की सीमा बनायी जाती है। अन्ततः कॉल ग्राफ (Call Graph) में प्रत्येक एज (Edge) के लिए एक इन्टीग्रेशन टेस्ट सेशन (Integration Test Session) बनाया जाता है। यह स्टब/ड्राइवर (Stub/Driver) के निर्माण में लगने वाला अत्यन्त समय बचाता है।

(2) **नेबरहुड इन्टीग्रेशन** (Neighbourhood Integration)—एक ग्राफ (Graph) में दी गयी नोड (Node) से एक एज (Edge) दूसरे नोड्स (Nodes) के सेट (Set) को, उस नोड (Node) का नेबरहुड (Neighbourhood) कहा जाता है। एक डायरेक्टेड ग्राफ (Directed Graph) में, यह समस्त सन्निकट पूर्ववर्ती नोड्स (Immediate Predecessor Nodes) और समस्त सन्निकट परवर्ती नोड्स (Immediate Successor Nodes) का समूह है। यह नोड के स्टब (Stub) और ड्राइवर (Driver) पर भी कार्य करता है।

किसी भी दिये गये कॉल ग्राफ (Call Graph) के लिये नेबर्स (Neighbours) की संख्या की गणना की जा सकती है। इसकी गणना करने के लिये सूत्र अग्रलिखित है—

$$\text{Interior Nodes} = \text{Nodes} - (\text{Source Nodes} + \text{Leaf Nodes}) \quad .....1$$

जहाँ इन्टीरियर नोड्स (Interior Nodes) वे नोड्स है जिसमें नॉन-जीरो इन डिग्री (Non Zero In... Degree) और नॉन-जीरो आउट डिग्री (Non Zero Out Degree) होती है।

नोड्स (Nodes), कॉल ग्राफ (Call Graph) की कुल नोड्स (Nodes) हैं।

Scanned by CamScanner

सोर्स नोड्स (Source Nodes) वे नोड्स (Nodes) हैं जिनमें जीरो इन-डिग्री (Zero In-Degree) होती हैं।
लीफ-नोड्स (Leaf Nodes) वे नोड्स (Nodes) हैं जिनमें जीरो आउट-डिग्री (Zero Out Degree) होती हैं।

Neighbourhood = Interior Nodes + Source Nodes 2

समीकरण 2 में समीकरण 1 से इन्टीरियर नोड्स (Interior Nodes) का मान विस्थापित करने पर

Neighbourhood = Nodes – Leaf Nodes

नेबरहुड इन्टीग्रेशन (Integration) इन्टीग्रेशन टेस्ट सेशन्स (Integration Sessions) की संख्या में काफी कमी ला देता है। यह स्टब (Stub) और ड्राइवर (Driver) के विकास को नजरअन्दाज करता है। ❑

प्रश्न 25 – यूजकेस सिनेरियो क्या है?

What is Use Case Scenario?

उत्तर–यूज केस सिनेरियो (Use Case Scenarior)–यूज केस सिनेरियो एक चरणबद्ध प्रक्रिया है, सिस्टम को यूजर, उसकी विभिन्न भूमिकाओं और सम्बन्धित मानदण्डों (Associated Parameters) के साथ, किस प्रकार प्रयोग करने का इरादा रखता है। एक यूज केस सिनेरियो में कथा (Stories), चित्र (Pictures) और परिनियोजन का विवरण (Deployment Details) हो सकते हैं। कस्टमर की समस्याओं और सॉफ्टवेयर, इन समस्याओं को बिना किसी संदिग्धता के किस प्रकार हल कर सकता है, की व्याख्या करने के लिये यूज केसेज़ उपयोगी होते हैं।

विभिन्न भूमिकाओं के साथ यूजर्स को यूज केस सिनेरियो में एक्टर्स (Actors) कहा जाता है। एक विशिष्ट एक्टिविटी के लिये प्रोडक्ट किस प्रकार व्यवहार करेगा, को सिस्टम बिहेवियर (System Behaviour) के नाम से जाना जाता है। विशिष्ट भूमिका वाले यूजर, जो एक्टर्स (Actors) और सिस्टम (System) के मध्य इन्टरएक्ट कराता है, को एजेन्ट (Agent) कहा जाता है।

यूज केस सिनेरियो की अवधारणा को स्पष्ट करने के लिये हम बैंक से कैश निकालने को उदाहरण के रूप में लेते हैं। इसके लिये कस्टमर एक चैक स्लिप को भरता है और उसे बैंक के कर्मचारी को देता है। बैक कर्मचारी कस्टमर के एकाउण्ट में से बैलेंस को जाँचता है और फिर कस्टमर को कैश दे देता है। इस उदाहरण में कस्टमर एक एक्टर (Actor) है, बैंक कर्मचारी एक ऐजेन्ट (Agent) है और वह कम्प्यूटर जिससे बैंक कर्मचारी कस्टमर के बैलेंस की जाँच करता है, को सिस्टम रिस्पॉन्स (System Response) कहा जाता है। ❑

प्रश्न 26 – फंक्शनल एवं नॉन फंक्शनल टेस्टिंग में क्या अन्तर है?

What isDifference between Functional and Non-functional Testing?

उत्तर–फंक्शनल और नॉन-फंक्शनल टेस्टिंग में अन्तर

(Difference between Functional and Non-Functional Testing)

फंक्शनल और नॉन-फंक्शनल टेस्टिंग में अन्तर को निम्नांकित तालिका में दर्शाया गया है—

फंक्शनल टेस्टिंग (Functional Testing)	नॉन-फंक्शनल टेस्टिंग (Non-Functional Testing)
1. इसमें उत्पाद की फंक्शनलिटी सम्मिलित होती है।	1. इसमें उत्पाद की गुणवत्ता सम्मिलित होती है।
2. इसमें उत्पाद के व्यवहार का परीक्षण किया जाता है।	2. इसमें उत्पाद के व्यवहार और अनुभव का परीक्षण किया जाता है।
3. इसमें प्राप्त एरर्स का मुख्य कारण कोड होता है।	3. इसमें प्राप्त एरर्स (Errors) का कारण आर्किटेक्चर या डिज़ाइन या कोड होता है।
4. इसको यूनिट (Unit), कम्पोनेन्ट (Component), इन्टीग्रेशन (Integration) और सिस्टम (System) टेस्टिंग (Testing) चरण में प्रयोग किया जाता है।	4. इसको सिस्टम टेस्टिंग (System Testing) चरण में प्रयोग किया जाता है।
5. इसके लिये डोमेन (Domain) की जानकारी के साथ-साथ उपभोक्ता (Customer) और प्रोडक्ट (Product) का भी ज्ञान होना आवश्यक है।	5. इसके लिये प्रोडक्ट के व्यवहार (Behaviour), डिज़ाइन (Design) और संरचना (Architecture) की समझ के साथ इस तथ्य का ज्ञान होना भी आवश्यक है कि प्रोडक्ट (Product) क्या उत्पन्न करता है।

❑

Scanned by CamScanner

प्रश्न 27– स्ट्रैस टेस्टिंग क्या है? इसके कौन-कौन से कार्य हैं?

What is stress testing? What is works.

उत्तर–स्ट्रैस टेस्टिंग (Stress Testing) अक्सर परफॉरमेन्स टेस्टिंग (Performance Testing) द्वारा अपनाई जाने वाली प्रोसेस (Process) की भांति की जाती है, परन्तु अन्तर यह है कि स्ट्रैस टेस्टिंग (Stress Testing) में सिम्यूलेटेड लोड (Simulated Load) के अत्यन्त उच्च स्तर का प्रयोग किया जाता है।

स्ट्रैस टेस्टिंग (Stress Testing) में उच्च लोड (Load) के दौरान दृढ़ता, उपलब्धता (Availability) और एरर-कन्ट्रोल (Error-Control) पर अधिक बल दिया जाता है। इसमें सामान्य स्थितियों में किये गये व्यवहार को नहीं देखा जाता। मुख्यतः इस प्रकार की टेस्टिंग का उद्देश्य यह सुनिश्चित करता है कि सॉफ्टवेयर (Software) अपर्याप्त गणनात्मक संसाधनों अर्थात् काउन्टेबल रिर्सोसेस (Countable Resources); जैसे—मेमोरी (Memory), असामान्य रूप से उच्च कॉनकरेन्सी (Concurrency) या सेवा को अस्वीकार करने (Denial of Service) आदि के दौरान क्रैश (Crash) न हो। उदाहरण के लिये, उच्च लोड (Load) वाले समय के दौरान किसी वेबसाइट (Website) के प्रदर्शन को टेस्ट (Test) करने के लिये वेब सर्वर (Web Server) की स्ट्रैस टेस्टिंग (Stress Testing) स्क्रिप्ट (Script), बॉट्स (Bots) और विभिन्न डिनायल ऑफ सर्विस (Denial of Service) आदि टूल्स (Tools) द्वारा की जा सकती है।

स्ट्रैस टेस्ट केस (Stress Test Case) का निर्माण असामान्य स्थितियों से प्रोग्राम (Program) को रन (Run) करने के लिये किया जाता है, जिसके लिये टेस्टर (Tester) सर्वप्रथम यह ज्ञात करता है कि प्रोडक्ट (Product) को विफलता से पूर्व कितने उच्च स्तर तक ले जाया जा सकता है।

स्ट्रैस टेस्टिंग (Stress Testing) एक सिस्टम को ऐसी विधि से इम्पलीमेन्ट (Implement) करती है जिसमें रिर्सोसेज़ (Resources) की असामान्य मात्रा एवं डेन्सिटी (Density) की आवश्यकता हो। स्ट्रैस टेस्टिंग (Stress Testing) के दौरान निम्न प्रकार के टेस्ट्स (Tests) किये जा सकते हैं—

(1) यदि प्रति सेकेण्ड उत्पन्न होने वाले इन्टरप्ट्स (Interupts) की औसत दर एक अथवा दो है, तो प्रति सेकेण्ड, 10 इन्टरप्ट्स (Interupts) उत्पन्न करने के लिये विशेष टेस्ट (Test) बनाए जा सकते हैं।

(2) इनपुट फंक्शन्स (Input Functions) की प्रतिक्रिया का निर्धारण करने के लिये डेटा दर को इनपुट डेटा के महत्व के क्रम द्वारा बढ़ाया जा सकता है।

(3) वे टेस्ट केस (Test Case) इम्प्लीमेन्ट (Implement) किये जाते हैं, जिनमें अन्य रिर्सोसेज़ (Resouces) या अधिकतम मेमोरी (Memory) की आवश्यकता हो।

(4) ऐसे टेस्ट केस (Test Case) बनाये जाते हैं, जिनसे ऑपरेटिंग सिस्टम (Operating System) पर अत्यधिक लोड (Load) बढ़ जाये।

(5) ऐसे टेस्ट केस (Test Case) बनाये जाते हैं, जिनसे डिस्क (Disk) में सुरक्षित डेटा को प्राप्त करने के लिये अत्यधिक खोजने की आवश्यकता हो। ❑

प्रश्न 28 – लोड टेस्टिंग क्या है? समझाइये।

What is Load Testing? Explain.

उत्तर–लोड टेस्टिंग (Load Testing)–लोड टेस्टिंग (Load Testing) परफॉरमन्स टेस्टिंग (Performance Testing) के प्रयासों के लिये सॉफ्टवेयर इण्डस्ट्री (Software Industry) में सर्वाधिक प्रयोग की जाती है। यहाँ लोड (Load) का तात्पर्य यूज़र्स (Users) की संख्या अथवा सिस्टम का ट्रैफिक (Traffic) है।

लोड टेस्टिंग (Load Testing) को इस तथ्य के निर्धारण के लिये किये जाने वाले टेस्ट (Test) के रूप में परिभाषित किया जाता है कि सिस्टम यूज़र्स (Users) की पूर्वानुमानित संख्या के साथ कार्य कर पाने में सक्षम है या नही।

लोड टेस्टिंग (Load Testing) किसी एप्लीकेशन (Application) को इम्प्लीमेन्ट (Implement) करने से पूर्व सिस्टम के व्यवहार का पूर्वानुमान करने और एप्लीकेशन (Application) और इसके आर्किटेक्चर (Architecture) में एरर्स (Errors) को दर्शाने/सुधारने के लिये इसे वास्तविक स्थितियों तथा अपेक्षित प्रयोग स्थितियों में सुव्यवस्थित रूप से उजागर करने की विधि है। लोड टेस्टिंग (Load Testing) का प्रयोग किसी एप्लीकेशन (Application) के कार्य की क्वालिटी (Quality) के निम्नलिखित तीन पहलुओं के एनालिसिस (Analysis) के लिये किया जाता है—

(1) प्रदर्शन (प्रतिक्रिया-समय) अथवा परफॉरमेन्स (रिस्पॉन्स-टाइम) (Performance(Response-Time)),

(2) स्केलेबिलिटी (थ्रू-आउट) (Scalability (Through-Out))

(3) विश्वसनीयता (उपलब्ध और कार्यात्मक सम्पूर्णता) अर्थात् रिलाएबिलिटी (एवेलेबिलिटी एण्ड फंक्शनल कम्प्लीटनेस) (Reliability (Availibility & Functional Completeness)। ❑

Scanned by CamScanner

प्रश्न 29 – ऑटोमेटेड टेस्टिंग के लाभ एवं हानियां बताओ?

Explain advantages and disadvantages of Automated Testing?

उत्तर–ऑटोमेटेड टेस्टिंग के लाभ (Advantages of Automated Testing)–ऑटोमेटेड टेस्टिंग (Automated Testing), मैनुअल टेस्टिंग प्रोसेस (Manual Testing Process) को स्वचालित अर्थात् ऑटोमेटिक (Automatic) करने की प्रक्रिया है। ऑटोमेशन (Automation) के लाभों में वर्धित सॉफ्टवेयर गुणवत्ता (Increased Software Quality), बाजार के लिये बेहतर समय (Improved Time to Market), दोहराने योग्य टेस्ट प्रोसीजर (Repeatable Test Procedure) और कम की गयी टेस्टिंग लागत (Reduced Testing Cost) सम्मिलित हैं। ऑटोमेशन (Automation) के मुख्य लाभ निम्नलिखित हैं–

(1) टेस्ट केसेज़ (Test Cases) का ऑटोमेटेड एग्जिक्यूशन (Automated Execution) मैनुअल एग्जिक्यूशन (Manual Execution) से अधिक तीव्र है। यह समय की बचत करता है।

(2) ऑटोमेटेड टेस्टिंग (Automated Testing) के कारण टेस्ट इन्जीनियर्स (Test Engineers) नीरस (Mundane) कार्यों से मुक्त होकर, कुछ सृजनात्मक (Creative) कार्यों पर अपना ध्यान केन्द्रित कर सकते हैं।

(3) ऑटोमेटेड टेस्ट्स (Automated Tests) अधिक विश्वसनीय हो सकते हैं। मैनुअल टेस्टिंग (Manual Testing) में गलतियाँ होने के अनेक कारण हो सकते हैं, जिनमें से कोई भी ऑटोमेटेड् टेस्टिंग (Automated Testing) को प्रभावित नहीं करता है।

(4) ऑटोमेशन (Automation), तात्कालिक टेस्टिंग (Immediate Testing) में सहायता करता है, क्योंकि इसमें टेस्ट इन्जीनियर्स (Test Engineers) की उपलब्धता की प्रतीक्षा नहीं करनी पड़ती।

(5) कुछ विशिष्ट प्रकार की टैस्टिंग्स (Testings); जैसे–रिलायबिलिटी टेस्टिंग (Reliability Testing), स्ट्रैस टेस्टिंग (Stress Testing), लोड एण्ड परफॉरमेन्स टेस्टिंग (Load and Performance Testing) के टेस्ट केसेज़ (Test Cases) को ऑटोमेशन (Automation) के बिना एग्जिक्यूट (Execute) नहीं किया जा सकता।

(6) मैनुअल टेस्टिंग (Manual Testing) को करने के लिये टेस्ट इन्जीनियर्स (Test Engineers) की आवश्यकता होती है, जबकि ऑटोमेटेड टेस्टिंग (Automated Testing) को 24 × 7 के वातावरण में एग्जिक्यूट (Execute) किया जाता है। अतः ऑटोमेटेड टेस्टिंग (Automated Testing) अनवरत रात-दिन की जा सकती है।

(7) ऑटोमेटेड (Automated) होने के पश्चात्, टेस्ट्स (Tests) को एग्जिक्यूट (Execute) करने में तुलनात्मक रूप से कम संसाधनों (Resources) की आवश्यकता होती है।

(8) ऑटोमेशन (Automation) में ऐसी स्क्रिप्ट्स (Scripts) होनी चाहियें, जिनकी सहायता से टेस्ट इन्जीनियर्स (Test Engineers) को उनका ज्ञान बढ़ाने के लिये प्रशिक्षित (Train) किया जा सके।

ऑटोमेशन (Automation) में ऐसी स्क्रिप्ट्स (Scripts) होनी चाहियें, जिनकी सहायता से अधिकतम इनपुट (Input) और आउटपुट (Output) के समूह के टेस्ट डेटा (Test Data) को प्रोड्यूस (Produce) किया जा सके। इन्हें टेस्ट डेटा जेनेरेटर्स (Test Data Generators) भी कहा जाता है।

ऑटोमेटेड टेस्टिंग की हानियाँ (Disadvantages of Automated Testing)–अनेक लाभों के साथ-साथ ऑटोमेटेड टेस्टिंग (Automated Testing) की कुछ हानियाँ भी हैं। ऑटोमेटेड टेस्टिंग (Automated Testing) की प्रमुख हानियाँ निम्नलिखित हैं–

(1) एक सामान्य ऑटोमेटेड टेस्ट स्यूट (Average Automated Test Suite) के विकास में पूर्ण मैनुअल टेस्ट साइकल (Manual Test Cycle) से 3 से 5 गुना अधिक लागत (Cost) लगती है।

(2) ऑटोमेटेशन (Automation) में कौन क्या करेगा, यह निर्णय लेना अत्यन्त जटिल हो जाता है।

(3) अनेक ऑर्गेनाइजेशन्स (Organizations) में टेस्ट ऑटोमेटेशन (Test Automation) का प्रयोग नहीं किया जाता।

(4) ऑटोमेटेशन (Automation) के लिये अत्यधिक ट्रेन्ड कर्मचारियों (Trained Staff) की आवश्यकता होती है, जो प्रत्येक ऑर्गेनाइजेशन (Organization) में उपलब्ध नहीं होते। □□

Scanned by CamScanner

UNIT-III

प्रश्न 1 – सॉफ्टवेयर क्या है? इसके प्रमुख गुणों को समझाइये।

What is software ? Explain the Main features.

उत्तर–सॉफ्टवेयर (Software)–सॉफ्टवेयर, कम्प्यूटर का महत्वपूर्ण हिस्सा है। सॉफ्टवेयर के बिना कम्प्यूटर को ऑपरेट नहीं किया जा सकता है। सॉफ्टवेयर, निर्देशों का एक समूह होता है जो विशेष कार्य को पूरा करता है। सॉफ्टवेयर वह इनफॉरमेशन होती है, जिसे सिर्फ हम देख सकते हैं, छूकर महसूस नहीं कर सकते हैं, जबकि हार्डवेयर को हम छूकर महसूस कर सकते हैं, जैसे–Keyboard, Monitor इत्यादि।

जैसे हम T.V. पर न्यूज देखते हैं तो हम सिर्फ सुन सकते हैं और देख सकते हैं। हम न्यूज को छूकर नहीं कह सकते कि यह कैसी है, जबकि हम T.V. स्क्रीन को छूकर महसूस कर सकते हैं। अतः कम्प्यूटर हार्डवेयर तथा सॉफ्टवेयर से मिलकर बना होता है। अतः किसी सिस्टम से सम्बन्धित डाटा तथा इनफॉरमेशन को सॉफ्टवेयर कहते हैं।

सॉफ्टवेयर के प्रमुख गुण निम्नलिखित हैं–

1. सॉफ्टवेयर मानव की आवश्यकताओं के अनुरूप डेवलप किये जाते हैं।
2. यह भिन्न-भिन्न दृष्टिकोण के अनुरूप डिजायन होते हैं।
3. सॉफ्टवेयर को डेवलप तथा कस्टोमाइज किया जाता है लेकिन इनका निर्माण नहीं किया जा सकता है।
4. सॉफ्टवेयर को डिस्ट्रीब्यूशन के लिये बल्क (bulk) में प्रोड्यूस नहीं किया जाता है, क्योंकि सॉफ्टवेयर के Case में कस्टमरों के समूहों की विभिन्न जरूरतें होती हैं।
5. सॉफ्टवेयर नष्ट नहीं होते हैं परन्तु ये खराब हो जाते हैं।
6. सॉफ्टवेयर की प्रोडक्शन कीमत, हार्डवेयर Products के Case में Customers की संख्याओं के बढ़ने पर कम नहीं (Come down) होती है, जैसा कि निम्नांकित Fig. में चित्रित किया गया है–

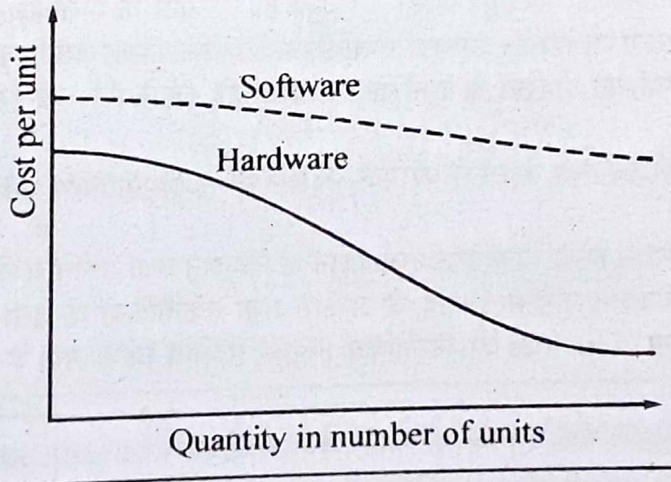

Fig. : Software cost does not come down in production

7. सॉफ्टवेयर को maintain करना costly होता है।
8. सॉफ्टवेयर को डेवलप करना one time प्रोसेस नहीं है लेकिन इसे बार-बार रन करते हैं। ❑

प्रश्न 2 – सॉफ्टवेयर के विभिन्न एप्लीकेशन कौन-कौन-से हैं?

What are the various applications of software?

उत्तर–किसी सिस्टम से सम्बन्धित डाटा तथा इनफॉरमेशन को सॉफ्टवेयर कहते हैं। किसी भी सिस्टम का हार्डवेयर, सॉफ्टवेयर के बिना अधूरा है।

डेवलपमेन्ट के प्रोडक्ट के एप्लीकेशन के nature के आधार पर सॉफ्टवेयर को निम्नलिखित प्रमुख रूपों में विभाजित किया गया है–

1. सिस्टम सॉफ्टवेयर (System Software), 2. प्रोग्रामिंग सॉफ्टवेयर (Programming Software), 3. एप्लीकेशन सॉफ्टवेयर (Application Software), 4. एडमिनिस्ट्रेटिव सॉफ्टवेयर (Administrative Software)

Scanned by CamScanner

1. सिस्टम सॉफ्टवेयर (System Software)–सिस्टम सॉफ्टवेयर एक कम्प्यूटर या कम्प्यूटर बेस्ड सिस्टम के हार्डवेयर तथा सॉफ्टवेयर के मध्य एक middle लेयर को बनाते हैं, इसलिये इन्हें Middle ware भी कहते हैं। सिस्टम सॉफ्टवेयर, सॉफ्टवेयर तथा हार्डवेयर के मध्य कम्यूनिकेशन की एक इंटरफेस को क्रिएट करते हैं। यह हार्डवेयर तथा सॉफ्टवेयर के मैसेजों को कम्यूनिकेशन के अनुरूप में Translate करते हैं। सिस्टम सॉफ्टवेयर का उद्देश्य, कम्प्यूटर के टाइप तथा इसके हार्डवेयर के एप्लीकेशन सॉफ्टवेयर को स्वतंत्र (Independent) बनाना होता है। यह हार्डवेयर की Complex details से एप्लीकेशन सॉफ्टवेयर को भी छुपाता है। ऑपरेटिंग सिस्टम्स तथा डिवाइस ड्राइवर्स, सिस्टम सॉफ्टवेयर के उदाहरण हैं।

2. प्रोग्रामिंग सॉफ्टवेयर (Programming Software)–प्रोग्रामिंग सॉफ्टवेयर, सॉफ्टवेयर डेवलपमेन्ट में प्रयोग किये जाते हैं। प्रोग्रामर जब प्रोग्रामों को लिखता है तब इन सॉफ्टवेयरों का प्रयोग किया जाता है। मॉडर्न प्रोग्रामिंग सॉफ्टवेयर, इंटीग्रेटेड डेवलपमेन्ट एनवायरनमेन्ट (IDE) के रूप में प्रकट होते हैं जो Text editor, Control editor, Interface editor, Interpreter, Compiler, Debugger, Linker, Builder तथा Viewers को Single environment, टूल या पैकेज के साथ इंटीग्रेट करते हैं। माइक्रोसॉफ्ट विजुअल स्टूडियो, प्रोग्रामिंग सॉफ्टवेयर का एक उदाहरण है।

3. एप्लीकेशन सॉफ्टवेयर (Application Software)–एप्लीकेशन सॉफ्टवेयर का प्रयोग, एबस्ट्रेक्शन के उच्च लेवल पर किया जाता है जहां पर यूजर को कम्प्यूटर सिस्टम के बारे में बहुत कम डिटेल की आवश्यकता होती है। यह सिस्टम कम्प्यूटर पर रन करता है तथा कम्प्यूटेशनल कार्य की सुविधा देते हैं। एप्लीकेशन सॉफ्टवेयर सामान्यतः कम्प्यूटर सिस्टम से अलग खरीदे जाते हैं, तथा इनके प्रयोग के लिये ऑपरेटिंग सिस्टम पर Install किये जाते हैं। एप्लीकेशन सॉफ्टवेयर को Install करने के लिये Users को कम्प्यूटर सिस्टम की अधिक जानकारी की आवश्यकता नहीं होती है। Image editors, Word editors, Video games इत्यादि एप्लीकेशन सॉफ्टवेयर के उदाहरण हैं।

4. एडमिनिस्ट्रेटिव सॉफ्टवेयर (Administrative Software)–यह एक विशेष प्रकार का एप्लीकेशन सॉफ्टवेयर होता है जो कम्प्यूटर के एडमिनिस्ट्रेशन के लिये प्रयोग किया जाता है, जिसमें हार्डवेयर कम्पोनेन्ट्स, डाटाबेस कम्पोनेन्ट्स तथा सॉफ्टवेयर कम्पोनेन्ट्स सम्मिलित होते हैं। एडमिनिस्ट्रेटिव सॉफ्टवेयर, सिस्टम के कम्पोनेन्ट्स को add या modify, access, परमीशन तथा restrictions को क्रिएट करने के लिये प्रयोग किये जाते हैं। ❑

प्रश्न 3 – सॉफ्टवेयर इंजीनियरिंग से आप क्या समझते हो?
What is software engineering?

उत्तर–सॉफ्टवेयर आज हमारे जीवन का महत्वपूर्ण हिस्सा बन चुका है। कम्प्यूटर के एप्लीकेशनों की जरूरतों को पूरा करने के लिये सॉफ्टवेयर की जरूरतें दिन प्रति-दिन बढ़ती जा रही हैं। कम्प्यूटर एप्लीकेशन के प्रत्येक फील्ड; जैसे बिजनेस, कम्यूनिकेशन, मॉनिटरिंग इत्यादि कम्प्यूटर के आविष्कार तथा इनसे सम्बन्धित प्रोडक्टों से प्रभावित हैं। सॉफ्टवेयर इंजीनियरिंग को हम निम्नलिखित रूप में परिभाषित कर सकते हैं–

"सॉफ्टवेयर के डेवलपमेन्ट, ऑपरेशन तथा मेन्टेनेन्स की एक Systematic, Disciplined, quantifiable एप्रोच की एप्लीकेशन को सॉफ्टवेयर इंजीनियरिंग कहते हैं।"

सॉफ्टवेयर इंजीनियरिंग एक प्रोफेशन है जो सॉफ्टवेयर एप्लीकेशन के क्रिएशन तथा मेन्टेनेन्स के लिये समर्पित है जो कम्प्यूटर साइंस, प्रोजेक्ट मैनेजमेन्ट, इंजीनियरिंग, एप्लीकेशन डोमेन्स इत्यादि के अभ्यास तथा तकनीकियों के द्वारा Apply किया जाता है।

सॉफ्टवेयर इंजीनियरिंग की प्रमुख Activities को निम्नांकित Fig. में प्रदर्शित किया गया है–

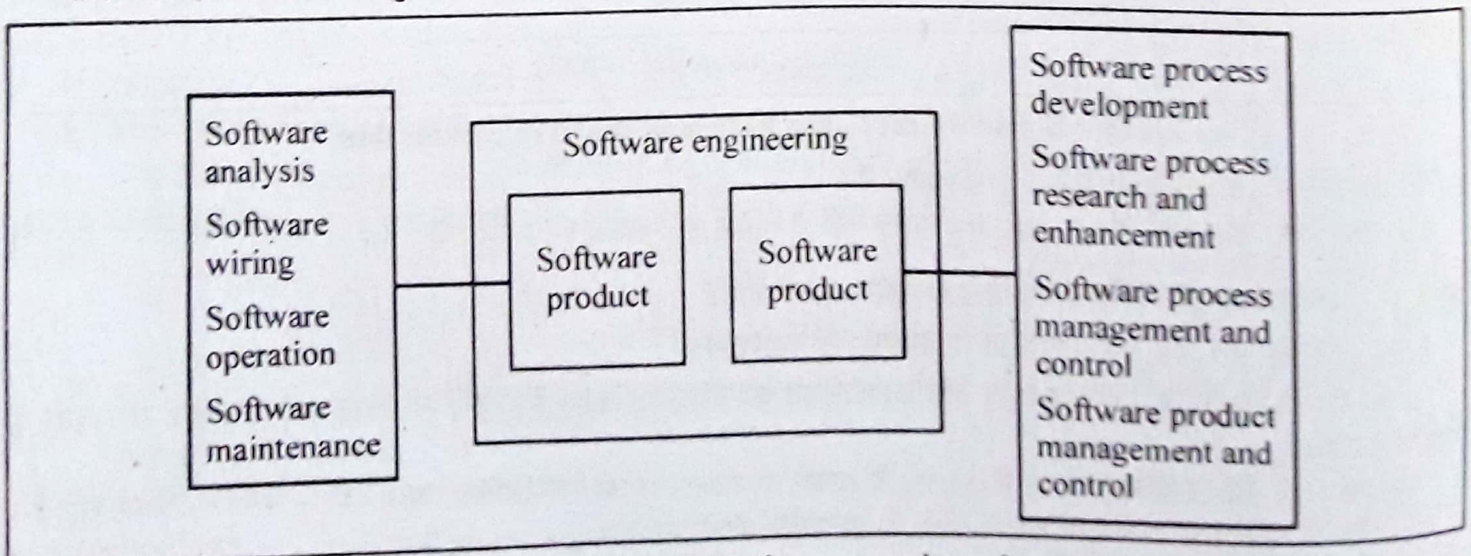

Fig. : Activities of Software engineering

Scanned by CamScanner

यह Complex सॉफ्टवेयर एप्लीकेशन्स के Cost effective production में semi-automated तथा automated टूल्स, Technically sound workforce, well defined मैथड्स, मैनेजमेन्ट नियंत्रित प्रोसेस्स के एप्लीकेशन को सपोर्ट तथा प्रमोट करता है। ❑

प्रश्न 4 – RAD मॉडल के बारे में विस्तारपूर्वक समझाइये।
Explain the RAD model in detail.

उत्तर–RAD मॉडल–मॉडर्न प्रोजेक्टों को मॉडुलर डिजाइनों पर डेवलप किया जाता है। यह विभिन्न प्रोजेक्टों में मॉड्यूलों के पुनः प्रयोग की सुविधा को प्रदान करते हैं; जैसे–Print facilitating सॉफ्टवेयर एक ही तरह के प्रिंटर इंटरफेस को विभिन्न प्रोजेक्टों की जरूरत के अनुसार प्रिंटिंग को प्रयोग करने के लिये प्रयोग किया जा सकता है।

यह उदाहरण दर्शाते हैं कि सॉफ्टवेयर डेवलपमेन्ट की मॉडुलर एप्रोच, भविष्य के प्रोजेक्ट में मॉडयूलों के प्रयोग के लिये लाभदायक है। इस तरह की एप्रोच को Rapid Application Development (RAD) में प्रयोग किया गया है। निम्नांकित Fig. में RAD मॉडल को प्रदर्शित किया गया है–

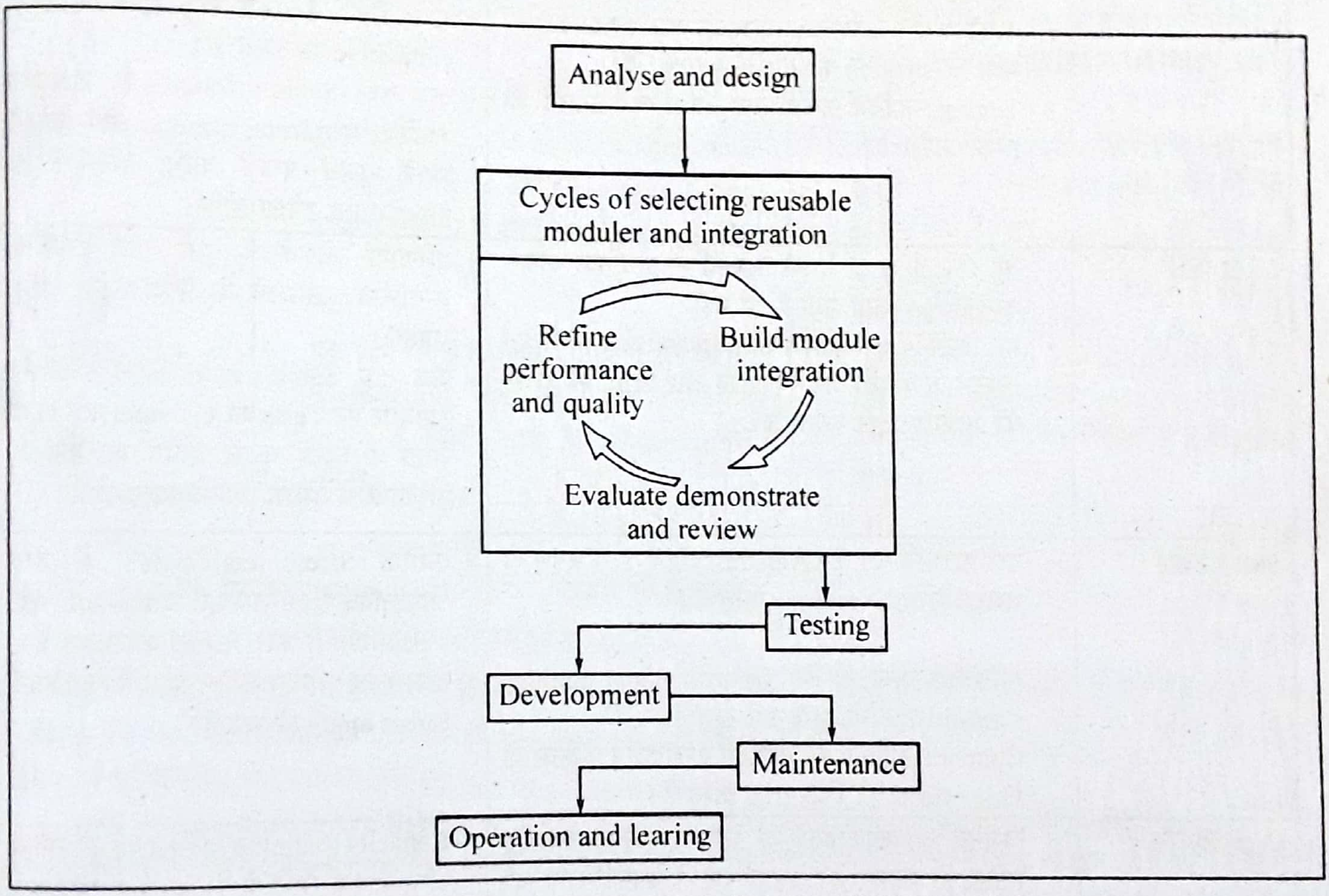

Fig. : Rapid Application Development (RAD) Model

RAD मॉडल को सॉल्यूशन की फास्ट डिलीवरी के लिये जरूरतों को सर्व करने के लिये परिभाषित किया गया है। इस मॉडल का फोकस कम से कम संभव समय में सॉल्यूशन को उपलब्ध कराता है।

RAD मॉडल, रियूज बेस्ड डेवलपमेन्ट प्रोसेस मॉड्यूल है तथा मॉडल की सफलता, मॉड्यूल कम्पोनेन्ट्स की उपलब्धता पर बहुत अधिक निर्भर करती है। बने हुये सिस्टम की विश्वसनीयता, मॉड्यूल की क्वालिटी के द्वारा मुख्यतः निर्धारित होती है। रियूज बेस्ड डेवलपमेन्ट CAD तथा CASE टूल्स के प्रयोग को उत्साहित करते हैं तथा शीघ्र परिणामों को प्राप्त करने के लिये एनालिसिस, डिजाइन, एसेम्बली, टेस्टिंग तथा इंस्टॉलेशन की स्पीड को बढ़ाने के लिये विशेष तकनीकी है।

यह मॉडल सामान्यतः लगातार तथा कस्टमर के साथ बहुत Interaction के साथ क्रियान्वित होता है।

वास्तव में डेवलपर तथा कस्टमर Parties का एक्टिव Interaction सॉल्यूशन की फास्ट डिलीवरी की चाबी (Key) है। RAD मॉडल, मॉडर्न डेवलपमेन्ट टूल्स के एक्सीलेन्ट कॉम्बिनेशन पर कार्य करता है तथा उच्च डेवलपमेन्ट योग्यता, बहुत तेज मैनेजमेन्ट कंट्रोल, समर्पित प्रोफेशन तथा कस्टमर के साथ आन्तरिक रिलेशनशिप के साथ कार्य करता है। ❑

Scanned by CamScanner

प्रश्न 5 – विभिन्न परफॉरमेन्स टेस्ट से सम्बन्धित लाभ तथा Challenges बताइये।

Explain the advantages & challenges related to various performance test.

उत्तर–विभिन्न परफॉरमेन्स टेस्ट से सम्बन्धित लाभ (Benefits) तथा Challenges को निम्नांकित Table में स्पष्ट रूप से प्रदर्शित किया गया है–

Term	Benefits	Challenges
Performance Test	• एक एप्लीकेशन के स्पीड, स्केलिबिलिटी तथा स्टेबिलिटी गुणों को निर्धारित करती है जिसके द्वारा बिजनेस निर्णय के लिये इनपुट उपलब्ध कराते हैं। • यदि सिस्टम का यूजर एप्लीकेशन के परफॉरमेन्स गुणों के साथ संतुष्ट है तो इरादों पर फोकस करता है। • परफॉरमेन्स से सम्बन्धित आशाओं तथा वास्तविकता के मध्य Mismatcfh को Identify करता है। • Tuning, प्लानिंग क्षमता तथा आशान्वित प्रयासों को सपोर्ट करता है।	• जो केवल लोड के अन्तर्गत प्रकट होते हैं उन कुछ फंक्शनल खराबी को डिटेक्ट नहीं कर सकता है। • यदि सावधानीपूर्वक डिजाइन तथा वेलीडेट नहीं किया गया तो प्रोडक्शन Scenarios के बहुत कम संख्या में परफॉरमेन्स गुणों के केवल Indicative हो सकते हैं। • जब तक Same machines से प्रोडक्शन हार्डवेयर पर Tests, Conduct नहीं होते हैं जिन्हें यूजर्स प्रयोग करेंगे, परिणामों में अनिश्चितता हमेशा होगी।
Load Test	• यह Check करता है कि Speed पर प्राथमिक फोकस को design किया गया है या नहीं? • यह किसी Application को Implement करने से पूर्व सिस्टम के व्यवहार का पुर्वानुमान और Error को दर्शाने की सुविधा प्रदान करती है।	• परिणामों को केवल दूसरे लोड टेस्टों से सम्बन्धित तुलनाओं के लिये प्रयोग करना चाहिये। • जब तक Same machines से प्रोडक्शन हार्डवेयर पर Tests को Conduct नहीं किया जाता है जिन्हें यूजर्स प्रयोग कर रहा है, परिणामों में हमेशा अनिश्चितता होगी।
Stress Test	• यदि डाटा सिस्टम के Overstressing के द्वारा भ्रमण हो सकता है तो Determine करते हैं। • सुनिश्चित करते हैं कि Security vulner abilities stressful शर्तों के द्वारा नहीं खुलती है। • Common हार्डवेयर या सपोर्टिंग एप्लीकेशन फेलियर के Side effects को डिटरमाइन करते हैं।	• क्योंकि Stress test डिजाइन के द्वारा अवास्तविक है इसलिये कुछ Stake-holoders Test परिणामों को Dismis कर सकते हैं। • अक्सर यह जानना कठिन होता है कि कितना Stress apply हो रहा है?
Capacity Test	• बिजनेस आवश्यकताओं के अनुसार कितना वर्कलोड हैण्डल हो सकता है? इसके बारे में इनफॉरमेशन को उपलब्ध कराता है। • वास्तविक डाटा को उपलब्ध कराता है जिसे Capacity Planners अपने Models या Predictions को Enhance या Validate करने के लिये प्रयोग कर सकते हैं।	• Capacity Planning Model की आकृति के अनुरूप एक समय में Testing के माध्यम से वेलीडेट नहीं कर सकते हैं जब ये आकृतियां Most वैल्यू को उपलब्ध करायेंगी। • Capacity Model में Validation Tests को क्रिएट करना simple होता है।

□

प्रश्न 6 – परफॉरमेन्स टेस्टिंग से सम्बन्धित Factors क्या हैं?

What are factors related to Performance testing?

अथवा (Or)

परफॉरमेन्स टेस्टिंग को करने के कारण बताइये।

What are the reasons to do Performance testing?

Scanned by CamScanner

उत्तर–परफॉरमेन्स टेस्टिंग (Performance Testing)–परफॉरमेन्स टेस्टिंग को स्पीड वेलीडिटी, स्केलेबिलिटी, टेस्ट के अंतर्गत प्रोडक्ट के स्टेबिलिटी लक्षणों के टैक्नीकल इनवेस्टीगेशन के रूप में परिभाषित किया जाता है। परफॉरमेन्स से सम्बन्धित गतिविधियाँ (जैसे टेस्टिंग तथा ट्यूनिंग, रेस्पांस टाईम, Thrmgput तथा resource utilization levels) जो टेस्ट के अंतर्गत एप्लीकेशन के परफॉरमेन्स उद्देश्यों से मिलती है, से सम्बन्धित है, क्योंकि परफॉरमेन्स टेस्टिंग एक सामान्य टर्म है जो इसके विभिन्न Subsets, प्रत्येक वैल्यू तथा फायदों को Cover करता है जिन्हें निम्नलिखित Table में इनके उद्देश्यों के साथ या Factors के साथ प्रदर्शित किया है–

परफॉरमेन्स टेस्टिंग के अधिकतर Common Types निम्नलिखित हैं–

Term	Purpose	Notes
Performance Test	• स्पीड, स्केलेबिलिटी, स्थायित्वता को Validate या Determine करना।	• परफॉरमेन्स टेस्ट, टेस्ट के अंतर्गत प्रोडक्ट के स्थायित्व लक्षणों, स्केलेबिलिटी, स्पीड के लिये टैक्नीकल इनवेस्टीगेशन है।
Load Test	• सामान्य तथा peak लोड कंडीशनों के अंतर्गत एप्लीकेशन Behaviour को Verify करना।	• लोड टेस्टिंग, verify करने के लिये Conduct की जाती है जो आपके इच्छित परफॉरमेन्स उद्देश्यों से तुम्हारे एप्लीकेशन का मिलान करते हैं।
Stress Test	• एक एप्लीकेशन के behaviour को Validate या Determine करना जब इसे normal या peak load कंडीशन्स के अतिरिक्त Push किया जाता है।	• Stress टेस्टिंग का उद्देश्य एप्लीकेशन bugs को छुपाना है जो केवल उच्च लोड कंडीशनों के अंतर्गत होते हैं। इन bugs में Synchronization issues, race conditions तथा मैमोरी leaks सम्मिलित हो सकते हैं।
Capacity Test	• एक दिये गये सिस्टम को कितने यूजर्स या/तथा Transactions सपोर्ट करेंगे तथा परफॉरमेन्स goals से मिलते-जुलते हैं, को Determine करना।	• Capacity testing, Capacity Planning के साथ Conjunction में Conduct होती है जिसे हम भविष्य के लिये प्लान करने के लिये प्रयोग करते हैं।

प्रश्न 7 – *Agile एवं Extreme Testing को विस्तार से लिखिये।*

Explain in detail Agile & Extreme testing.

उत्तर–Agile Testing एक सॉफ्टवेयर टेस्टिंग प्रेक्टिस है जो agile मैनिफेस्टो के सिद्धान्त को Follow करता है। Agile Testing, डिफाइन्ड टेस्टिंग प्रोसीजर्स को दृढ़ता से emphasize नहीं करता है। Agile word का अर्थ है "Moving Quickly" तथा यह Agile टेस्टिंग के पूरे कॉन्सेप्ट की व्याख्या करता है।

Agile Testing, Customer Perspective को जितनी जल्दी संभव हो उतनी जल्दी टेस्टिंग में Involve करता है। सॉफ्टवेयर का Working Increments अक्सर Agile सॉफ्टवेयर डेवलपमेन्ट में रिलीज होता है अतः अक्सर Test को करने की भी जरूरत होती है।

Agile Testing में टेस्टर्स Quality Police के लम्बे Form में नहीं होते हैं। टेस्टिंग नई रणनीति की प्रोजेक्ट फॉरवर्ड में मूव करते हैं। इसे Test Driven Development कहते हैं। टेस्टर इनफॉरमेशन, फीडबैक तथा सुझावों को उपलब्ध कराते हैं। जैसा कि हम जानते हैं कि कोडिंग के शुरू होने से पहले यूनिट टेस्ट का क्रिएशन XP एप्रोच का एक Key Element है। यूनिट टेस्ट, एक फ्रेमवर्क का प्रयोग करके इम्प्लीमेन्ट होना चाहिये जो उन्हें आटोमेशन में सक्षम बनाता है। यह Regression टेस्टिंग रणनीति को उत्साहित करता है जब भी कोड संशोधित होता है।

हालांकि पहले का कार्य के Ideas तथा मैथड्स Extreme Programming (XP) से सम्बन्धित होता है। Extreme Programmins (XP) ऑब्जेक्ट ऑरियेन्टेड एप्रोच को प्रयोग करता है। XP, Rules तथा Practices के एक set को उपलब्ध कराता है, जो चार फ्रेमवर्क गतिविधियों में होता है–

Scanned by CamScanner

1. Planning
2. Design
3. Coding
4. Testing

1. Planning–Planning गतिविधि Stories को एक सेट के साथ शुरू होती है जो सॉफ्टवेयर को बनाने के लिये आवश्यक फीचर्स तथा Functionality का वर्णन करती है। प्रत्येक Story को Customer के द्वारा लिखा जाता है तथा Index Card में स्थापित किया जाता है।

2. Deasgin–XP, KIS (Keep it simple) सिद्धान्त को Follow करती है। एक साधारण डिजाइन हमेशा Prefer की जाती है।

3. Coding–Stories के पश्चात् XP डेवलपमेन्ट तथा Preleminary डिजाइन को Recommend करती है।

4. Testing–Testing Process, XP में डिजाइन किये गये सॉफ्टवेयर को टेस्ट करती है।

निम्नांकित Fig. में Extreme Programming Process को चित्रित किया गया है–

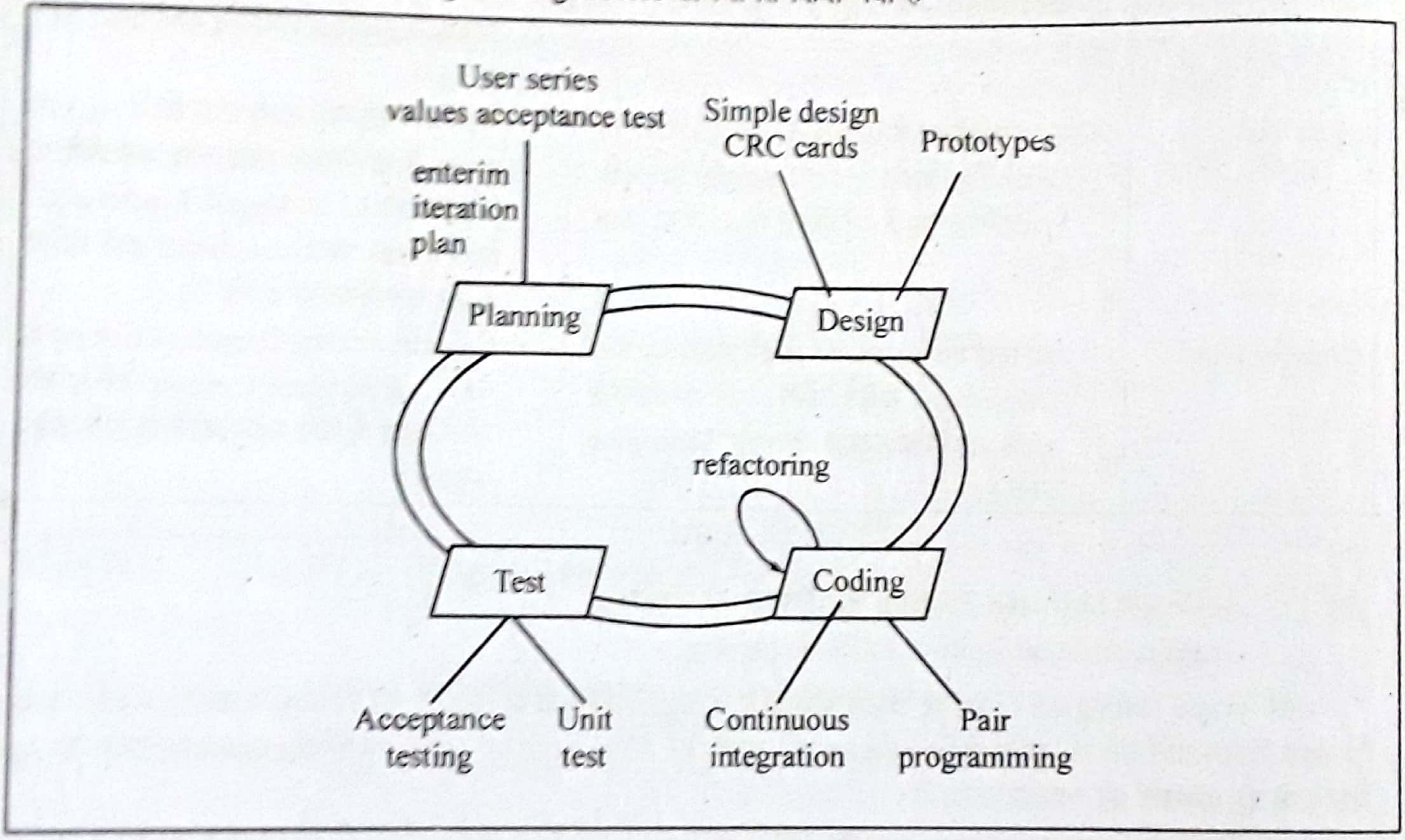

Fig. : Extreme Programming Process

प्रश्न 8 – ***Exploratory Testing से आप क्या समझते हो? Interacting Testing को समझाइये।***

What is Exploratory Testing ? Explain the Interacting testing.

उत्तर–Exploratory Testing–Exploratory Testing सॉफ्टवेयर टेस्टिंग की एक एप्रोच है जिसका टेस्ट डिजाइन तथा टेस्ट क्रियान्वयन के रूप का वर्णन किया जाता है। Exploratory टेस्टिंग को संक्षिप्त रूप में इस तरह परिभाषित कर सकते हैं–

"A style of software Testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his/her work by treating test-related learning, test design, test execution and test result interpretation as mutually supportive activities that run in Parallel through put the project."

जब सॉफ्टवेयर की टेस्टिंग होती है, टेस्टर अनुभव तथा क्रिएटिविटी दोनों के साथ रन करने के लिये नये अच्छे Tests को जनरेट करता है।

Exploratory Testing हमेशा skilled testers के द्वारा परफॉर्म की जातीं है। Exploratory testing यह पता करती है कि कैसे

Scanned by CamScanner

सॉफ्टवेयर क्वालिटी कार्य करती है तथा इसे कैसे आसानी से handle कर सकते हैं। टेस्टिंग की क्वालिटी टेस्टर के टेस्ट केस के आविष्कारों के Skill पर निर्भर करती है। टेस्टर प्रोडक्ट तथा विभिन्न test methods के बारे में जानता है तथा अच्छी टेस्टिंग की कोशिश करेगा।

जब हम एक यूनिट टेस्ट को लिखते हैं तो हमारे एजर्ट क्या होते हैं? अधिकतर समय हम State based टेस्टिंग कर रहे होते हैं अर्थात् हमारे यूनिट Test या तो मेथड कॉल से एक रिटर्न वैल्यू को वेलिडेट करते हैं या टेस्टेड क्लास की प्रॉपर्टी में Change करते हैं।

Interacting Testing–जब Tests इतने कठिन होते हैं कि इन्हें लिखना या प्लान करना काफी Laborious होता है तो इस समस्या से निकलने के लिये Interaction-based टेस्टिंग का प्रयोग करते हैं। Interaction-based Testing क्लास तथा इसकी Dependencies के मध्य आशान्वित Interaction को Verify करते हैं। यूनिट टेस्ट यह Verify करेगा कि आशान्वित Interaction मेथड कॉल्स को स्थापित करेगा। ❑

प्रश्न 9 – Object Oriented Testing पर टिप्पणी लिखिये।

Describe object oriented testing.

अथवा (Or)

ऑब्जेक्ट ओरिएन्टेड तथा पारम्परिक टेस्टिंग में क्या अन्तर है?

Differentiate between object oriented & traditional testing?

उत्तर–ऑब्जेक्ट ओरियेन्टेड सॉफ्टवेयर या आर्किटेक्चर, लेयर्ड उप-सिस्टम की एक सीरीज में रिजल्ट देता है जो Collaborate classes को Encapsulate करता है। इन सिस्टम एलीमेन्ट्स में से प्रत्येक फंक्शन्स को परफॉर्म करता है जो सिस्टम आवश्यकताओं को एचीव करने में सहायता करता हैं। विभिन्न Levels की Variety पर ऑब्जेक्ट ओरियेन्ट सिस्टम को टेस्ट करना आवश्यक होता है।

ऑब्जेक्ट ऑरियेन्टेड टेस्टिंग, पारम्परिक सिस्टम्स की टेस्टिंग की तरह ही होती है, लेकिन यह थोड़ी-सी भिन्न है, क्योंकि ऑब्जेक्ट ओरियेन्टेड एनालेसिस तथा डिजाइन मॉड्यूल्स, स्ट्रक्चर तथा कन्टेन्ट में Similar है। टेस्टिंग इन Models के review के साथ प्रारम्भ होती है। एक बार कोड जनरेट हो जाता है, वास्तविक ऑब्जेक्ट ऑरियेन्टेड टेस्टिंग "in the small" की सिरीज में शुरू हो जाती है। पारम्परिक test case डिजाइन सॉफ्टवेयर या व्यक्तिगत मॉड्यूलों के एल्गोरिथम्स के Input-Process-Output view के द्वारा derive किये जाते हैं। ऑब्जेक्ट ओरियेन्टेड टेस्टिंग एक क्लास के states अभ्यास के ऑपरेशन्स के अनुरूप परिणामों के डिजाइन पर फोकस करती है।

एक क्लास को एनालेसिस तथा डिजाइन मॉडल के माध्यम से वर्णित करते हैं। यह Test case design के लिये टारगेट होती है। क्योंकि Attributes तथा ऑपरेशन्स Encapsulate होते हैं। क्लास के बाहर के टेस्टिंग ऑपरेशन सामान्यतः Unproductive होते हैं। यद्यपि Encapsulation, ऑब्जेक्ट ऑरियेन्टेड के लिये आवश्यक डिजाइन कॉन्सेप्ट है। यह टेस्टिंग के समय एक Minor obstacle क्रिएट कर सकते हैं।

इनहेरिटेन्स भी टेस्ट केस डिजाइनर के लिये एक और चुनौती को पेश करती है। हम पहले ही यह नोट कर चुके हैं कि Usage के प्रत्येक नये Context को पुनः टेस्टिंग की आवश्यकता होती है। मल्टीपल इनहेरिटेन्स, Context की संख्या बढ़ने पर मुश्किल पेश करता है, जिसके लिये टेस्टिंग की आवश्यकता होती है।

यदि एक Super class से Instantiate की गयी Sub classes एक ही जैसे प्रॉब्लम डोमेन के साथ प्रयोग की जाती है तो Sub class को टेस्ट करते समय Super class के लिये Test case प्रयोग किये जा सकते हैं। हालांकि Sub class एक पूर्ण रूप से भिन्न Context में प्रयोग की जाती है, Super class test case थोड़ी-सी Applicability को रखेगी तथा Tests का एक नया सेट अवश्य डिजाइन करना चाहिये। ❑

प्रश्न 10 – सिस्टम टेस्टिंग के विभिन्न पहलुओं को समझाइये।

Explain Various types of System testing.

उत्तर–सिस्टम टेस्टिंग (System Testing)–सॉफ्टवेयर, दूसरे सिस्टम तत्वों; जैसे हार्डवेयर, व्यक्ति, इनफॉरमेशन से मिलकर बना होता है तथा सिस्टम इंटीग्रेशन और वेलीडेशन टेस्ट की श्रेणी को Conduct करता है। सिस्टम टेस्टिंग, विभिन्न Tests की एक श्रेणी को conduct करता है। सिस्टम टेस्टिंग, विभिन्न Tests की एक श्रेणी है जिसका प्राइमरी उद्देश्य कम्प्यूटर बेस्ड सिस्टम का पूर्ण अभ्यास है। यद्यपि प्रत्येक Test विभिन्न उद्देश्यों को रखता है। सॉफ्टवेयर बेस्ड सिस्टम्स के विभिन्न टाईप के System tests निम्नलिखित हैं–

1. रिकवरी टेस्टिंग (Recovery Testing)–बहुत सारे कम्प्यूटर बेस्ड सिस्टमों को Fault से अवश्य रिकवर करना चाहिये तथा पहले से निर्धारित समय में प्रोसेसिंग को करना चाहिये। कुछ Cases में सिस्टम को अवश्य Tolerant होना चाहिये अर्थात् प्रोसेसिंग Fault

Scanned by CamScanner

को पूरे सिस्टम फंक्शन को बन्द नहीं करना चाहिये। रिकवरी टेस्टिंग एक सिस्टम टेस्ट है जो रिकवरी के सही तरीके से पूरा होने के लिये Verify करता है।

2. Security Testing–कोई भी कम्प्यूटर बेस्ड सिस्टम जो Sensitive इनफॉरमेशन को मैनेज करता है। Security testing इस बात की पुष्टि करता है कि एक सिस्टम में प्रोटेक्शन विधि होगी जो असामान्य असुविधा से इसे बचाता है।

सिक्योरिटी टेस्टिंग के समय टेस्टर व्यक्तिगत रूप से रोल को निभाता है।

3. Stress Testing–Stress testing, एक तरीके से सिस्टम को क्रियान्वित करता है जो असामान्य मात्रा, Frequency, या वॉल्यूम में संसाधनों की मांग करता है। जैसे Special tests डिजाइन किये जा सकते हैं जो दस Interrupts प्रति सैकण्ड को जनरेट करते हैं जब एक या दो एवरेज रेट होते हैं।

Stress testing का Variation एक तकनीकी है, जो Sensitive testing कहलाती है। Sensitive testing, वैध इनपुट क्लासों के साथ uncover डाटा Combinations को चुनती है जो Improper प्रोसेसिंग का कारण हो सकता है।

4. परफॉरमेन्स टेस्टिंग (Performance Testing)–Real time तथा एम्बेडेड सिस्टमों के लिये, सॉफ्टवेयर जो आवश्यक फंक्शन को उपलब्ध कराता है लेकिन परफॉरमेन्स आवश्यकताओं की पुष्टि नहीं करता है, अस्वीकार होती है। परफॉरमेन्स टेस्टिंग, इंटीग्रेटेड सिस्टम के कॉन्टेक्स्ट के साथ सॉफ्टवेयर की रन टाईम परफॉरमेन्स को Test करता है। परफॉरमेन्स टेस्टिंग, टेस्टिंग प्रोसेस में सभी steps को रखता है।

परफॉरमेन्स टेस्टिंग अक्सर Stress testing के साथ संयुक्त रूप में होती है तथा सामान्यतः हार्डवेयर और सॉफ्टवेयर दोनों Instrumentation की आवश्यकता होती है। ❑

प्रश्न 11 – Basis Path Testing के बारे में विस्तारपूर्वक समझाइये।

Explain in detail Basis Path Testing.

उत्तर–Basis Path Testing एक White-box testing तकनीकी है जिसे सर्वप्रथम Tom McCabe ने प्रपोज किया। Basis Path मैथड् टेस्ट केस डिजाइनर के लिये प्रोसीजरल डिजाइन के लॉजिकल Complexity मापन को ड्राइव करने के लिये सक्षम बनाता है तथा इस मापन का प्रयोग Execution Paths के Basis सेट की परिभाषा के लिये एक मार्गदर्शन के रूप में करता है। इसके प्रमुख बिन्दु निम्नलिखित हैं–

1. Flow graph Notation–Basis Path मैथड के आने से पहले कंट्रोल फ्लो के रिप्रजन्टेशन के लिये एक साधारण Notation का परिचय कराया गया जिसे Flow graph या Paogram graph कहते हैं। Flow graph, Notation का प्रयोग करके लॉजिकल कंट्रोल फ्लो को चित्रित करता है, जैसा कि निम्नलिखित Fig. में प्रदर्शित किया गया है–

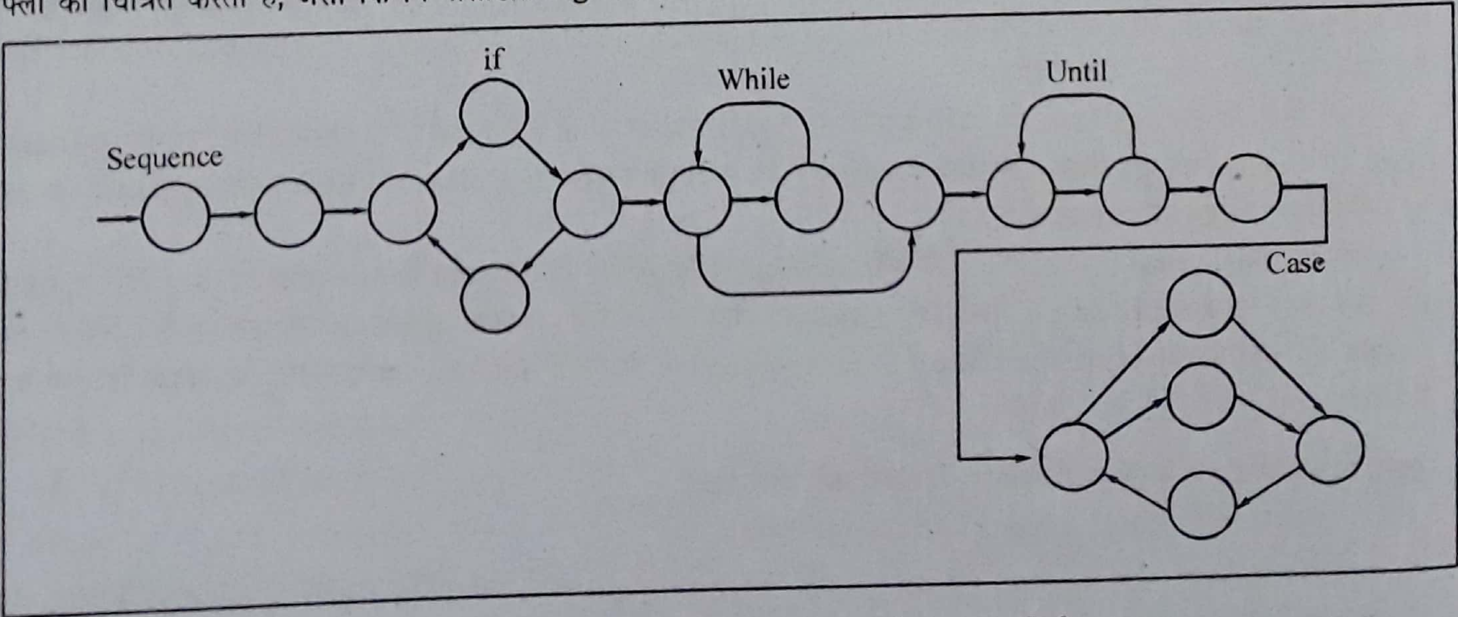

Fig. : The Structured Constructs in Flow graph form

Flow graph का प्रयोग करके हमने निम्नलिखित Fig. में प्रोसीजरल डिजाइन रिप्रजन्टेशन को विचार किया है–

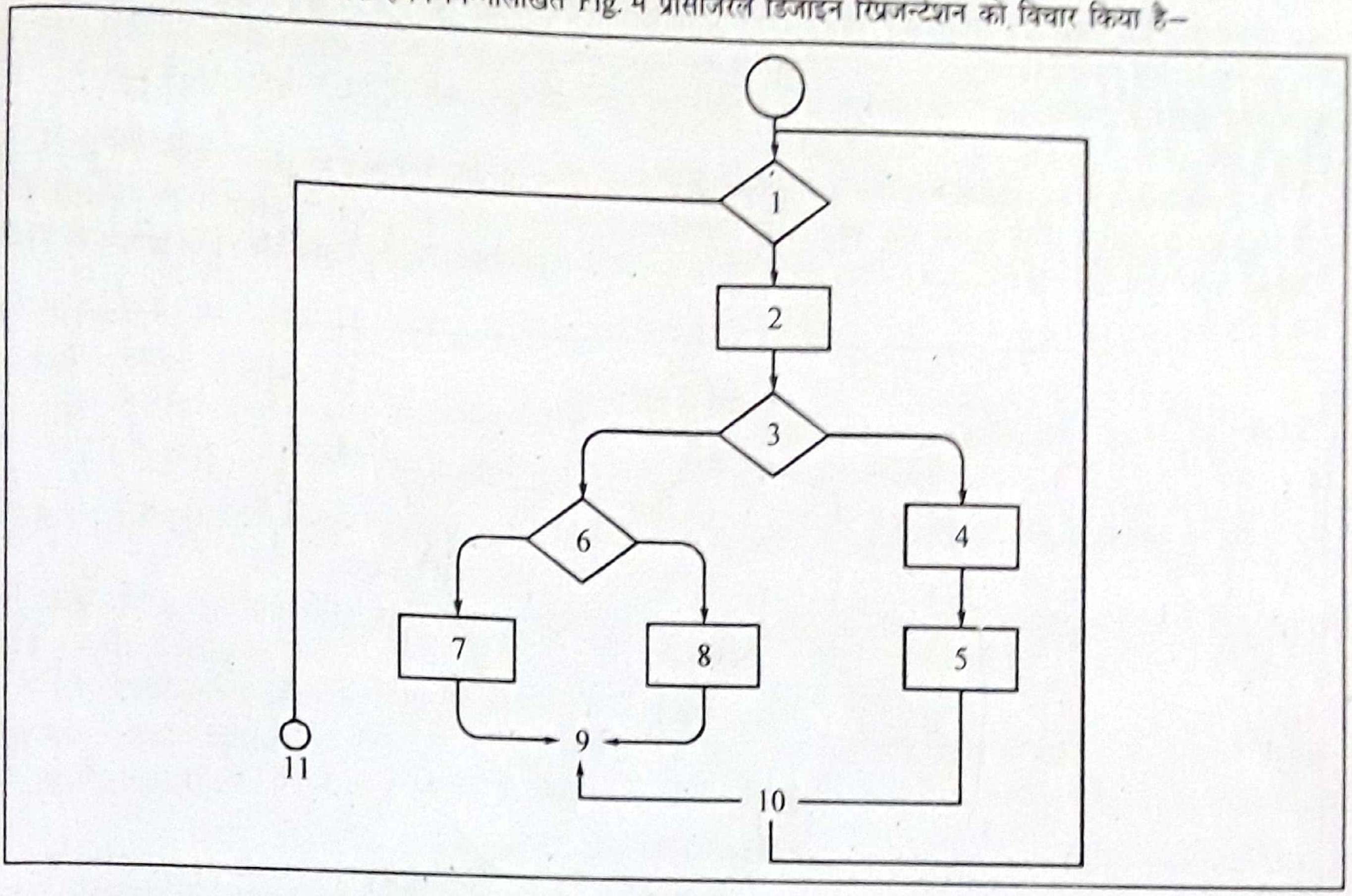

Fig. : Flow Chart

यहां पर प्रोग्राम कंट्रोल स्ट्रक्चर को चित्रित करने के लिये फ्लो चार्ट का प्रयोग किया गया है। निम्नलिखित Fig. में इस Flow Chart के Correspond Flow graph को map किया है–

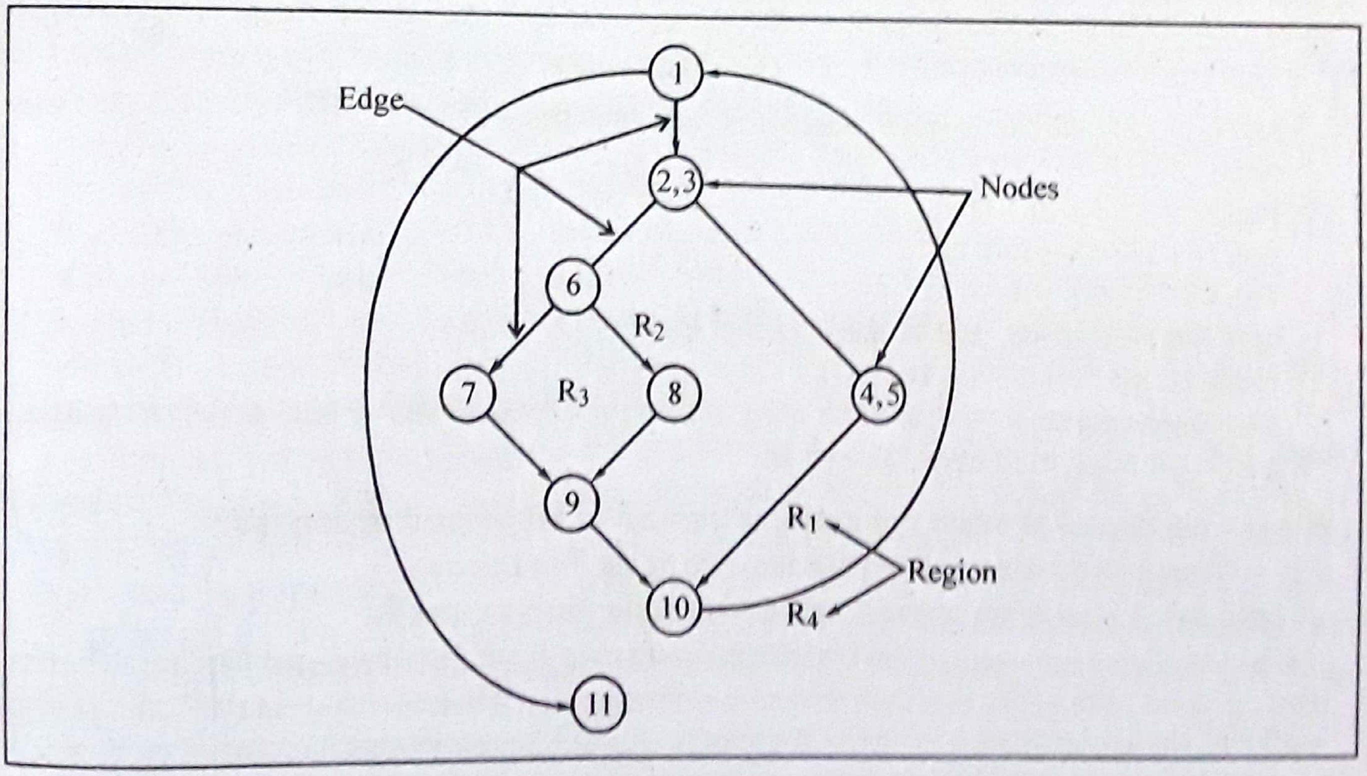

Fig. : Flow graph

Scanned by CamScanner

यहां Fig. Flow graph के correspond Flow chart को map करती है जिसमें फ्लोचार्ट के decision diamonds में Compound Conditions को नहीं रखती है। इस Fig. के अनुसार प्रत्येक Circle, Flow graph node कहलाता है जो एक या अधिक प्रोसीजरल स्टेटमेण्ट्स को रिप्रजेन्ट करता है। प्रोसेस बॉक्सों का सीक्वेन्स (Sequence) तथा decision diamond को एक Single node में map कर सकते हैं। फ्लो ग्राफ के arrows, edges या Links कहलाते हैं।

2. Independent Program Paths—एक Independent Path कोई Path होता है जिसके माध्यम से प्रोग्राम कम से कम एक नयी कंडीशन या प्रोसेसिंग स्टेटमेण्ट्स के एक नये Set को Introduce करता है। एक Flow graph के Term में एक Independent Path को कम-से-कम एक edge के along अवश्य मूव करना चाहिये; जैसे—निम्नलिखित Fig. में दर्शाये गये Flow graph को दर्शाया गया है—

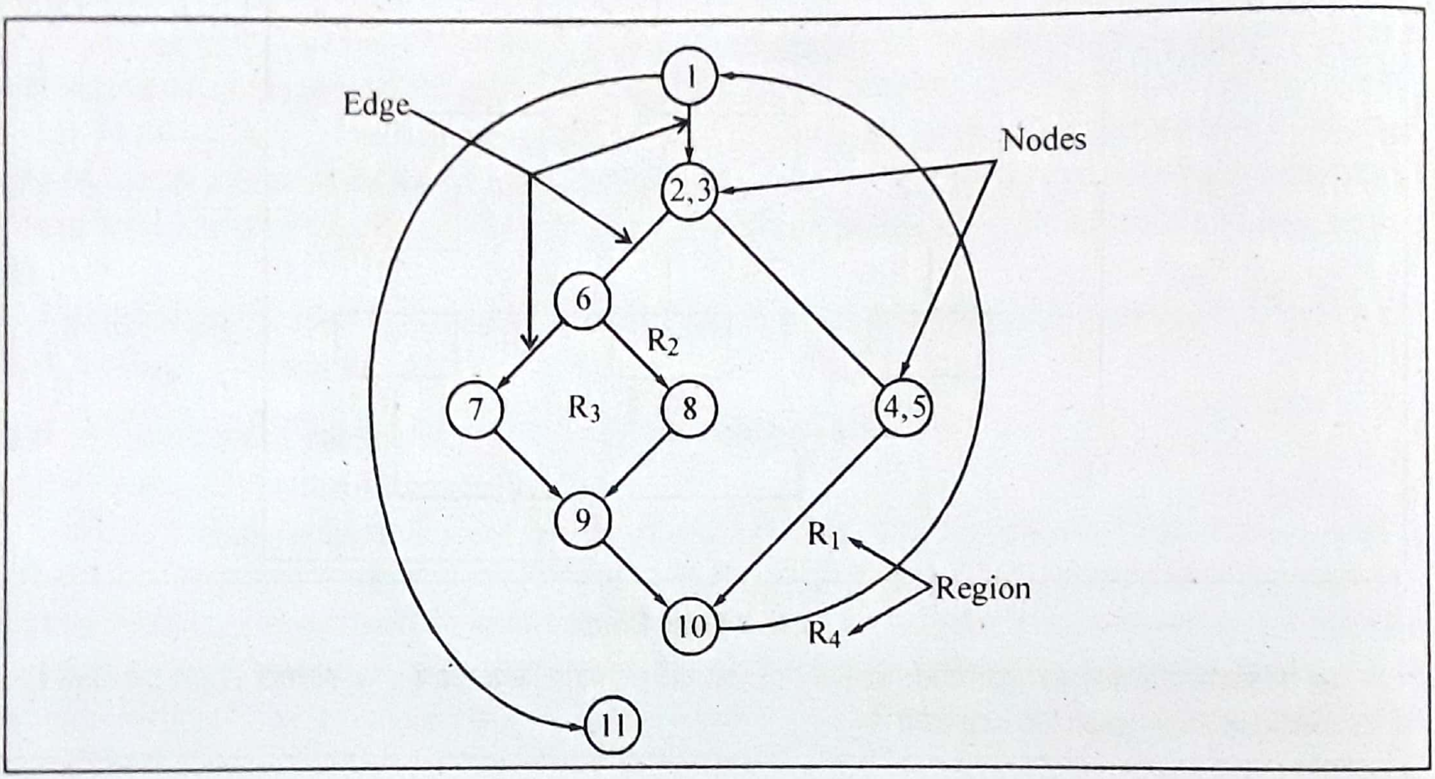

Fig. : Flow graph

इस Fig. में दर्शाये गये Flow graph के Independent Paths निम्नलिखित हैं—

Path 1 : 1 -11

Path 2 : 1-2-3-4-5-10-1-11

Path 3 : 1-2-3-6-8-9-10-1-11

Path 4 : 1-2-3-6-7-9-10-1-11

प्रत्येक नया Path एक नयी Edge को Introduce करता है—

Path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11

एक Independent Path के रूप में नहीं विचारा जायेगा, क्योंकि यह पहले से Specify किये गये Paths का साधारण Combination है तथा किसी नयी Edges को Traverse नहीं करता है। ❑

प्रश्न 12– Test Director के लक्षणों तथा उसके द्वारा प्रदान की जा रही सुविधाओं को समझाइये।

Explain the features and facilities provided by the Test Director.

उत्तर—Test director के द्वारा उपलब्ध फीचर्स तथा facilities का विवरण इस प्रकार है—

Test director, test management होता है जो सॉफ्टवेयर टेस्टिंग प्रोसेस के सभी चरणों (Phases) को मैनेज तथा ऑर्गेनाइज करने में सहायता करता है। इसमें प्लानिंग, tests क्रिएट करना, tests को क्रियान्वित करना तथा defects को check इत्यादि करना भी सम्मिलित होता है। Test director के माध्यम से हम प्रोजेक्ट के डाटाबेस के tests को maintain कर सकते हैं। प्रोजेक्ट के रूप में हम tests के test sets groups को बना सकते हैं तथा एक विशेष goal को प्राप्त करने के लिये इन्हें क्रियान्वित कर सकते हैं। जैसे—हम एक testset क्रिएट कर सकते हैं जो सॉफ्टवेयर के एक नये संस्करण को check करता है या एक specific फीचर को check कर सकता है।

Scanned by CamScanner

जैसे ही हम tests को क्रियान्वित करते हैं, Test Director हमें सॉफ्टवेयर में पाये गये defects की रिपोर्ट देता है। Defect records एक डाटाबेस में स्टोर किये जाते हैं, जहां हम इन्हें track कर सकते हैं जब तक कि ये सॉफ्टवेयर में resolve नहीं हो जाते हैं।

Test Director, Win runner के साथ कार्य करता है। Win Runner हमें ऑटोमेटेड् test scripts को क्रिएट तथा क्रियान्वित करने के लिये सक्षम बनाता है। हम Win runner ऑटोमेट्ड tests को अपने प्रोजेक्ट में सम्मिलित कर सकते हैं तथा इन्हें Test Director से सीधे क्रियान्वित कर सकते हैं।

Test Director का workflow तीन मुख्य चरणों (Phases) से मिलकर बना होता है। प्रत्येक चरण कई tasks को पूरा करता है–

(i) Planning tests

(ii) Running tests

(iii) Tracking Defects

Planning tests के अंतर्गत test को क्रियान्वित कराने की योजना आती है। Running tests के अन्तर्गत test का क्रियान्वयन होता है। Tracking Defects के अन्तर्गत test के क्रियान्वयन के समय आने वाले defects होते हैं जो एक डाटाबेस में स्टोर होते हैं। ❑

प्रश्न 13– Object Oriented Testing (OOT) की विभिन्न तकनीकों को उदाहरण के साथ समझाइए।

Explain the different Techniques of object oriented testing (OOT) with example.

उत्तर–Object oriented testing methodology टेस्टिंग तकनीको का एक कलेक्शन है जो ऑब्जैक्ट ऑरियेण्टेड सॉफ्टवेयर को Verify तथा Validate करता है।

Object oriented testing, प्रारम्भिक सिस्टमूस की टेस्टिंग की तरह ही होती है लेकिन कुछ तथ्यों पर यह भिन्न है क्योंकि Object oriented analysis तथा design मॉडल्स स्ट्रक्चर तथा कण्टेन्ट में समान है। एक बार कोड जनरेट हो जाता है, तो वास्तविक object oriented testing क्लास ऑपरेशनूस के अभ्यास के साथ शुरू हो जाती है। Object oriented testing क्लास के स्टेट पर फोकस करती है।

प्रमुख Object oriented testing methods निम्नलिखित हैं–

(i) Random Testing–इस मैथड को स्पष्ट रूप से समझने के लिये हम एक बैंकिंग एप्लीकेशन का उदाहरण लेते हैं। बैंकिंग एप्लीकेशन एकाउण्ट क्लास में निम्नलिखित ऑपरेशनूस होते हैं–Open(), Setup( ), deposit( ), withdraw( ), balance( ), Summarize( ), Credit Limit( ), तथा Close( ) इनमें से प्रत्येक ऑपरेशनूस एकाउण्ट के लिये apply किये जा सकते हैं लेकिन कुछ-कुछ निश्चित Constraints भी होते हैं जैसे–दूसरे ऑपरेशनूस को apply करने से पहले एकाउण्ट अवश्य खुला होना चाहिये तथा सभी ऑपरेशनूस को पूरा हो जाने के पश्चात् बन्द होना चाहिये। इन Constraints के साथ ऑपरेशनूस के कई permutations होते हैं। Account के एक Instance की minimum व्यवहारिक life history में निम्नलिखित ऑपरेशनूस सम्मिलित होते हैं–

Open * setup * deposit * windraw * Close यह Account के लिये minimum test sequence को रिप्रजेन्ट करता है।

भिन्न ऑपरेशन sequences की वैरायटी randomly जनरेट की जा सकती हैं जैसे–

Test case r1; open * setup * deposit * balance * summarize * withdraw * close

Test case r2 : open * setup * deposit * withdraw * deposit * balance * credit limit * withdraw * close

ये तथा दूसरे random क्रम tests, Instance life histories में विभिन्न class अभ्यास में conduct होते हैं।

अतः जब विभिन्न classes के विभिन्न ऑपरेशनूस की randomly testing की आवश्यकता होती है तब क्लासेस की random testing के लिये इस मैथड का प्रयोग करना लाभदायक होता है।

(ii) Partition Testing at the class level–पारम्परिक सॉफ्टवेयर के लिये equivalence partitioning की तरह क्लास के लिये आवश्यक Test cases की संख्या को Partition testing परिवर्तित कर देती है। इनपुट तथा आउटपुट को श्रेणी में विभाजित कर दिया जाता है तथा test cases को प्रत्येक श्रेणी की Exercise के लिये डिजाइन किया जाता है।

State-based partitioning categories क्लास ऑपरेशनूस क्लास के स्टेट को बदलने की उनकी ability पर आधारित होते हैं। इस मैथड् को समझने के लिये हम पुनः Account क्लास के बारे में विचार करते हैं। State operations में deposite( ) तथा withdraw( ) फंक्शन सम्मिलित होते हैं जबकि Nonstate ऑपरेशनूस में balance( ), summarize( ) तथा credit Limit( ) ऑपरेशनूस सम्मिलित होते हैं। इसलिये–

Test Case P1 : Open * setup * deposit * withdraw * withdraw * close

Test Case P2 : Open * setup * deposit * summarize * credit limit * withdraw * close

Test Case P1 State को change करता है जबकि Test case P2 Exercise operatings है जो state को change नहीं करता

Scanned by CamScanner

है। इस तरह जब पूरे सॉफ्टवेयर में सभी ऑपरेशन्स के बजाय class level अर्थात् सॉफ्टवेयर के कुछ Classes की partition testing करने की आवश्यकता हो तब इस method के द्वारा testing करना लाभदायक है। ❑

प्रश्न 14– प्रतिगमन परीक्षण (Regression testing) तकनीक को उपर्युक्त उदाहरणों के साथ विस्तार से समझाइये।

Explain Regression testing techniques in detail with suitable examples.

उत्तर–Regression Testing–यदि किसी कारणवश एक Software के किसी Part को Modify करना पड़े, तो हमे Testing के द्वारा ये पता लगाना चाहिए कि क्या नयी Modification सही कार्य कर रही है? क्या नयी Modifications के कारण Software पर कोई Negative प्रभाव तो नहीं पड़ा? इन सब बातों को सुनिश्चित करने के लिए जो Testing हम करते हैं उसे Regression Testing कहते है।

Regression Testing मुख्य रूप से निम्न बातों को Check करता है–

(i) Modification के बाद Software वैसे कार्य कर रहा है जैसे हम चाहते है।

(ii) Modification का Software की Original Functionality पर कोई Negative प्रभाव तो नहीं पड़ा।

(iii) नयी Modification के कारण Software में कोई नया Bug तो नहीं आ गया।

दूसरों शब्दों में हम यह भी कह सकते हैं कि यदि हम किसी भी समय एक प्रोग्राम के इम्पलीमेन्टेशन में संशोधन करना चाहते हैं तो हम ऐसा कर सकते हैं। ऐसा करने के लिये हमें Regression testing का प्रयोग करना चाहिये।

वास्तव में Regression testing, एक्सट्रीम प्रोग्रामिंग सॉफ्टवेयर डेवलपमेन्ट मैथड् का एक इंटीगरल भाग है। इस मैथड में पूरे सॉफ्टवेयर पैकेज के extensive तथा ऑटोमेटेड टेस्टिंग के द्वारा डिजाइन डॉक्यूमेण्ट्स को प्रतिस्थापित किया जाता है।

Regression testing को न केवल एक प्रोग्राम की correctness की टेस्टिंग के लिये प्रयोग किया जाता है बल्कि इसकी आउटपुट की क्वालिटी को track करने के लिये भी प्रयोग किया जाता है। Regression testing ऑटोमेट्ड हो सकती है। Regression Testing को Test plan का हिस्सा होना चाहिये।

जैसे–माना हम एक प्रोडक्ट को पूर्ण रूप से test कर चुके हैं और इसमें कोई भी error नहीं मिली है। माना प्रोडक्ट के एक area में बदलाव किया है तथा हम यह sure करना चाहते हैं कि यह अब भी सभी tests को Pass कर लेता है। जैसे कि इसने बदलाव से पहले किये हैं। बदलाव किसी defect को नहीं दर्शाता है। टेस्टिंग यह सुनिश्चित करती है कि सॉफ्टवेयर ने एक भी step को backword या regress नहीं किया है तो इस स्थिति में यह Regression testing कहलाता है।

Regression testing को मैनेज करने का प्रैक्टिकल तरीका इसे ऑटोमेट करना है।

इस तरह हम कह सकते हैं कि Regression testing, existing software के साथ डाटा का सही तरीके से कार्य करने के प्रोसेस को सुनिश्चित करना है। Regression testing प्रोडक्ट की क्वालिटी को इसकी specified आवश्यकतायें तथा कोड को maintain करने के लिये Control measure के रूप में कार्य करता है। इसका मुख्य उद्देश्य bugs को fix करना तथा यह सुनिश्चित करना है कि समस्याओं को हल कर लिया गया है। ❑

प्रश्न 15– Buddy और Pair Testing को विस्तारपूर्वक समझाइये।

Explain Buddy and Pair Testing in detail.

उत्तर–Buddy and Pair testing–Buddy testing, एक डेवलपर के pairing up के अभ्यास को रेफर करती है तथा टेस्टर buddies के रूप में कोड के टुकड़े को test करता है। इस टाईप की टेस्टिंग, यूनिट टेस्टिंग को पूरा करने के पश्चात् पहले डेवलपमेन्ट स्टेज को apply करने के लिये अच्छी होती है तथा कोड की चेकिंग की प्राथमिकता को प्रदान करती है। प्रोडक्ट डेवलपमेन्ट के इस स्टेज पर buddy टेस्टिंग white तथा black box तकनीकि की वैरायटी को ग्रहण कर सकती है।

Buddy यूनिट टेस्ट कवरेज की अनियमितता को दूर करती है तथा उन क्षेत्रों को कवर करती हैं जो unit tests में दर्शाये नहीं जा सकते है। जैसे यूनिट इंटरफेस। buddy testing का दूसरा महत्वपूर्ण मुख्य कोडिंग स्टैण्डर्ड तथा प्रेक्टिस के review, cover include को express करती है। Buddy testing का फायदेमन्द प्रभाव प्रोडक्ट विशिष्टता को प्राप्त करता है जो टेस्टर के लिये सबसे अच्छे tests के लिये सहायता करता है तथा डिजाइन में पहले किये गये बदलाव को पूर्ण करता है।

Pair testing, एक मशीन पर माड्यूल या फीचर की टेस्टिंग में सम्मिलित होती है। वास्तविक टेस्टिंग प्रोसेस के समय जब एक व्यक्ति फीचर को टेस्ट करता है तो दूसरा व्यक्ति अतिरिक्त prespectives के प्रेक्षण को नोट करता है। दो व्यक्ति, views, idea के विभिन्न sets में एक ही फीचर में सहायता करते हैं। Pair testing नये व्यक्ति के लिये, अनुभवी व्यक्ति की अपेक्षा शीघ्रता से ramp-up करने में सहायता करती है। Pair testing कई मॉड्यूल्स को एक साथ test करने में सहायता करती है। Buddy testing, समय की कमी की वजह से, कोडिंग प्रोग्राम्स के समय test engineers के लिये टेस्टिंग करने में सहायता करती है। इस टाईप की टेस्टिंग में टेस्टिंग प्रोग्रामर्स पूर्ण build को प्राप्त करने में इंतजार नहीं करते हैं। Buddy का अर्थ प्रोग्रामर्स तथा टेस्ट इंजीनियर्स का समूह है।

Pair testing में knowledge की कमी की वजह से टेस्ट इंजीनियर्स को आपस में ग्रुप बनाकर टेस्टिंग को Conduct किया जाता है। टेस्टर का प्रत्येक भाग senior tester तथा Junior tester को मिलकर बना होता है तथा ये अपनी knowledge को share करते हैं। ❑

Scanned by CamScanner

प्रश्न 16 – परफॉरमेन्स बैन्चमार्किंग किसे कहते हैं?

What is Performance Benchmarking?

उत्तर–परफॉरमेन्स बैन्चमार्किंग (Performance Benchmarking)–परफॉरमेन्स बैन्चमार्किंग (Performance Benchmarking) एक ऐसी प्रक्रिया है, जिससे प्रोडक्ट ट्रान्जेक्शन्स (Product Transactions) की परफॉरमेन्स (Performance) की उसके प्रतियोगियों से तुलना की जाती है। किन्हीं दो प्रोडक्ट्स (Products) का आर्किटेक्चर (Architecture), डिज़ाइन (Design), फंक्शनेलिटी (Functionality) और कोड (Code) एक समान नहीं हो सकते। अतः इन तथ्यों के आधार पर दो प्रोडक्ट्स (Products) की तुलना करना अत्यन्त कठिन है।

परफॉरमेन्स बैन्चमार्किंग (Performance Benchmarking) में निम्नलिखित Steps सम्मिलित होते हैं—

Step-1 : ट्रान्जैक्शन्स (Transactions)/सिनेरियोज़ (Scenarios) और टेस्ट कॉन्फिगरेशन (Test Configuration) की पहचान करना।

Step-2 : विभिन्न प्रोडक्ट्स (Products) की परफॉरमेन्स (Performance) की तुलना करना।

Step-3 : पैरामीटर्स (Parameters) को ट्यून करना।

Step-4 : परफॉरमेन्स बैन्चमार्किंग (Performance Benchmarking) के परिणामों को प्रकाशित करना।

परफॉरमेन्स बैन्चमार्किंग (Performance Benchmarking) से निम्नलिखित तीन प्रकार के परिणाम (Outcomes) प्राप्त हो सकते हैं—

(1) पॉजिटिव परिणाम (Positive Outcome)—प्रतियोगियों की अपेक्षा ट्रान्जैक्शन्स (Transactions)/सिनेरियोज़ (Scenarios) की परफॉरमेन्स (Performance) अच्छी है।

(2) न्यूट्रल परिणाम (Neutral Outcome)—प्रतियोगियों की अपेक्षा ट्रान्जैक्शन्स (Transactions) का समूह तुलनीय (Comparable) है।

(3) निगेटिव परिणाम (Negative Outcome)—प्रतियोगियों की अपेक्षा ट्रान्जैक्शन्स (Transactions)/सिनेरियोज़ (Scenarios) की परफॉरमेन्स (Performance) खराब है। ❑

प्रश्न 17 – सामान्य टेस्टिंग एवं रिग्रेशन टेस्टिंग में अन्तर समझाइये।

Differentiate between General Testing and Regression Testing?

उत्तर – सामान्य और रिग्रेशन टेस्टिंग में अन्तर **(Difference between Normal and Regression Testing)**

सामान्य टेस्टिंग (Normal Testing) और रिग्रेशन टेस्टिंग (Regression Testing) के मध्य अन्तर को निम्नलिखित तालिका में प्रदर्शित किया गया है—

सामान्य टेस्टिंग (Normal Testing)	रिग्रेशन टेस्टिंग (Regression Testing)
(1) यह प्रक्रियन्यतः SDLC के चौथे चरण में किया जाता है।	(1) इसे मेन्टिनेन्स चरण के दौरान किया जाता है।
(2) यह मूलतः सॉफ्टवेयर के वैलिडेशन और वैरिफिकेशन (Validation & Verification) की प्रक्रिया है।	(2) इसे प्रोग्राम रिवैलिडेशन (Program Revalidation) कहा जाता है।
(3) कोड (Code) को नये टेस्ट केसेज़ (Test Cases) की सहायता से टेस्ट (Test) किया जाता है।	(3) टेस्टिंग (Testing) के लिये नये और पुराने दोनों प्रकार के टेस्ट केसेज़ (Test Cases) का प्रयोग किया जाता है।
(4) इसे वास्तविक सॉफ्टवेयर पर किया जाता है।	(4) इसे मॉडीफाइड सॉफ्टवेयर पर किया जाता है।
(5) यह टेस्टिंग सस्ती है।	(5) यह टेस्टिंग महंगी है।

❑❑

Scanned by CamScanner