महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई
(स्वायत्त) (ISO 21001:2018) (ISO/IEC 27001:2013)

अभियांत्रिकी आणि तंत्रज्ञान पदविका

शिक्षण पुस्तिका
(Learning Material)

सॉफ्टवेअर टेस्टिंग
SOFTWARE TESTING

(316314)

K-Scheme

संगणक अभियांत्रिकी गट
(के-स्कीम)

मराठी-इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील सहावे सत्र पदविका)

शिक्षण पुस्तिका

(Learning Material)

सॉफ्टवेअर टेस्टिंग

SOFTWARE TESTING

(316314)

संगणक अभियांत्रिकी गट

(के-स्कीम)

K-Scheme

मराठी-इंग्रजी (द्विभाषिक) माध्यम

(अभियांत्रिकी व तंत्रज्ञानातील सहावे सत्र पदविका)

महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई
(स्वायत्त)(ISO 21001:2018) (ISO/IEC 27001:2013)

सॉफ्टवेअर टेस्टिंग
Software Testing
(316314)

मार्गदर्शक

श्री. विजय नामदेवराव कुकरे

विभागप्रमुख, संगणक अभियांत्रिकी

प्रकल्प समन्वयक

श्री. विजय नामदेवराव कुकरे

विभागप्रमुख, संगणक अभियांत्रिकी

संकलक

श्री. संतोष यादवराव दिवेकर

अधिव्याख्याता संगणक अभियांत्रिकी

सौ. हेमलता अजयकुमार शिंदे

अधिव्याख्याता संगणक अभियांत्रिकी

महाराष्ट्र राज्य तंत्र शिक्षण मंडळ.

(स्वायत्त) (ISO: २१००१:२०१८) (ISO/IEC: २७००१-२०१३)

शासकीय तंत्रनिकेतन इमारत, चौथा मजला, ४९, खेरवाडी, बांद्रा (पूर्व), मुंबई - ४०० ०५१.

दूरध्वनी क्र.: ०२२-६२५४२१०० / १५३ / १७०

email : director@msbte.com web site : www.msbte.ac.in

प्रास्ताविक

महाराष्ट्र राज्यातील पदविका स्तरावरील तंत्रशिक्षणामध्ये विद्यार्थ्यांचे रोजगार कौशल्य विकसित करून विद्यार्थ्यांचा सर्वांगीण विकास घडवून आणण्याकरिता महाराष्ट्र राज्य तंत्रशिक्षण मंडळ कटिबद्ध आहे. उद्योगधंद्यातील बदलत्या तंत्रज्ञानाशी संबंधित गरजा लक्षात घेऊन महाराष्ट्र राज्य तंत्र शिक्षण मंडळाकडून पदविका अभ्यासक्रम वेळोवेळी अद्यावत करण्यात येतो. अभियांत्रिकी पदविका अभ्यासक्रम शिकत असताना संकल्पनात्मक ज्ञान, सुसंगत संदर्भ, प्रश्न विचारणे, विश्वसनीय पुरावे, कारणमीमांसा आणि सुस्पष्ट निकष यांचा वापर करून अर्थाची उकल करण्याची, विश्लेषण व मूल्यमापन करण्याची तसेच तर्काने अनुमान काढण्याची क्षमता म्हणजेच चिकित्सक विचार विद्यार्थ्यांमध्ये अधिक दृढ होतील असा मला विश्वास आहे. जेव्हा विद्यार्थी ज्ञान मिळवण्याच्या माध्यमाशी पूर्णपणे परिचित आणि सोयीस्कर असतात, तेव्हा त्यांच्यासाठी वर्गातील चर्चेत भाग घेणे सोपे होते, संकल्पनात्मक व सैद्धांतिक बाबींचे आकलन परिपूर्ण होते, संज्ञानात्मक क्षमता सुधारते आणि त्यांचा आत्मविश्वास देखील वाढतो. या सर्व गोष्टींचा विचार करून मंडळाकडून शैक्षणिक सामुग्रीची निर्मिती करण्यात आलेली आहे. भारत देश हा खेड्यापाडयातून विकसित झालेला देश असून ग्रामीण भागातील विद्यार्थ्यांना तांत्रिक शिक्षण घेताना भाषेचा अडसर न येता तांत्रिक बाबींचा आशय समजून घेणे शक्य होईल या दृष्टिकोनातून महाराष्ट्र राज्य तंत्र शिक्षण मंडळाने पदविका स्तरावरील तांत्रिक शिक्षणाकरीता विद्यार्थ्यांना मराठी-इंग्रजी द्विभाषिक माध्यमाचा पर्याय उपलब्ध करुन दिलेला आहे.

राष्ट्रीय शैक्षणिक धोरण-२०२० प्रादेशिक भाषेतील शिक्षणास प्रोत्साहन देते, ज्यामुळे विद्यार्थ्यांना तांत्रिक अभ्यासक्रमांसाठी प्रादेशिक भाषेतुन शिक्षणाचे माध्यम निवडता येते. त्या अनुषंगाने प्रादेशिक भाषांमध्ये तांत्रिक सामग्री आणि अभ्यास सामग्रीचा विकास आणि भाषांतर करण्याची आवश्यकता आहे. या धोरणास अनुसरुन मंडळाने भागधारकांसाठी शैक्षणिक वर्ष २०२१-२२ पासून I-Scheme तसेच शैक्षणिक वर्ष २०२३-२४ पासून K-Scheme मध्ये व्दिभाषिक माध्यमाचा पर्याय प्रथम ते तृतीय वर्षाकरिता उपलब्ध करून दिलेला आहे. या पर्यायास अनुसरून मंडळाने मराठी-इंग्रजी द्विभाषिक शैक्षणिक सामग्रीही संबंधीत विद्यार्थी व अधिव्याख्यातांकरिता उपलब्ध करून दिली आहे.

पदविका स्तरावरील तंत्रशिक्षण अधिक दर्जेदार करण्यासाठी महाराष्ट्रातील अनुभवी व तज्ञ अध्यापकांनी व्यवहारिक मराठी भाषा व इंग्रजी भाषेतील तांत्रिक शब्दावली यांचा वापर करून मराठी इंग्रजी भाषेचा सुवर्णमध्य साधण्याचा प्रयत्न केलेला आहे. मंडळाच्या स्तरावर गठीत सुकाणू समितीमार्फत सदर शैक्षणिक सामुग्रीचा दर्जा, तसेच इतर बाबींची तपासणी करण्यात आलेली आहे. त्यामुळे सदर शैक्षणिक सामुग्री अधिक संपन्न झालेली असून,विद्यार्थी त्यांच्या व्यक्तिमत्त्वाचा सुसंवादी आणि सर्वांगीण विकास साधतील. परिणामतः विश्वस्तरीय मनुष्यबळाच्या गरजा पूर्ण करण्यात महाराष्ट्र राज्य अग्रेसर राहील व पर्यायाने राष्ट्रनिर्मिती करीता निश्चितच हातभार लागेल, असा मला विश्वास आहे.

अभियांत्रिकी पदविका अभ्यासक्रमातील विषयांची मराठी-इंग्रजी (द्विभाषिक) शैक्षणिक सामुग्री बनविण्यासाठी अध्यापक व सुकाणू समितीचे सदस्य यांनी दर्शविलेले समर्पण व वचनबद्धता कौतुकास पात्र आहे, या सर्वांचे मी मनःपूर्वक अभिनंदन करतो!

(डॉ. प्रमोद नाईक)
संचालक
म. रा. तंत्र शिक्षण मंडळ, मुंबई

अनुक्रमणिका

युनिट-1
सॉफ्टवेअर टेस्टिंग ॲण्ड टेस्टिंग मेथड्स
(Software Testing and Testing Methods)

विषय निष्पत्ती (Course Outcome):

CO1: विविध सॉफ्टवेअर टेस्टिंग पद्धती स्पष्ट करा.

घटक निष्पत्ती (Theory Learning Outcome):

1. दिलेल्या प्रोग्राम मधील एरर्स आणि बग्स ओळखा.
2. दिलेल्या टेस्ट ॲप्लिकेशन साठी एंट्री क्रायटेरिया आणि एक्झिट क्रायटेरिया स्पष्ट करा.
3. सॉफ्टवेअर टेस्टिंग च्या विविध मेथड्स स्पष्ट करा.

1.1 सॉफ्टवेअर टेस्टिंग (Software Testing)

सॉफ्टवेअर टेस्टिंग ही प्रक्रिया आहे ज्यामध्ये सॉफ्टवेअर प्रॉडक्ट किंवा ॲप्लिकेशन अपेक्षित उद्देशानुसार कार्य करते का हे इव्हॅल्युएट आणि व्हेरिफाय केले जाते. यामध्ये सॉफ्टवेअर एक्झिक्यूट करून डिफेक्ट्स ओळखले जातात, क्वालिटी सुनिश्चित केली जाते आणि ते स्पेसिफाईड रिक्वायरमेंट्स पूर्ण करते का याची खात्री केली जाते. सॉफ्टवेअर टेस्टिंग ही एक स्ट्रक्चर्ड प्रोसेस आहे, ज्यामध्ये ठरावीक परिस्थितींमध्ये सॉफ्टवेअर एक्झिक्यूट करून त्याचे मूल्यमापन केले जाते, जेणेकरून डिफेक्ट्स डिटेक्ट करता येतील आणि सॉफ्टवेअर स्पेसिफाईड रिक्वायरमेंट्स पूर्ण करते का हे निश्चित करता येईल. ही प्रक्रिया सॉफ्टवेअर डेव्हलपमेंट लाईफ सायकल चा अत्यंत महत्त्वाचा भाग आहे. सॉफ्टवेअर टेस्टिंगमध्ये व्हेरिफिकेशन (डिझाईन स्पेसिफिकेशन्सनुसार प्रॉडक्ट योग्य प्रकारे तयार झाले आहे का) आणि व्हॅलिडेशन (डेव्हलप केलेले प्रॉडक्ट युजरच्या गरजा पूर्ण करते का) या दोन्ही बाबींचा समावेश होतो. तसेच, रिलीजपूर्वी फंक्शनल, परफॉर्मन्स, सिक्युरिटी आणि युजेबिलिटी संबंधित समस्या ओळखून सॉफ्टवेअरची एकूण क्वालिटी सुधारण्यावर टेस्टिंग भर देते. अव्हेलेबिलिटी कायम ठेवून साध्य केले जाते. यासाठी योग्य सेक्युरिटी पॉलिसीज आणि टेक्नॉलॉजीजचा वापर केला जातो.

सॉफ्टवेअर टेस्टिंगचे प्रकार (Types of Software Testing)

सॉफ्टवेअर टेस्टिंग ही एक व्यापक प्रक्रिया आहे, जी ती कशा प्रकारे केली जाते, टेस्टरकडे कुठली माहिती उपलब्ध आहे, डेव्हलपमेंट प्रोसेसमधील कोणत्या टप्प्यावर टेस्टिंग केली जाते, आणि तिचा नेमका उद्देश काय आहे यावर आधारित विविध प्रकारांमध्ये वर्गीकृत करता येते.

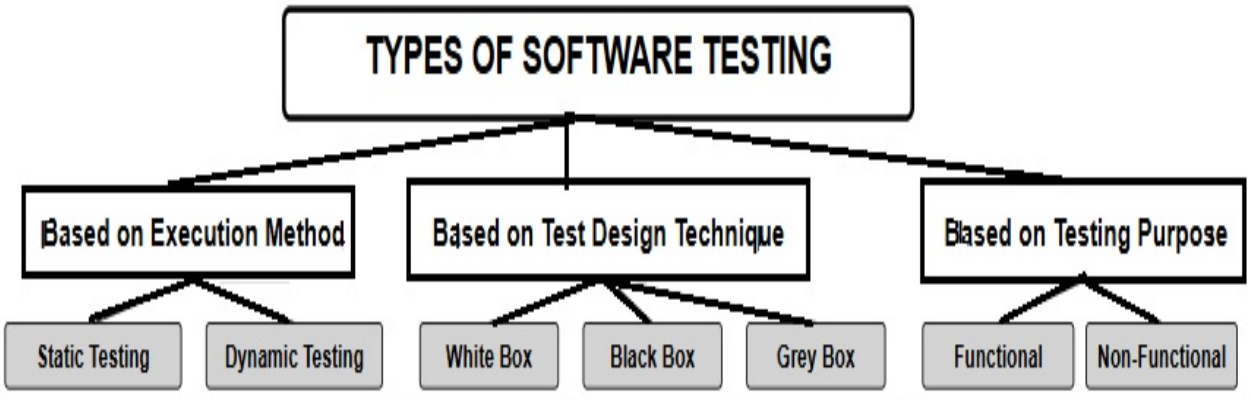

Fig 1.1: सॉफ्टवेअर टेस्टिंगचे प्रकार (Types of Software Testing)

1.1.1 टेस्टिंगची उद्दिष्टे (Objectives of Testing)

सॉफ्टवेअर टेस्टिंगचा मुख्य उद्देश म्हणजे विकसित केलेले सॉफ्टवेअर नेमके अपेक्षेप्रमाणे कार्य करते का आणि सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन (Software Requirement Specification (SRS)) मध्ये नमूद केलेल्या सर्व आवश्यकता पूर्ण करते का हे सुनिश्चित करणे होय. टेस्टिंगमुळे लवकर चुका ओळखता येतात, सॉफ्टवेअरची क्वालिटी सुधारते आणि डिप्लॉयमेंटनंतर होणाऱ्या फेल्युअर्सचा धोका कमी होतो.

टेस्टिंगची मुख्य उद्दिष्टे पुढीलप्रमाणे आहेत:

1 **एरर डिटेक्शन (Error Detection):** सॉफ्टवेअर युजरकडे रिलीज करण्यापूर्वी त्यातील फॉल्ट्स, डिफेक्ट्स किंवा बग्स ओळखणे.
2 **रिक्वायरमेंट व्हॅलिडेशन (Requirement Validation):** सॉफ्टवेअर SRS मध्ये नमूद केलेल्या सर्व फंक्शनल आणि नॉन-फंक्शनल रिक्वायरमेंट्स पूर्ण करते का हे निश्चित करणे.
3 **फंक्शनॅलिटी व्हेरिफिकेशन (Functionality Verification):** सिस्टीम अपेक्षित परिस्थितींमध्ये तसेच एक्सेप्शनल कंडिशन्स मध्येही योग्य प्रकारे कार्य करते का हे तपासणे.
4 **क्वालिटी ॲश्युरन्स (Quality Assurance):** प्रॉडक्टच्या रिलायबिलिटी, परफॉर्मन्स आणि सिक्युरिटी बाबत विश्वास निर्माण करणे.
5 **रिस्क रिडक्शन (Risk Reduction):** ऑपरेशन दरम्यान होणाऱ्या महागड्या फेल्युअर्स ची शक्यता कमी करणे.
6 **कॉम्प्लायन्स (Compliance):** सॉफ्टवेअर संबंधित स्टँडर्ड्स, गाईडलाईन्स आणि इंडस्ट्री बेस्ट प्रॅक्टिसेस पूर्ण करते का हे व्हेरिफाय करणे.

1.1.2 सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन (Software Requirement Specification (SRS))

सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन (SRS) हा एक फॉर्मल आणि स्ट्रक्चर्ड डॉक्युमेंट आहे, जो सॉफ्टवेअर सिस्टीमसाठी आवश्यक असलेल्या संपूर्ण रिक्वायरमेंट्सचा संच परिभाषित करतो. यामध्ये सिस्टीमने पार पाडावयाच्या विशिष्ट ऑपरेशन्स आणि फीचर्स सांगणाऱ्या फंक्शनल रिक्वायरमेंट्स तसेच परफॉर्मन्स, रिलायबिलिटी, सिक्युरिटी आणि युजेबिलिटी यांसारख्या बाबींचा समावेश असलेल्या नॉन-फंक्शनल रिक्वायरमेंट्स यांचा समावेश असतो. सॉफ्टवेअर इंजिनिअरिंगच्या मान्यताप्राप्त साहित्यानुसार, SRS हे डेव्हलपमेंट प्रोसेससाठी एक ब्लूप्रिंट म्हणून काम करते आणि क्लायंटच्या अपेक्षा व डेव्हलपमेंट टीमच्या इम्प्लिमेंटेशन यांमधील दरी भरून काढते.

सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन SRS चे महत्त्व:

सॉफ्टवेअर डेव्हलपमेंट लाईफ सायकलमध्ये SRS ची भूमिका अत्यंत महत्त्वाची आहे, कारण ते:

1. सॉफ्टवेअरच्या अपेक्षित फंक्शनॅलिटी आणि कन्स्ट्रेंट्स यांचे स्पष्ट आणि संदिग्धतारहित (unambiguous) वर्णन प्रदान करते.
2. डिझाईन, कोडिंग आणि टेस्टिंग ॲक्टिव्हिटीज साठी एक रेफरन्स पॉईंट म्हणून काम करते.
3. क्लायंट्स, डेव्हलपर्स आणि टेस्टर यांसारख्या सर्व स्टेकहोल्डर्स मध्ये प्रोजेक्टचा स्कोप आणि उद्दिष्टे याबाबत समान समज निर्माण करते.
4. संभाव्य समस्या लवकर ओळखण्यास मदत करते, ज्यामुळे रीवर्क आणि प्रोजेक्ट कॉस्ट कमी होते.
5. व्हेरिफिकेशन, व्हॅलिडेशन आणि भविष्यातील मेंटेनन्स साठी एक बेसलाईन डॉक्युमेंट म्हणून कार्य करते.

चांगल्या SRS ची मुख्य वैशिष्ट्ये (Key Characteristics of a Good SRS):

प्रस्थापित सॉफ्टवेअर इंजिनिअरिंग गाईडलाईन्स नुसार, उच्च दर्जाचे SRS खालीलप्रमाणे असावे:

1. कररेक्ट (Correct): अपेक्षित सिस्टीमचे अचूक प्रतिनिधित्व करणारे असावे.
2. अनॲम्बिग्युअस (Unambiguous): प्रत्येक रिक्वायरमेंट स्पष्टपणे मांडलेली असावी, ज्याचा फक्त एकच अर्थ निघतो.
3. कम्प्लीट (Complete): सर्व आवश्यक फंक्शनल, नॉन-फंक्शनल आणि इंटरफेस रिक्वायरमेंट्स समाविष्ट असाव्यात.
4. कन्सिस्टंट (Consistent): अंतर्गत विसंगती (contradictions) नसलेले असावे.
5. व्हेरिफायएबल (Verifiable): मोजता येणाऱ्या आणि टेस्ट करता येणाऱ्या टर्म्स मध्ये व्यक्त केलेले असावे.
6. मॉडिफायएबल (Modifiable): बदल सहजपणे करता येतील अशा पद्धतीने ऑर्गनाईज केलेले असावे.
7. ट्रेसेबल (Traceable): प्रत्येक रिक्वायरमेंटचा त्याच्या सोर्सशी तसेच संबंधित डिझाईन किंवा इम्प्लिमेंटेशन एलिमेंट्स शी दुवा असावा.

SRS डॉक्युमेंटची रचना (IEEE 830 स्टँडर्ड)

1. इंट्रोडक्शन (Introduction)

- पर्पज (Purpose): सॉफ्टवेअर सिस्टीमची उद्दिष्टे आणि ध्येय स्पष्ट करते.
- स्कोप (Scope): सिस्टीम काय करेल आणि काय करणार नाही हे वर्णन करते.

- डेफिनिशन्स, ॲक्रोनिम्स आणि अब्रिविएशन्स (Definitions, Acronyms, and Abbreviations): टर्मिनॉलॉजीची स्पष्टता सुनिश्चित करते.

2. **ओव्हरऑल डिस्क्रिप्शन (Overall Description)**
   - प्रॉडक्ट पर्स्पेक्टिव्ह (Product Perspective): सॉफ्टवेअर इतर सिस्टीम्समध्ये कसे बसते किंवा त्यांच्याशी कसे इंटरॅक्ट करते हे स्पष्ट करते.
   - प्रॉडक्ट फंक्शन्स (Product Functions): सिस्टीमच्या मुख्य क्षमतांचा सारांश देते.
   - युजर कॅरेक्टरिस्टिक्स (User Characteristics): अपेक्षित युजर्स, त्यांची कौशल्य पातळी आणि अपेक्षा वर्णन करते.
   - कन्स्ट्रेंट्स (Constraints): रेग्युलेटरी, हार्डवेअर किंवा डिझाईन मर्यादा सूचीबद्ध करते.
3. **स्पेसिफिक रिक्वायरमेंट्स (Specific Requirements)**
   - फंक्शनल रिक्वायरमेंट्स (Functional Requirements): सॉफ्टवेअरने पुरवायची फीचर्स आणि बिहेविअर्स स्पष्ट करतात.
   - परफॉर्मन्स रिक्वायरमेंट्स (Performance Requirements): स्वीकारार्ह रिस्पॉन्स टाइम, थ्रूपुट आणि स्केलेबिलिटी नमूद करतात.
   - डिझाईन कन्स्ट्रेंट्स (Design Constraints): टेक्नॉलॉजी स्टँडर्ड्स, प्रोग्रामिंग लँग्वेजेस आणि इंटरफेस रिक्वायरमेंट्स समाविष्ट करतात.
   - एक्सटर्नल इंटरफेस रिक्वायरमेंट्स (External Interface Requirements): हार्डवेअर डिव्हाइसेस, सॉफ्टवेअर सिस्टीम्स आणि कम्युनिकेशन प्रोटोकॉल्स सोबतचे इंटरॅक्शन्स परिभाषित करतात.

उदाहरण SRS – ऑनलाइन स्टुडंट ॲटेन्डन्स सिस्टीम

1. परिचय (Introduction)

1.1 उद्देश (Purpose): या सिस्टीमचा उद्देश शाळा किंवा महाविद्यालयांसाठी विद्यार्थ्यांची उपस्थिती डिजिटल पद्धतीने नोंदवणे आणि व्यवस्थापन करणे हा आहे. ही सिस्टीम मॅन्युअल रजिस्टर ची जागा घेऊन सोप्या आणि वापरण्यास सुलभ संगणकीकृत सिस्टीम प्रदान करेल.

1.2 स्कोप (Scope): ही सिस्टीम पुढील गोष्टी करेल:
   - शिक्षकांना प्रत्येक वर्गासाठी उपस्थिती नोंदवण्याची सुविधा देईल.
   - उपस्थिती रेकॉर्ड्स साठवेल आणि गरजेनुसार परत मिळवता येतील.
   - विद्यार्थ्यांसाठी मासिक उपस्थिती अहवाल (Monthly Attendance Reports) तयार करेल.

1.3 व्याख्या (Definitions): ॲटेन्डन्स रिपोर्ट (Attendance Report) – एखादा विद्यार्थी किती दिवस उपस्थित किंवा अनुपस्थित होता याचा सारांश दर्शवणारा अहवाल.

2. ओव्हरऑल डिस्क्रिप्शन (Overall Description)

2.1 प्रॉडक्ट पर्स्पेक्टिव्ह (Product Perspective):ही एक स्टँड-अलोन ॲप्लिकेशन असेल, जी शाळेच्या संगणकावर इन्स्टॉल केली जाईल. सर्व डेटा लोकल डेटाबेस मध्ये साठवला जाईल.

2.2 प्रॉडक्ट फंक्शन्स (Product Functions):
   - विद्यार्थ्यांची माहिती ॲड करणे.
   - दैनंदिन उपस्थिती नोंदवणे.
   - उपस्थिती अहवाल तयार करणे.

2.3 युजर कॅरेक्टरिस्टिक्स (User Characteristics): ही सिस्टीम शिक्षक वापरणार आहेत. त्यांना मूलभूत संगणक ज्ञान असणे अपेक्षित आहे.

3. स्पेसिफिक रिक्वायरमेंट्स (Specific Requirements)

3.1 फंक्शनल रिक्वायरमेंट्स (Functional Requirements)
   - सिस्टीमने विद्यार्थ्यांचे रेकॉर्ड्स ॲड, एडिट आणि डिलीट करण्याची सुविधा दिली पाहिजे.

- सिस्टीमने प्रत्येक विद्यार्थ्याला त्या दिवशी Present किंवा Aabsent म्हणून मार्क करण्याची सुविधा दिली पाहिजे.
- सिस्टीमने प्रत्येक विद्यार्थ्यासाठी मासिक उपस्थिती अहवाल जनरेट केला पाहिजे.

3.2 नॉन-फंक्शनल रिक्वायरमेंट्स (Non-Functional Requirements)

- सिस्टीम Windows OS वर कार्यरत असली पाहिजे.
- अहवाल 2 सेकंदांच्या आत जनरेट झाले पाहिजेत.
- फक्त ऑथराईज्ड स्टाफ ला लॉगिन करण्याची परवानगी असावी.

1.2 फेल्युअर, एरर, फॉल्ट, डिफेक्ट, बग टर्मिनॉलॉजी (Failure, Error, Fault, Defect, Bug Terminology)

सॉफ्टवेअर इंजिनिअरिंगमध्ये योग्य टर्मिनॉलॉजी वापरणे अत्यंत महत्त्वाचे असते, कारण त्यामुळे डेव्हलपमेंट आणि टेस्टिंग प्रक्रियेमधील समस्या अचूकपणे ओळखता आणि सोडवता येतात. एरर, फॉल्ट, डिफेक्ट, बग आणि फेल्युअर हे शब्द एकमेकांशी संबंधित असले तरी, प्रत्येकाचा अर्थ वेगळा आहे आणि ते समस्येच्या लाईफ सायकलमधील वेगवेगळे टप्पे दर्शवतात.

a. फेल्युअर (Failure)

फेल्युअर म्हणजे सॉफ्टवेअर सिस्टीमचे प्रत्यक्ष दिसणारे चुकीचे वर्तन, जेव्हा ऑपरेशन दरम्यान एखादा फॉल्ट एक्झिक्यूट होतो. सिस्टीमचा आउटपुट किंवा वर्तन हे रिक्वायरमेंट्समध्ये अपेक्षित असलेल्या निकालापेक्षा वेगळे असल्यास फेल्युअर घडतो.

उदाहरण: रेल्वे आरक्षण प्रणाली चुकीच्या भाडे गणनेमुळे तिकीटासाठी चुकीची रक्कम आकारते.

b. एरर (Error)

एरर म्हणजे सॉफ्टवेअर डेव्हलपमेंटच्या कोणत्याही टप्प्यावर माणसाकडून होणारी चूक. ही चूक रिक्वायरमेंट्स चुकीच्या प्रकारे समजल्यामुळे, चुकीचा कोड लिहिल्यामुळे किंवा दोषपूर्ण डिझाईन केल्यामुळे होऊ शकते. एरर ही बहुतेक सॉफ्टवेअर समस्यांची मूळ कारणे (root cause) असतात आणि त्या पुढे फॉल्ट्स निर्माण करतात.

उदाहरण: डेव्हलपर total = price * quantity असे लिहितो, प्रत्यक्षात total = price * quantity + tax असणे आवश्यक होते.

c. फॉल्ट (Fault)

फॉल्ट म्हणजे एररमुळे सॉफ्टवेअरच्या कोड किंवा डिझाईनमध्ये निर्माण झालेली चूक. फॉल्ट लगेचच समस्या निर्माण करेलच असे नाही, पण जेव्हा तो कोड एक्झिक्यूट होतो, तेव्हा सिस्टीमचे वर्तन चुकीचे होऊ शकते.

उदाहरण: बँकिंग ॲप्लिकेशनमध्ये व्याज मोजण्यासाठी चुकीचे फॉर्म्युला कोडमध्ये वापरणे.

d. डिफेक्ट (Defect)

डिफेक्ट म्हणजे सॉफ्टवेअर प्रॉडक्टमधील अशी त्रुटी, ज्यामुळे सॉफ्टवेअर अपेक्षेप्रमाणे कार्य करत नाही किंवा रिक्वायरमेंट्स पूर्ण करत नाही. हा शब्द सहसा डिफेक्ट ट्रॅकिंग सिस्टिम्स मध्ये वापरला जातो, जिथे समस्या नोंदवल्या जातात आणि त्या सोडविल्या जाईपर्यंत मॉनिटर केल्या जातात.

उदाहरण: रजिस्ट्रेशन फॉर्ममध्ये Email Address फील्डसाठी इनपुट व्हॅलिडेशन नसणे.

e. बग (Bug)

बग हा डिफेक्ट साठी वापरला जाणारा अनौपचारिक (informal) शब्द आहे. सॉफ्टवेअर अपेक्षेप्रमाणे कार्य करण्यास अडथळा आणणारी कोणतीही समस्या म्हणजे बग. दैनंदिन भाषेत "बग" जास्त वापरला जातो, पण औपचारिक डॉक्युमेंटेशनमध्ये "डिफेक्ट" हा शब्द अधिक योग्य मानला जातो.

उदाहरण: असमर्थित (unsupported) फाईल फॉरमॅट अपलोड केल्यावर ॲप्लिकेशन फ्रीझ होणे.

1.3. टेस्ट केस, एंट्री आणि एक्झिट क्रायटेरिया फॉर टेस्टिंग (Test Case, Entry and Exit Criteria for Testing)

1.3.1. टेस्ट केस (Test Case)

टेस्ट केस म्हणजे सॉफ्टवेअर ॲप्लिकेशनमधील एखाद्या विशिष्ट फीचर किंवा फंक्शनॅलिटीची चाचणी घेण्यासाठी तयार केलेल्या सूचनांचा (instructions) सुव्यवस्थित संच होय. टेस्ट केसमध्ये ठरावीक इनपुट्स, एक्झिक्यूशनसाठी सविस्तर स्टेप्स आणि अपेक्षित निकाल (Expected Results) दिलेले असतात, ज्यांच्या आधारे प्रत्यक्ष मिळालेल्या निकालांची तुलना केली जाते. टेस्ट केसचा मुख्य उद्देश म्हणजे ठरवलेल्या परिस्थितीत सॉफ्टवेअर योग्य प्रकारे वागत आहे का हे तपासणे आणि अपेक्षित वर्तनापासून झालेली चूक किंवा डिफेक्ट ओळखणे.

टेस्ट केसचे घटक (Components of a Test Case)

1. टेस्ट केस आयडी (Test Case ID): प्रत्येक टेस्ट केससाठी दिलेला युनिक ओळख क्रमांक, जो संदर्भासाठी वापरला जातो.
2. टेस्ट ऑब्जेक्टिव्ह / डिस्क्रिप्शन (Test Objective / Description): कोणती फंक्शनॅलिटी किंवा कोणता सीनारिओ टेस्ट केला जात आहे याचे संक्षिप्त वर्णन.
3. टेस्ट स्टेप्स / इनपुट (Test Steps / Input): टेस्ट करण्यासाठी आवश्यक असलेल्या स्टेप-बाय-स्टेप सूचना आणि इनपुट्स.
4. अपेक्षित निकाल (Expected Results): सॉफ्टवेअर योग्य प्रकारे काम केल्यास मिळायला हवा असा अपेक्षित परिणाम.
5. प्रत्यक्ष निकाल (Actual Results): टेस्ट एक्झिक्यूशननंतर प्रत्यक्ष मिळालेला निकाल.
6. स्टेटस (Status): अपेक्षित निकाल आणि प्रत्यक्ष निकाल यांच्या तुलनेनुसार टेस्ट केस यशस्वी (Passed) किंवा अयशस्वी (Failed) झाला आहे का ते दर्शवते.

उदाहरण 1: LOGIN फॉर्मसाठी टेस्ट केस डिझाईन करा

Test Case ID (टेस्ट केस आयडी)	Objective / Description (उद्दिष्ट / वर्णन)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
TC001	Valid username and valid password वैध युजरनेम आणि वैध पासवर्ड	username: user1,password: Pass@123	Login successful, redirect to dashboard लॉगिन यशस्वी, डॅशबोर्डकडे रिडायरेक्ट	Login successful, redirected to dashboard लॉगिन यशस्वी, डॅशबोर्डकडे रिडायरेक्ट	Pass (यशस्वी)
TC002	Invalid username and valid password अवैध युजरनेम आणि वैध पासवर्ड	username: wrongUser,password: Pass@123	Login failed, show error “Invalid username or password” लॉगिन अयशस्वी, “Invalid username or password” एरर दाखवावी	Login successful, redirected to dashboard (bug) लॉगिन यशस्वी, डॅशबोर्डकडे रिडायरेक्ट (बग)	Fail (अयशस्वी)

TC003	Valid username and invalid password वैध युजरनेम आणि अवैध पासवर्ड	username: user1,password: wrongPass	Login failed, show error “Invalid username or password” लॉगिन अयशस्वी, एरर दाखवावी	Login failed, error displayed लॉगिन अयशस्वी, एरर दाखवली	Pass (यशस्वी)
TC004	Invalid username and invalid password अवैध युजरनेम आणि अवैध पासवर्ड	username: wrongUser,password: wrongPass	Login failed, show error “Invalid username or password” लॉगिन अयशस्वी, एरर दाखवावी	Login failed, error displayed लॉगिन अयशस्वी, एरर दाखवली	Pass (यशस्वी)
TC005	Empty username and valid password रिकामा युजरनेम आणि वैध पासवर्ड	username: (empty),password: Pass@123	Login failed, show error “Username cannot be empty” लॉगिन अयशस्वी, “Username cannot be empty” एरर दाखवावी	Login failed, error displayed लॉगिन अयशस्वी, एरर दाखवली	Pass (यशस्वी)
TC006	Valid username and empty password वैध युजरनेम आणि रिकामा पासवर्ड	username: user1,password: (empty)	Login failed, show error “Password cannot be empty” लॉगिन अयशस्वी, “Password cannot be empty” एरर दाखवावी	Login accepted (bug) लॉगिन स्वीकारले (बग)	Fail (अयशस्वी)

या सेटमध्ये TC002 आणि TC006 हे Fail म्हणून मार्क केलेले आहेत, जे सिस्टीम डिफेक्ट्स दर्शवतात:

- TC002: सिस्टीमने अवैध (invalid) युजरनेम असतानाही लॉगिन करण्याची परवानगी दिली.
- TC006: सिस्टीमने पासवर्डशिवाय लॉगिन करण्याची परवानगी दिली.

ही उदाहरणे दाखवतात की टेस्ट केसेस कशा प्रकारे ॲप्लिकेशनमधील फंक्शनल इश्यूज प्रभावीपणे ओळखतात.

उदाहरण 2: स्टँडर्ड कॅल्क्युलेटरसाठी टेस्ट केसेस डिझाईन करा

Test Case ID (टेस्ट केस आयडी)	Objective / Description (उद्दिष्ट / वर्णन)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
TC001	Addition of two positive numbers दोन सकारात्मक संख्यांची बेरीज	5 + 3	8	8	Pass (यशस्वी)
TC002	Addition of positive and negative number सकारात्मक व नकारात्मक संख्यांची बेरीज	7 + (-2)	5	6 (bug) 6 (बग)	Fail (अयशस्वी)

TC003	Subtraction of two positive numbers दोन सकारात्मक संख्यांची वजाबाकी	9 - 4	5	5	Pass (यशस्वी)
TC004	Subtraction resulting in negative number नकारात्मक निकाल देणारी वजाबाकी	3 - 8	-5	-5	Pass (यशस्वी)
TC005	Multiplication of two positive numbers दोन सकारात्मक संख्यांचा गुणाकार	6 × 7	42	42	Pass (यशस्वी)
TC006	Multiplication by zero शून्याने गुणाकार	9 × 0	0	1 (bug) 1 (बग)	Fail (अयशस्वी)
TC007	Division of two positive numbers दोन सकारात्मक संख्यांचे भागाकार	20 ÷ 4	5	5	Pass (यशस्वी)
TC008	Division by zero शून्याने भागाकार	5 ÷ 0	Error / Cannot divide by zero त्रुटी / शून्याने भागाकार करता येत नाही	0 (bug) 0 (बग)	Fail (अयशस्वी)

टेस्ट केस रिझल्ट्समध्ये Pass आणि Fail अशा दोन्ही प्रकारचे आऊटकम्स समाविष्ट आहेत:

- Pass: TC001, TC003, TC004, TC005, TC007
- Fail: TC002, TC006, TC008

Fail झालेले टेस्ट केसेस संभाव्य फंक्शनल डिफेक्ट्स दर्शवतात, जसे की:

- निगेटिव्ह नंबर्स वापरल्यावर चुकीची बेरीज होणे.
- शून्याने गुणाकार केल्यावर चुकीचा निकाल मिळणे.
- शून्याने भागाकार करताना अपेक्षित एरर जनरेट न होणे

यावरून स्पष्ट होते की सिस्टिमॅटिक टेस्टिंग केवळ सामान्य ऑपरेशन्समधील चुका नाही तर एज-केस इश्यूज ओळखण्यास देखील प्रभावी ठरते.

1.3.2. टेस्टिंगसाठी एंट्री आणि एग्झिट क्रायटेरिया (Entry and Exit Criteria for Testing)

a. एंट्री क्रायटेरिया (Entry Criteria): एंट्री क्रायटेरिया म्हणजे टेस्टिंग प्रक्रिया सुरू करण्यापूर्वी पूर्ण करणे आवश्यक असलेल्या विशिष्ट अटी (conditions) होत. या अटी पूर्ण झाल्या आहेत याची खात्री केल्यामुळे टेस्टिंग टीम योग्य प्रकारे तयार असते आणि टेस्टिंग एन्व्हायर्नमेंट प्रभावी एक्झिक्यूशनसाठी योग्य असते. टेस्टिंग सुरू करण्यापूर्वी या अटी पूर्ण झाल्यास, प्रक्रिया योग्य परिस्थितीत सुरू होते आणि सुरळीतपणे पुढे जाते.

सामान्य एंट्री क्रायटेरिया:

1. **रिक्वायरमेंट्स अप्रूव्हल (Requirements Approval):** सर्व सॉफ्टवेअर रिक्वायरमेंट्स स्पष्टपणे परिभाषित केलेल्या आणि औपचारिकरित्या मंजूर झालेल्या असाव्यात.
2. **टेस्ट प्लॅनिंग (Test Planning):** टेस्ट डिझाईन, टेस्ट केसेस आणि संबंधित डॉक्युमेंटेशन तयार केलेले व रिव्ह्यू झालेले असावे.
3. **टेस्ट एन्व्हायर्नमेंट (Test Environment):** टेस्ट एक्झिक्यूट करण्यासाठी आवश्यक असलेले हार्डवेअर, सॉफ्टवेअर, नेटवर्क कॉन्फिगरेशन्स, सेटिंग्स आणि टूल्स उपलब्ध असावेत.
4. **टेस्ट डेटा (Test Data):** योग्य आणि पुरेसा टेस्ट डेटा उपलब्ध असावा.
5. **टीमची तयारी (Team Preparedness):** टेस्टर प्रशिक्षित असावेत आणि आवश्यक रिसोर्सेस उपलब्ध असावेत.

6. **डेव्हलपमेंट पूर्णता (Development Completion):** वापरल्या जाणाऱ्या सॉफ्टवेअर डेव्हलपमेंट मॉडेल नुसार डेव्हलपमेंट आणि युनिट टेस्टिंग फेज पूर्ण झालेला असावा.
   या पूर्वअटी पूर्ण केल्यामुळे विलंब टाळता येतो आणि टेस्टिंग ॲक्टिव्हिटीजसाठी एक मजबूत पाया तयार होतो.

b. एग्झिट क्रायटेरिया (Exit Criteria): एग्झिट क्रायटेरिया म्हणजे टेस्टिंग फेज यशस्वीरीत्या पूर्ण करण्यासाठी आवश्यक असलेल्या अटी होत. या अटी पूर्ण झाल्यावर हे निश्चित होते की सॉफ्टवेअरची सखोल टेस्टिंग झाली आहे आणि ते रिलीज किंवा डेव्हलपमेंटच्या पुढील टप्प्यासाठी तयार आहे. टेस्टिंग प्रक्रिया फक्त तेव्हाच संपते, जेव्हा हे पूर्वनिश्चित एग्झिट क्रायटेरिया पूर्ण होतात.

सामान्य एग्झिट क्रायटेरिया:

1. **टेस्ट केस एक्झिक्यूशन (Test Case Execution):** सर्व क्रिटिकल आणि हाय-प्रायोरिटी टेस्ट केसेसचे एक्झिक्यूशन पूर्ण झालेले असावे.
2. **डिफेक्ट रिझोल्यूशन (Defect Resolution):** प्रमुख डिफेक्ट्स बंद (closed) केलेले असावेत किंवा ज्ञात मायनर डिफेक्ट्स योग्य डॉक्युमेंटेशनसह औपचारिकरित्या स्वीकारलेले असावेत.
3. **टेस्ट कव्हरेज (Test Coverage):** आवश्यक असलेले फंक्शनल आणि कोड कव्हरेज लेव्हल्स साध्य झालेले असावेत.
4. **क्वालिटी स्टँडर्ड्स (Quality Standards):** सॉफ्टवेअरने परिभाषित केलेल्या फंक्शनल आणि नॉन-फंक्शनल रिक्वायरमेंट्स पूर्ण केल्याची खात्री झालेली असावी.
5. **टेस्ट रिपोर्टिंग (Test Reporting):** टेस्ट समरी रिपोर्ट्स पूर्ण झालेले असावेत आणि संबंधित स्टेकहोल्डर्स कडून मंजुरी मिळालेली असावी.
6. **मॅनेजमेंट डिसिजन (Management Decision):** डिप्लॉयमेंट किंवा रिलीजसाठी पुढे जाण्याचा अंतिम निर्णय मॅनेजमेंटकडून मिळालेला असावा.

एग्झिट क्रायटेरियाचे पालन केल्यामुळे सॉफ्टवेअर स्थिर (stable), विश्वासार्ह (dependable) आणि आवश्यक क्वालिटी स्टँडर्ड्स पूर्ण करणारे आहे याची खात्री होते, आणि त्यामुळे ते डिप्लॉयमेंटसाठी तयार असल्याचे सिद्ध होते.

1.4 टेस्टिंगच्या पद्धती (Methods of Testing)

सॉफ्टवेअरची तपासणी कशा प्रकारे केली जाते यावर आधारित सॉफ्टवेअर टेस्टिंगच्या पद्धती प्रामुख्याने स्टॅटिक टेस्टिंग आणि डायनॅमिक टेस्टिंग अशा दोन प्रकारांमध्ये विभागल्या जातात.

1.4.1 स्टॅटिक टेस्टिंग (Static Testing)

स्टॅटिक टेस्टिंग ही सॉफ्टवेअर प्रोग्राम रन न करता केली जाणारी टेस्टिंग पद्धत आहे. या पद्धतीत डॉक्युमेंट्स, डिझाईन डायग्राम्स आणि सोर्स कोड यांसारख्या विविध आर्टिफॅक्ट्स चे परीक्षण करून डेव्हलपमेंट लाईफ सायकलच्या सुरुवातीच्या टप्प्यातच चुका ओळखण्यावर भर दिला जातो. स्टॅटिक टेस्टिंगमध्ये सामान्यतः रिव्ह्यूज, वॉकथ्रूज आणि इन्स्पेक्शन्स यांसारख्या तंत्रांचा वापर केला जातो. या पद्धतीमुळे चुकीचे अल्गोरिदम्स, कोडिंग स्टँडर्ड्सपासून झालेले विचलन, अपूर्ण रिक्वायरमेंट्स आणि डिझाईनमधील विसंगती यांसारख्या समस्या कोडिंग फेज पूर्ण होण्यापूर्वीच ओळखता येतात. लवकर टप्प्यात डिफेक्ट्स सापडल्यामुळे नंतर समस्या दुरुस्त करण्यासाठी लागणारा वेळ आणि खर्च लक्षणीयरीत्या कमी होतो, म्हणून स्टॅटिक टेस्टिंग ही अत्यंत उपयुक्त पद्धत मानली जाते.

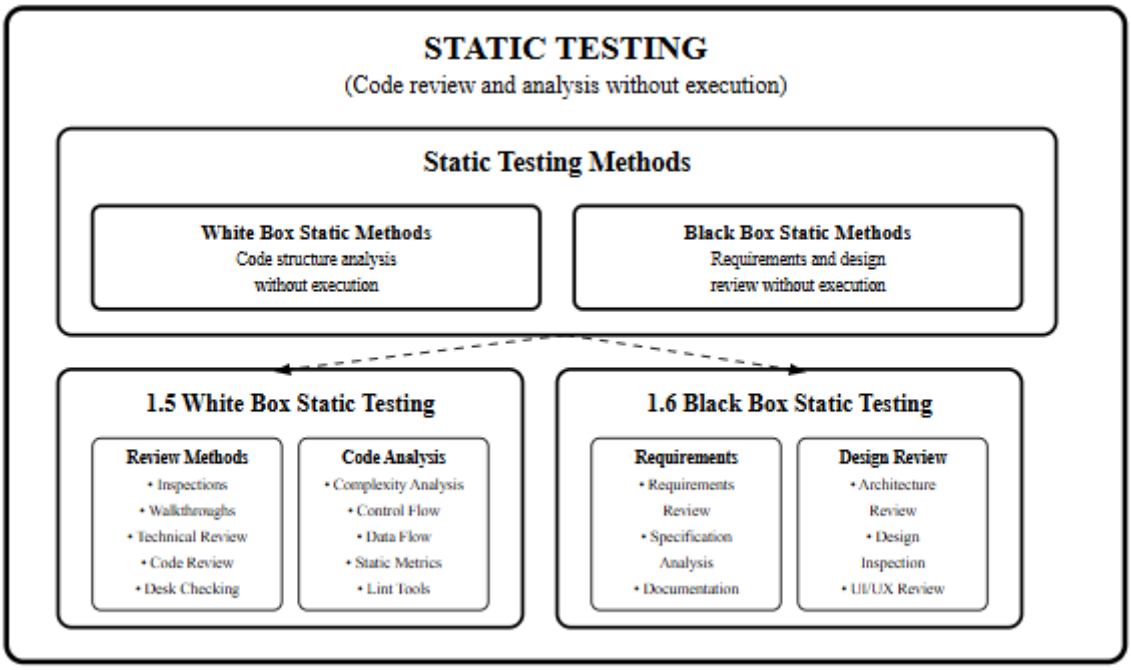

Fig 1.2: स्टॅटिक टेस्टिंग मेथड्स (Static Testing methods)

उदाहरण (Example): कोडिंग प्रक्रिया सुरू करण्यापूर्वी Software Requirement Specification (SRS) डॉक्युमेंटचे रिव्ह्यू करून अपूर्ण किंवा अस्पष्ट रिक्वायरमेंट्स ओळखणे.

1.4.2 डायनॅमिक टेस्टिंग (Dynamic Testing)

डायनॅमिक टेस्टिंग मध्ये सॉफ्टवेअर विविध परिस्थितींमध्ये एक्झिक्यूट करून त्याचे वर्तन तपासले जाते. ही टेस्टिंग कोड डेव्हलप आणि कम्पाइल झाल्यानंतर केली जाते. या टेस्टिंगचा मुख्य उद्देश टेस्ट केसेस रन करून प्रत्यक्ष मिळालेल्या निकालांची अपेक्षित निकालांशी तुलना करून सॉफ्टवेअरचे फंक्शनल आणि नॉन-फंक्शनल पैलू व्हॅलिडेट करणे हा आहे. डायनॅमिक टेस्टिंगच्या उदाहरणांमध्ये फंक्शनल टेस्टिंग, परफॉर्मन्स टेस्टिंग आणि युजेबिलिटी टेस्टिंग यांचा समावेश होतो. सॉफ्टवेअर रन होत असताना दिसणाऱ्या आणि स्टॅटिक टेस्टिंगमध्ये न सापडणाऱ्या त्रुटी शोधण्यासाठी डायनॅमिक टेस्टिंग अत्यंत महत्त्वाची असते.

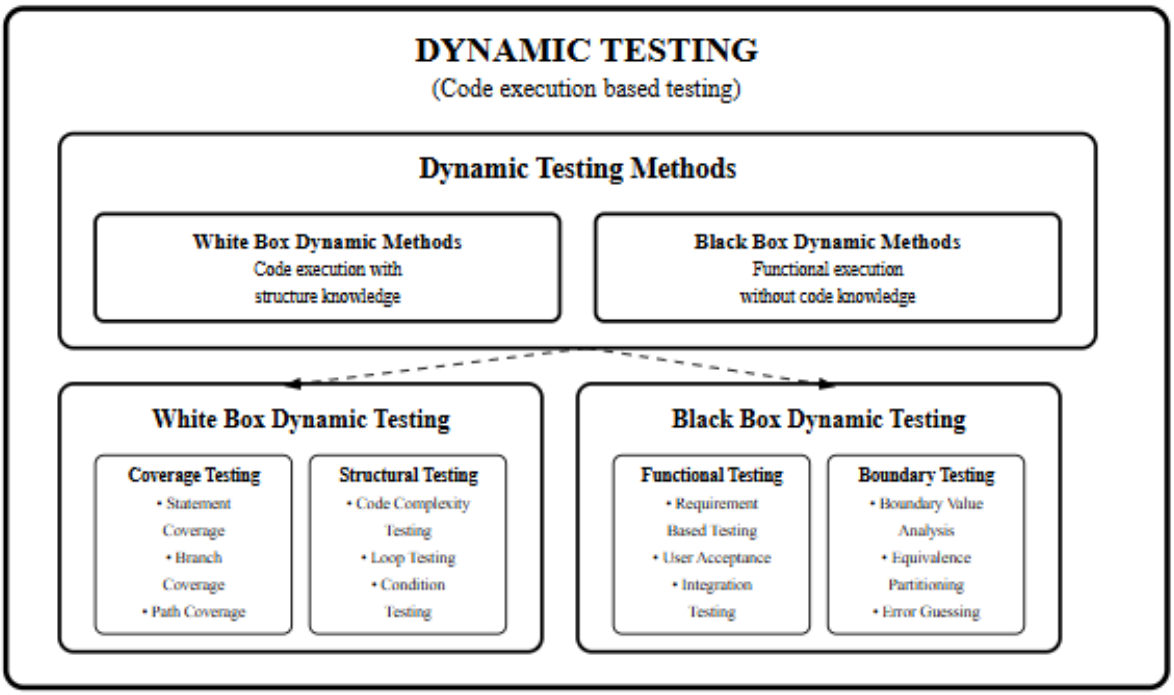

Fig 1.3: डायनॅमिक टेस्टिंग मेथड्स (Dynamic Testing methods)

उदाहरण (Example): ऑनलाईन बँकिंग ॲप्लिकेशनमधील "फंड ट्रान्सफर" फीचरची टेस्टिंग करून एक्झिक्यूशन दरम्यान रक्कम खात्यांदरम्यान योग्य प्रकारे ट्रान्सफर होत आहे का हे सुनिश्चित करणे.

Table 1.1: स्टॅटिक टेस्टिंग वि. डायनॅमिक टेस्टिंग (Comparison between Static and Dynamic Testing)

निकष (Criteria)	स्टॅटिक टेस्टिंग (Static Testing)	डायनॅमिक टेस्टिंग (Dynamic Testing)
व्याख्या (Definition)	सॉफ्टवेअर एक्झिक्यूट न करता केली जाणारी टेस्टिंग; डॉक्युमेंट्स, कोड आणि डिझाईनचे रिव्ह्यू केले जाते.	सॉफ्टवेअर एक्झिक्यूट करून त्याचे वर्तन निरीक्षण करून केली जाणारी टेस्टिंग.
केव्हा केली जाते (When Performed)	सॉफ्टवेअर डेव्हलपमेंटच्या सुरुवातीच्या टप्प्यात, कोडिंगपूर्वी किंवा कोडिंगदरम्यान.	सॉफ्टवेअर डेव्हलप आणि कम्पाइल झाल्यानंतर.
उद्देश (Purpose)	रिक्वायरमेंट्समधील चुका, डिझाईन फ्लॉज किंवा कोडिंग स्टँडर्ड्सचे उल्लंघन लवकर ओळखणे.	प्रत्यक्ष एक्झिक्यूशनदरम्यान सॉफ्टवेअरचे फंक्शनल आणि नॉन-फंक्शनल वर्तन तपासणे.
वापरली जाणारी तंत्रे (Techniques Used)	रिव्ह्यूज, वॉकथ्रूज, इन्स्पेक्शन्स, स्टॅटिक कोड ॲनालिसिस टूल्स.	फंक्शनल टेस्टिंग, परफॉर्मन्स टेस्टिंग, युजेबिलिटी टेस्टिंग आणि इतर टेस्ट एक्झिक्यूशन पद्धती.
सापडणारे डिफेक्ट्स (Type of Defects Found)	डिझाईन एरर्स, अपूर्ण रिक्वायरमेंट्स, सिंटॅक्स एरर्स, स्टँडर्ड्सपासून झालेले विचलन.	रनटाइम एरर्स, लॉजिक एरर्स, परफॉर्मन्स इश्यूज, इंटिग्रेशन प्रॉब्लेम्स.
कॉस्ट इफेक्टिव्हनेस (Cost Effectiveness)	अधिक किफायतशीर, कारण डिफेक्ट्स लवकर सापडतात आणि नंतरचा रीवर्क कमी होतो.	उशिरा डिफेक्ट्स सापडल्यास खर्चिक ठरू शकते, कारण रनटाइम एरर्स दुरुस्त करण्यासाठी मोठे बदल करावे लागू शकतात.
वापरली जाणारी टूल्स (Tools Used)	स्टॅटिक कोड ॲनालायझर्स, डॉक्युमेंट रिव्ह्यू चेकलिस्ट्स.	टेस्ट ऑटोमेशन टूल्स, डिबगर्स, परफॉर्मन्स टेस्टिंग टूल्स.
आउटपुट (Output)	सॉफ्टवेअर एक्झिक्यूट न करता डॉक्युमेंट्स, कोड किंवा डिझाईनमधील डिफेक्ट्सचे रिपोर्ट्स.	एक्झिक्यूट केलेल्या टेस्ट केसेसवर आधारित Pass/Fail स्टेटस दर्शवणारे टेस्ट रिझल्ट्स.
उदाहरण (Example)	SRS डॉक्युमेंटचे रिव्ह्यू करून अस्पष्ट किंवा अपूर्ण माहिती शोधणे.	लॉगिन फीचर एक्झिक्यूट करून युजर ऑथेंटिकेशन यशस्वी होत आहे का हे तपासणे.

1.5 व्हाईट बॉक्स टेस्टिंग (White Box Testing)

व्हाईट बॉक्स टेस्टिंग ही सॉफ्टवेअरच्या अंतर्गत रचनेवर (internal structure) आणि कार्यप्रणालीवर (functioning) लक्ष केंद्रित करणारी टेस्टिंग पद्धत आहे. याला स्ट्रक्चरल टेस्टिंग (Structural Testing) असेही म्हणतात. या टेस्टिंगसाठी सॉफ्टवेअरचा सोर्स कोड, लॉजिक आणि आर्किटेक्चर यांचे सखोल ज्ञान आवश्यक असते. या पद्धतीत टेस्टर विशिष्ट कोड पाथ्स, डिसिजन पॉइंट्स, लूप्स आणि ॲप्लिकेशनमधील डेटा फ्लो यांना लक्ष्य करून टेस्ट केसेस तयार करतो. व्हाईट बॉक्स टेस्टिंगमध्ये वापरल्या जाणाऱ्या सामान्य तंत्रांमध्ये कोड कव्हरेज ॲनालिसिस, कोड कॉम्प्लेक्सिटी मोजणे आणि पाथ टेस्टिंग यांचा समावेश होतो. या पद्धतीमुळे कोडमधील सर्व भाग योग्य प्रकारे एक्झिक्यूट आणि व्हेरिफाय झाले आहेत याची खात्री होते.

उदाहरण (Example): सॉर्टिंग अल्गोरिदमची टेस्टिंग करताना कोडमधील प्रत्येक कंडिशनल ब्रँच आणि लूप एक्झिक्यूट होत आहे का हे तपासणे, जेणेकरून कोडमधील सर्व शक्य परिस्थिती (scenarios) कव्हर होतील.

व्हाईट बॉक्स टेस्टिंगच्या पद्धती (Methods of White Box Testing):

व्हाईट बॉक्स टेस्टिंगमध्ये सॉफ्टवेअरच्या अंतर्गत कार्यप्रणालीचे सखोल मूल्यमापन करण्यासाठी विविध तंत्रांचा वापर केला जातो.

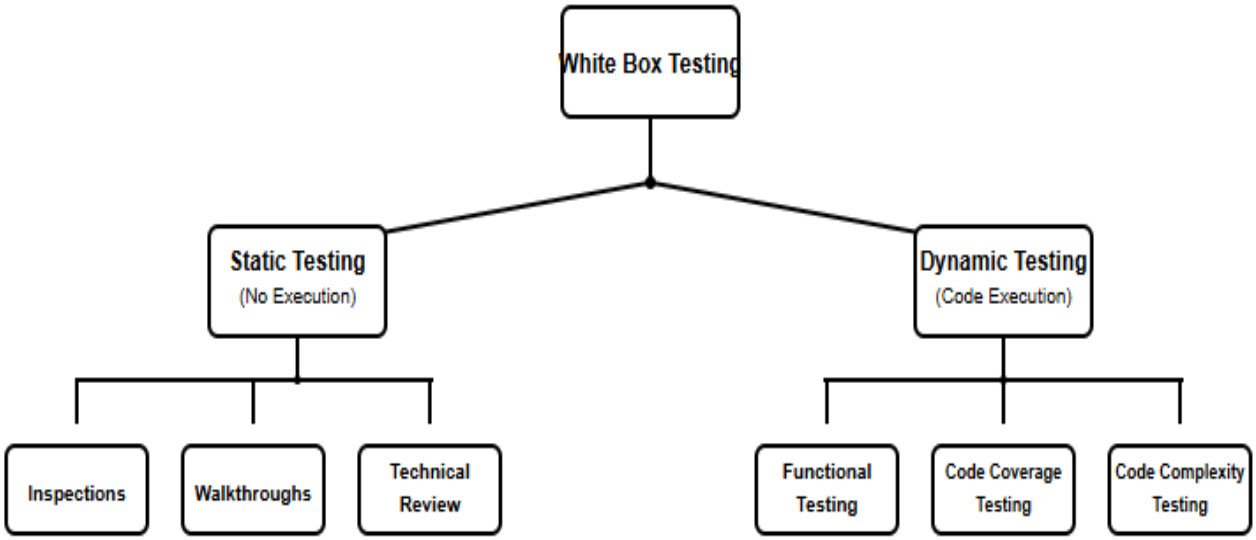

Fig 1.4: व्हाईट बॉक्स टेस्टिंग मेथड्स (White Box Testing methods)

1.5.1 इन्स्पेक्शन्स (Inspections)

इन्स्पेक्शन्स म्हणजे सोर्स कोड किंवा डिझाईन डॉक्युमेंट्स यांची तज्ञांकडून (experts) केली जाणारी औपचारिक (formal) आणि सखोल तपासणी होय. या प्रक्रियेचा उद्देश डिफेक्ट्स, कोडिंग स्टँडर्ड्सचे उल्लंघन किंवा लॉजिकल एरर्स ओळखणे हा असतो. या पद्धतीत सॉफ्टवेअर रन न करता विविध सॉफ्टवेअर आर्टिफॅक्ट्स जसे की सोर्स कोड, डिझाईन डॉक्युमेंट्स किंवा रिक्वायरमेंट्स यांचे परीक्षण केले जाते. या प्रक्रियेमध्ये तज्ञांची टीम संबंधित साहित्य काळजीपूर्वक अभ्यासते आणि विसंगती (inconsistencies), चुका किंवा प्रस्थापित स्टँडर्ड्सपासून झालेले विचलन ओळखते. इन्स्पेक्शन्सचा मुख्य उद्देश म्हणजे लॉजिकल मिस्टेक्स, चुकीचे अल्गोरिदम्स किंवा मिसिंग डिटेल्स यांसारख्या चुका सुरुवातीच्या टप्प्यातच शोधणे. ही प्रक्रिया मॉडरेटर, ऑथर आणि रिव्ह्यूअर्स अशा निश्चित भूमिका (roles) असलेल्या व्यक्तींकडून पार पाडली जाते आणि त्यामध्ये तयारी (preparation), सविस्तर तपासणी, डिफेक्ट्सची नोंद (logging) आणि फॉलो-अप ॲक्शन्स यांसारख्या ठरावीक स्टेप्सचा समावेश असतो. इन्स्पेक्शन्समुळे अशा समस्या ओळखता येतात ज्या केवळ टेस्टिंगद्वारे सापडतीलच असे नाही. त्यामुळे सॉफ्टवेअरची एकूण क्वालिटी सुधारते आणि डेव्हलपमेंट प्रक्रियेच्या पुढील टप्प्यात लागणाऱ्या महागड्या दुरुस्त्या टाळता येतात.

इन्स्पेक्शन टेस्टिंग प्रोसेस (Inspection Testing Process):

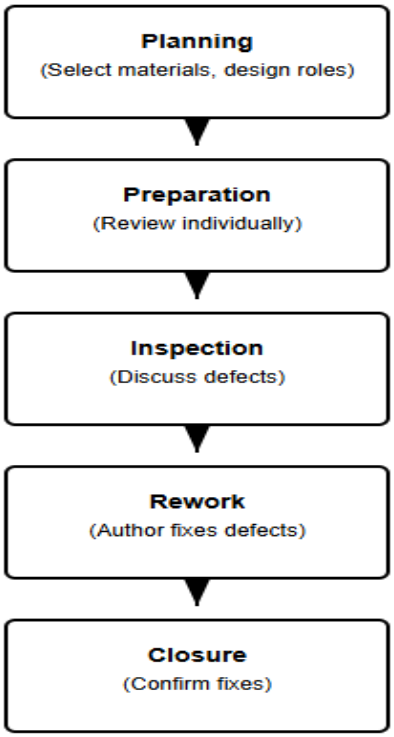

Fig 1.5: इन्स्पेक्शन टेस्टिंग प्रोसेस (Inspection Testing Process)

डायग्राम इन्स्पेक्शन प्रोसेस एक स्पष्ट आणि क्रमवार (sequential) वर्कफ्लो म्हणून दर्शवतो:

1. **प्लॅनिंग (Planning):** हा प्रारंभिक टप्पा आहे, ज्यामध्ये इन्स्पेक्शनसाठी आवश्यक असलेले मटेरियल निवडले जाते आणि टीम मेंबर्सना भूमिका (roles) दिल्या जातात. योग्य प्लॅनिंगमुळे इन्स्पेक्शन प्रक्रिया सुव्यवस्थित होते.
2. **प्रिपरेशन (Preparation):** या टप्प्यात प्रत्येक रिव्ह्यूअर स्वतंत्रपणे मटेरियलचा अभ्यास करून संभाव्य डिफेक्ट्स किंवा इश्यूज ओळखतो, जे ग्रुप मीटिंगपूर्वी केले जाते.
3. **इन्स्पेक्शन (Inspection):** या फेजमध्ये टीम एकत्र येऊन तयारीदरम्यान सापडलेल्या डिफेक्ट्सवर चर्चा करते. येथे मुख्य भर इश्यूज लॉग करणे आणि त्यांचे सहकार्याने समजून घेणे यावर असतो.
4. **रीवर्क (Rework):** इन्स्पेक्शन मीटिंगनंतर ऑथर किंवा जबाबदार व्यक्ती ओळखलेले डिफेक्ट्स दुरुस्त करते, ज्यामुळे सॉफ्टवेअर आर्टिफॅक्टची क्वालिटी सुधारते.
5. **क्लोजर (Closure):** हा अंतिम टप्पा आहे, ज्यामध्ये सर्व डिफेक्ट्स दुरुस्त झाले आहेत का आणि केलेल्या फिक्सेस समाधानकारक आहेत का याची खात्री केली जाते. त्यानंतर इन्स्पेक्शन सायकल अधिकृतरीत्या पूर्ण (close) केली जाते.

1.5.2 वॉकथ्रूज (Walkthroughs)

वॉकथ्रू ही एक अर्ध-औपचारिक (semi-formal) रिव्ह्यू प्रक्रिया आहे, ज्यामध्ये सॉफ्टवेअर आर्टिफॅक्टचा ऑथर (जसे की सोर्स कोड, डिझाईन डॉक्युमेंट्स किंवा रिक्वायरमेंट्स) टीम मेंबर्सना त्या मटेरियलमधून स्टेप-बाय-स्टेप मार्गदर्शन करतो. या प्रक्रियेचा उद्देश सिस्टीमचे लॉजिक समजावून सांगणे आणि टीमकडून फीडबॅक गोळा करणे हा असतो. इन्स्पेक्शन्स च्या तुलनेत, वॉकथ्रूज अधिक कमी औपचारिक असतात आणि त्यांचा भर फक्त डिफेक्ट्स शोधण्यावर नसून समज वाढवणे, स्पष्टीकरण देणे आणि आर्टिफॅक्ट सुधारणे यावर असतो. वॉकथ्रू दरम्यान सहभागी व्यक्ती कंटेंटवर चर्चा करतात, सूचना देतात आणि संभाव्य चुका दाखवतात. ही प्रक्रिया नॉलेज शेअरिंगला प्रोत्साहन देते, रिक्वायरमेंट्स स्पष्ट करते आणि सुरुवातीच्या टप्प्यातच स्पष्ट चुका ओळखण्यास मदत करते.

वॉकथ्रूजमध्ये सामान्यतः खालील भूमिका असतात:

- ऑथर – सत्राचे नेतृत्व करतो आणि मटेरियल समजावून सांगतो.
- पार्टिसिपंट्स – प्रश्न विचारतात, सूचना देतात आणि फीडबॅक प्रदान करतात.

वॉकथ्रूज टेस्टिंग प्रोसेस (Walkthroughs Testing Process)

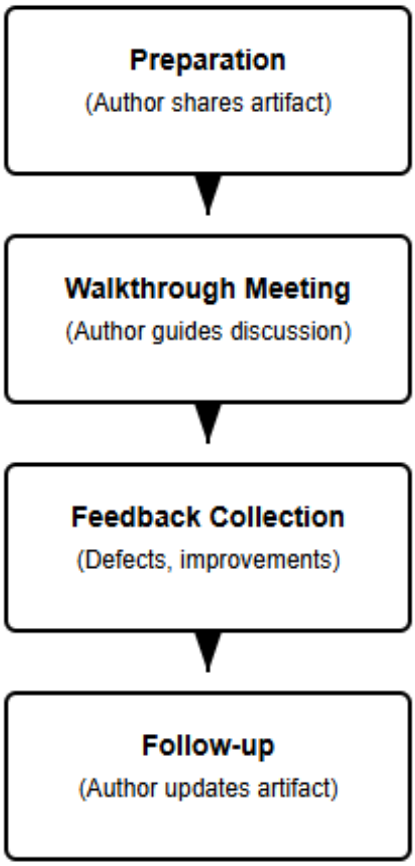

Fig 1.6: वॉकथ्रूज टेस्टिंग प्रोसेस (Walkthroughs Testing Process)

वरील आकृती वॉकथ्रू प्रोसेस चार क्रमवार (sequential) टप्प्यांमध्ये दर्शवते:

1. **प्रिपरेशन (Preparation):** मीटिंगपूर्वी ऑथर सॉफ्टवेअर आर्टिफॅक्ट पार्टिसिपंट्सना शेअर करतो. *("*Author shares artifact")
2. **वॉकथ्रू मीटिंग (Walkthrough Meeting):** ऑथर चर्चा नेतृत्व करतो आणि आर्टिफॅक्ट समजावून सांगतो, तर पार्टिसिपंट्स त्याचे अनुसरण करतात. *("*Author guides discussion")
3. **फीडबॅक कलेक्शन (Feedback Collection):** पार्टिसिपंट्स डिफेक्ट्स ओळखतात आणि सुधारणांसाठी सूचना देतात. *(*"Defects, improvements")
4. **फॉलो-अप (Follow-up):** मिळालेल्या फीडबॅकनुसार ऑथर आर्टिफॅक्ट अपडेट करतो. *("*Author updates artifact")

उदाहरण (Example): एखादा डेव्हलपर नवीन मॉड्यूलचे डिझाईन सहकाऱ्यांसमोर सादर करतो, त्याचा वर्कफ्लो समजावून सांगतो आणि सुधारणा सुचवण्यासाठी किंवा विसंगती ओळखण्यासाठी सूचना मागतो.

1.5.3 टेक्निकल रिव्ह्यू (Technical Review)

टेक्निकल रिव्ह्यू ही एक संरचित (structured) मूल्यमापन प्रक्रिया आहे, ज्यामध्ये रिक्वायरमेंट स्पेसिफिकेशन्स, डिझाईन डॉक्युमेंट्स, सोर्स कोड किंवा टेस्ट प्लॅन्स यांसारख्या सॉफ्टवेअर प्रॉडक्ट्सचे परीक्षण पात्र तज्ञांकडून (qualified personnel) केले जाते. या प्रक्रियेचा उद्देश डिफेक्ट्स ओळखणे, स्टँडर्ड्सचे पालन होत आहे का ते तपासणे आणि टेक्निकल अचूकता सुनिश्चित करणे हा आहे. टेक्निकल रिव्ह्यूचा मुख्य हेतू म्हणजे सॉफ्टवेअर प्रॉडक्ट त्याच्या अपेक्षित रिक्वायरमेंट्स पूर्ण करते का आणि पुढील डेव्हलपमेंट फेजकडे जाण्यापूर्वी मान्य टेक्निकल गाईडलाईन्सचे पालन करते का हे निश्चित करणे. टेक्निकल रिव्ह्यूमध्ये प्रॉडक्टची टेक्निकल साउंडनेस (technical soundness) तपासली जाते. यामुळे संभाव्य समस्या लवकर टप्प्यातच ओळखता येतात, परिणामी नंतर लागणाऱ्या दुरुस्तीचा खर्च कमी होतो. ही प्रक्रिया वॉकथ्रूपेक्षा अधिक औपचारिक असते, परंतु इन्स्पेक्शनइतकी कठोर (rigid) नसू शकते.

टेक्निकल रिव्ह्यूचे उद्देश (Purpose of Technical Review):

1. टेस्टिंगपूर्वी सॉफ्टवेअर आर्टिफॅक्ट्समधील डिफेक्ट्स ओळखणे.
2. प्रोजेक्ट आणि इंडस्ट्री स्टँडर्ड्सचे पालन होत आहे का हे सुनिश्चित करणे.
3. सॉफ्टवेअरची टेक्निकल फिजिबिलिटी आणि क्वालिटी तपासणे.
4. सिस्टीमची मेंटेनेबिलिटी आणि रिलायबिलिटी सुधारण्यास मदत करणे.

टेक्निकल रिव्ह्यूचे प्रकार (Types of Technical Reviews):

1. **फॉर्मल टेक्निकल रिव्ह्यू (Formal Technical Review – FTR)**
   - डॉक्युमेंटेड प्रोसेस नुसार आयोजित केला जातो.
   - मॉडरेटर, ऑथर आणि रिव्ह्यूअर्स अशा पूर्वनिश्चित भूमिका असतात.
   - टेक्निकल पैलूंचे संपूर्ण कव्हरेज सुनिश्चित करण्यासाठी स्ट्रक्चर्ड चेकलिस्ट्स वापरल्या जातात.
   - फॉर्मल रिव्ह्यू रेकॉर्ड्स आणि फॉलो-अप ॲक्शन रिपोर्ट्स तयार केले जातात.
2. **इन्फॉर्मल टेक्निकल रिव्ह्यू (Informal Technical Review)**
   - कमी संरचित (less structured) असतो; डॉक्युमेंटेड प्रोसीजरची गरज नसते.
   - जलद फीडबॅक आणि ब्रेनस्टॉर्मिंग साठी आयोजित केला जातो.
   - सुरुवातीच्या डिझाईन संकल्पना, प्रोटोटाइप्स किंवा कोड स्निपेट्स साठी उपयुक्त असतो.

टेक्निकल रिव्ह्यू प्रोसेस (Technical Review Process)

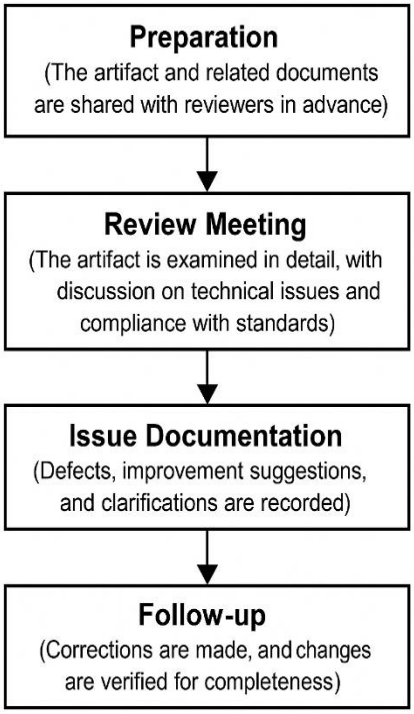

Fig 1.7: टेक्निकल रिव्ह्यू प्रोसेस (Technical Review Process)

प्रोसेस स्टेप्स (Process Steps)

1. **प्रिपरेशन (Preparation):** इन्स्पेक्शन/रिव्ह्यूसाठी आवश्यक असलेले आर्टिफॅक्ट आणि संबंधित डॉक्युमेंट्स आधीच रिव्ह्यूअर्सना शेअर केले जातात.
2. **रिव्ह्यू मीटिंग (Review Meeting):** आर्टिफॅक्टचे सविस्तर परीक्षण केले जाते आणि टेक्निकल इश्यूज तसेच स्टँडर्ड्सचे पालन यावर चर्चा केली जाते.
3. **इश्यू डॉक्युमेंटेशन (Issue Documentation):** ओळखलेले डिफेक्ट्स, सुधारणा सुचना आणि आवश्यक स्पष्टीकरणे नोंदवली जातात.
4. **फॉलो-अप (Follow-up):** आवश्यक दुरुस्त्या केल्या जातात आणि केलेले बदल पूर्ण व योग्य आहेत का ते तपासले जाते.

उदाहरण (Example): औद्योगिक नियंत्रण प्रणाली (Industrial Control System) च्या डिझाईन स्पेसिफिकेशन्स चे सिस्टीम आर्किटेक्ट्सचे पॅनल रिव्ह्यू करते, जेणेकरून डेव्हलपमेंट सुरू होण्यापूर्वी सिक्युरिटी, परफॉर्मन्स आणि स्केलेबिलिटी रिक्वायरमेंट्स पूर्ण होत आहेत याची खात्री करता येईल.

1.5.4 फंक्शनल टेस्टिंग (Functional Testing)

फंक्शनल टेस्टिंग ही सॉफ्टवेअर टेस्टिंगची एक पद्धत आहे, जी सिस्टीम किंवा तिचे घटक रिक्वायरमेंट्समध्ये नमूद केलेली कार्ये (functions) योग्य प्रकारे पार पाडतात का हे तपासते. या टेस्टिंगमध्ये सॉफ्टवेअरचे वर्तन फंक्शनल स्पेसिफिकेशन्सशी तुलना करून तपासले जाते, परंतु अंतर्गत कोड स्ट्रक्चरचा विचार केला जात नाही. फंक्शनल टेस्टिंगमध्ये सिस्टीम काय करते (what) यावर लक्ष दिले जाते, कसे करते (how) यावर नाही. ही टेस्टिंग सहसा ब्लॅक-बॉक्स टेस्टिंग टेक्निक्स वापरून केली जाते.

फंक्शनल टेस्टिंगचे उद्देश (Purpose of Functional Testing)

1. सर्व नमूद केलेल्या फंक्शनल रिक्वायरमेंट्स योग्यरीत्या इम्प्लिमेंट झाल्या आहेत का हे सुनिश्चित करणे.
2. सिस्टीमच्या फीचर्स आणि ऑपरेशन्समधील डिफेक्ट्स शोधणे.
3. युजर इनपुट्स दिल्यावर अपेक्षित आउटपुट्स मिळतात का हे तपासणे.
4. विविध कॉम्पोनंट्समधील योग्य इंटिग्रेशन होत आहे का हे निश्चित करणे.

फंक्शनल टेस्टिंगचे प्रकार (Types of Functional Testing)

1. **युनिट टेस्टिंग (Unit Testing)** – स्वतंत्र मॉड्यूल्सची चाचणी.
   - ड्रायव्हर (Driver) – टेस्ट होणाऱ्या युनिटला कॉल करणारे तात्पुरते मॉड्यूल.
   - स्टब (Stub) – कॉल होणाऱ्या फंक्शनचे सिम्युलेशन करणारे तात्पुरते मॉड्यूल.
2. **इंटिग्रेशन टेस्टिंग (Integration Testing)** – मॉड्यूल्समधील योग्य परस्परसंवाद तपासणे.
   - टॉप-डाऊन, बॉटम-अप, बाय-डायरेक्शनल पद्धती.
3. **सिस्टीम टेस्टिंग (System Testing)** – पूर्णपणे इंटिग्रेट झालेल्या सिस्टीमची चाचणी.
4. **ॲक्सेप्टन्स टेस्टिंग (Acceptance Testing)** – सॉफ्टवेअर रिलीजसाठी तयार आहे का हे तपासणे.
   - अल्फा टेस्टिंग – डेव्हलपर साइटवर.
   - बीटा टेस्टिंग – कस्टमर साइटवर.
5. **स्पेशल टेस्टिंग (Special Testing)** – विशिष्ट पैलूंवर लक्ष केंद्रित करणारी टेस्टिंग:
   - परफॉर्मन्स टेस्टिंग (लोड, स्ट्रेस)
   - रिग्रेशन टेस्टिंग
   - सिक्युरिटी टेस्टिंग
   - क्लायंट-सर्व्हर टेस्टिंग
   - GUI टेस्टिंग
   - डेटाबेस टेस्टिंग
   - सॅनिटी टेस्टिंग
   - स्मोक टेस्टिंग

फंक्शनल टेस्टिंग प्रोसेस (Functional Testing Process)

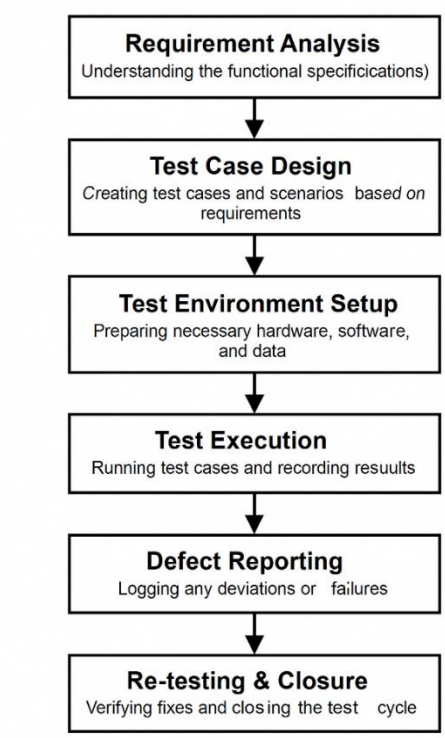

Fig 1.8: फंक्शनल टेस्टिंग प्रोसेस (Functional Testing Process)

1. **रिक्वायरमेंट ॲनालिसिस (Requirement Analysis):** फंक्शनल स्पेसिफिकेशन्स समजून घेणे.
2. **टेस्ट केस डिझाईन (Test Case Design):** रिक्वायरमेंट्सच्या आधारे टेस्ट केसेस आणि टेस्ट सीनारिओज तयार करणे.
3. **टेस्ट एन्व्हायर्नमेंट सेटअप (Test Environment Setup):** आवश्यक हार्डवेअर, सॉफ्टवेअर आणि टेस्ट डेटा तयार करणे.
4. **टेस्ट एक्झिक्यूशन (Test Execution):** टेस्ट केसेस रन करून रिझल्ट्स रेकॉर्ड करणे.
5. **डिफेक्ट रिपोर्टिंग (Defect Reporting):** अपेक्षित व प्रत्यक्ष निकालांमधील विचलन (deviations) किंवा फेल्युअर्स नोंदवणे.
6. **री-टेस्टिंग आणि क्लोजर (Re-testing and Closure):** दुरुस्त्या व्हेरिफाय करणे आणि टेस्ट सायकल क्लोज करणे.

उदाहरण: ऑनलाईन बँकिंग लॉगिन फंक्शन ची टेस्टिंग करून योग्य क्रेडेन्शियल्स दिल्यावर ॲक्सेस मिळतो का, चुकीच्या क्रेडेन्शियल्सवर एरर मेसेज दिसतो का, तसेच अकाउंट लॉकआउट सारखी सिक्युरिटी मेकॅनिझम्स योग्यरित्या काम करतात का हे तपासणे.

1.5.5 कोड कव्हरेज टेस्टिंग (Code Coverage Testing)

कोड कव्हरेज टेस्टिंग ही एक सिस्टिमॅटिक प्रक्रिया आहे, जी एखाद्या ठरावीक टेस्ट केसेसच्या सेटमुळे प्रोग्रामचा सोर्स कोड किती प्रमाणात एक्झिक्यूट झाला आहे हे मोजते. यामुळे टेस्टिंगचे क्वांटिटेटिव्ह (quantitative) मूल्यांकन मिळते, कोडमधील न-टेस्ट झालेले भाग ओळखता येतात आणि टेस्टिंगची पूर्णता (completeness) वाढवता येते.

सामान्य सूत्र (General Formula):

$$\text{Coverage}(\%) = \frac{Number\ of\ items\ executed\ during\ testing}{Total\ Number\ of\ items\ in\ the\ program} \times 100$$

येथे आयटम्स पुढीलपैकी असू शकतात:

- स्टेटमेंट्स (Statement Coverage)
- ब्रँचेस / डिसिजन पॉइंट्स (Branch Coverage)
- पाथ्स (Path Coverage)

कोड कव्हरेजचे प्रकार (Types of Code Coverage)

1. स्टेटमेंट कव्हरेज (Statement Coverage) – कोडमधील प्रत्येक एक्झिक्यूटेबल स्टेटमेंट किमान एकदा तरी रन होते याची खात्री करते.
2. ब्रँच कव्हरेज (Branch Coverage) – प्रत्येक डिसिजन पॉइंटचे true/false दोन्ही ब्रँचेस एक्झिक्यूट झाल्या आहेत का ते तपासते.
3. कंडिशन कव्हरेज (Condition Coverage) – प्रत्येक Boolean सब-एक्सप्रेशन true आणि false दोन्ही व्हॅल्यूजसाठी एक्झिक्यूट होते का ते सुनिश्चित करते.
4. पाथ कव्हरेज (Path Coverage) – प्रोग्राममधील सर्व शक्य कंट्रोल फ्लो पाथ्स एक्झिक्यूट झाले आहेत का ते तपासते.

कोड कव्हरेज प्रोसेस (Code Coverage Process)

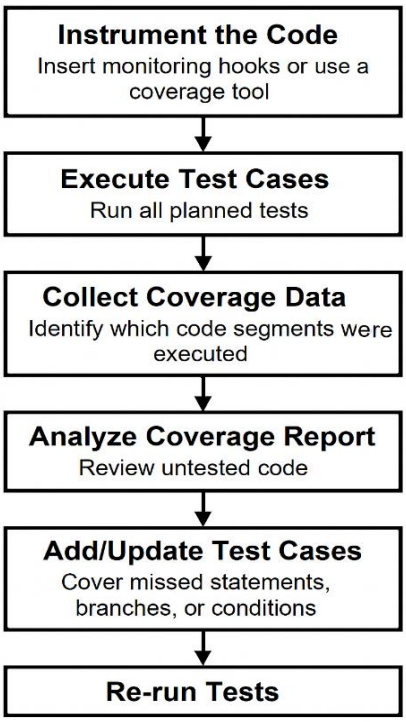

Figure 1.9: कोड कव्हरेज प्रोसेस (Code Coverage Process)

1. **कोड इन्स्ट्रुमेंटेशन (Instrument the Code):** एक्झिक्यूशन ट्रॅक करण्यासाठी मॉनिटरिंग टूल्स वापरणे किंवा कोडमध्ये आवश्यक स्टेटमेंट्स इन्सर्ट करणे.
2. **टेस्ट केसेस एक्झिक्यूट करणे (Execute Test Cases):** तयार केलेल्या टेस्ट केसेसचा सेट रन करणे.
3. **कव्हरेज डेटा कलेक्ट करणे (Collect Coverage Data):** कोणते स्टेटमेंट्स, ब्रँचेस आणि कंडिशन्स एक्झिक्यूट झाले याची माहिती गोळा करणे.
4. **कव्हरेज रिपोर्ट ॲनालिसिस (Analyze Coverage Report):** टेस्टिंगदरम्यान कव्हर न झालेले कोडचे भाग ओळखणे.
5. **टेस्ट केसेस सुधारणा (Enhance Test Cases):** कव्हरेज वाढवण्यासाठी नवीन टेस्ट केसेस ॲड करणे किंवा विद्यमान टेस्ट केसेस सुधारित करणे.

उदाहरण (Example): जर एखाद्या सॉफ्टवेअर मॉड्यूलमध्ये 200 एक्झिक्यूटेबल स्टेटमेंट्स असतील आणि टेस्ट सूटने त्यापैकी 160 स्टेटमेंट्स एक्झिक्यूट केली असतील, तर:

$$\text{Coverage}(\%) = \frac{Number\ of\ Statement\ executed\ during\ testing}{Total\ Number\ of\ Statement\ in\ the\ program} \times 100$$

$$\text{Statement Coverage (\%)} = (160 / 200) \times 100 = 80\%$$

म्हणजेच, या मॉड्यूलसाठी स्टेटमेंट कव्हरेज 80% आहे.

1.5.6 कोड कॉम्प्लेक्सिटी टेस्टिंग (Code Complexity Testing)

कोड कॉम्प्लेक्सिटी टेस्टिंग ही सॉफ्टवेअर टेस्टिंग आणि ॲनालिसिसची एक पद्धत आहे, जी प्रोग्रामच्या सोर्स कोडमधील स्ट्रक्चरल आणि लॉजिकल गुंतागुंत (complexity) मोजते. या टेस्टिंगचा उद्देश सॉफ्टवेअरची मेंटेनेबिलिटी, रिलायबिलिटी आणि डिफेक्ट-प्रोननेस (दोष होण्याची शक्यता) यांचे मूल्यांकन करणे हा आहे. या पद्धतीमध्ये सायक्लोमॅटिक कॉम्प्लेक्सिटी, हॉलस्टेड मेट्रिक्स आणि नेस्टिंग डेप्थ (Cyclomatic Complexity, Halstead Metrics and Nesting Depth) यांसारख्या सॉफ्टवेअर मेट्रिक्स चा वापर केला जातो, ज्यामुळे असे कोड सेगमेंट्स ओळखता येतात ज्यांना सोपे करणे (simplification) किंवा रिफॅक्टरिंग करण्याची गरज असते. कोड कॉम्प्लेक्सिटी टेस्टिंगचा मुख्य हेतू म्हणजे सॉफ्टवेअर मॉड्यूल्स साधे, समजण्यास सोपे आणि मेंटेन करायला सुलभ ठेवणे, तसेच असे कोड भाग ओळखणे जे भविष्यात डिफेक्ट्सचा धोका वाढवू शकतात.

कोड कॉम्प्लेक्सिटी प्रोसेस (Code Complexity Process)

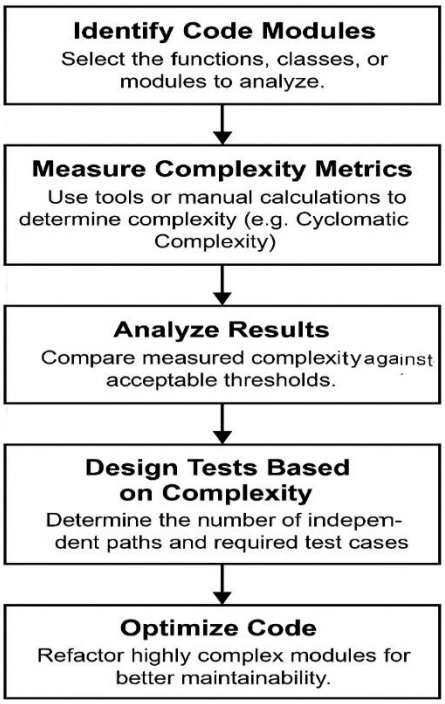

Fig 1.10: कोड कॉम्प्लेक्सिटी प्रोसेस (Code Complexity Process)

1. **कोड मॉड्यूल्स ओळखणे (Identify Code Modules):** ॲनालिसिससाठी आवश्यक असलेली फंक्शन्स, क्लासेस किंवा मॉड्यूल्स निवडणे.
2. **कॉम्प्लेक्सिटी मेट्रिक्स मोजणे (Measure Complexity Metrics):** टूल्स किंवा मॅन्युअल कॅल्क्युलेशन्स वापरून कॉम्प्लेक्सिटी मोजणे. (उदा. Cyclomatic Complexity).
3. **रिझल्ट्सचे विश्लेषण (Analyze Results):** मोजलेली कॉम्प्लेक्सिटी स्वीकार्य मर्यादांशी (acceptable thresholds) तुलना करणे.
4. **कॉम्प्लेक्सिटीवर आधारित टेस्ट डिझाईन (Design Tests Based on Complexity):** इंडिपेंडंट पाथ्सची संख्या आणि आवश्यक टेस्ट केसेस निश्चित करणे.
5. **कोड ऑप्टिमायझेशन (Optimize Code):** जास्त कॉम्प्लेक्स असलेली मॉड्यूल्स रिफॅक्टर करून मेंटेन करणे सुलभ करणे.

सामान्य कॉम्प्लेक्सिटी मेट्रिक्स (Common Complexity Metrics)

1. सायक्लोमॅटिक कॉम्प्लेक्सिटी (Cyclomatic Complexity – McCabe's Metric)
   कोडमधील इंडिपेंडंट एक्झिक्यूशन पाथ्सची संख्या मोजते.
   सूत्र (Formula):

$$V(G) = E - N + 2P$$

जिथे: E = कंट्रोल फ्लो ग्राफमधील एजेसची संख्या, N = नोड्सची संख्या, P = कनेक्टेड कॉम्पोनंट्सची संख्या
मार्गदर्शक तत्त्वे (Guideline):

- 1–10: साधा कोड, कमी धोका
- 11–20: मध्यम कॉम्प्लेक्सिटी
- >20: जास्त कॉम्प्लेक्सिटी आणि धोकादायक

2. हॅलस्टेड मेट्रिक्स (Halstead Metrics)
   ऑपरेटर्स, ऑपरेन्ड्स आणि प्रोग्राम व्होकॅब्युलरी वर आधारित कॉम्प्लेक्सिटी मोजते.
3. नेस्टिंग डेप्थ (Nesting Depth)
   कोडमधील कंट्रोल स्ट्रक्चर्स किती स्तरांपर्यंत नेस्टेड आहेत हे मोजते.

उदाहरण (Example): जर एखाद्या फंक्शनच्या कंट्रोल फ्लो ग्राफ मध्ये E = 25 एजेस, N = 20 नोड्स आणि P = 1 कनेक्टेड कॉम्पोनंट असतील, तर:

$$V(G) = 25 - 20 + 2(1) = 7$$

याचा अर्थ असा की या फंक्शनची कॉम्प्लेक्सिटी कमी आहे आणि टेस्टिंगसाठी कमी प्रयत्न आवश्यक आहेत.

1.6 ब्लॅक बॉक्स टेस्टिंग (Black Box Testing)

ब्लॅक बॉक्स टेस्टिंग ही सॉफ्टवेअर टेस्टिंगची एक पद्धत आहे, ज्यामध्ये टेस्टरला ॲप्लिकेशनची अंतर्गत रचना, सोर्स कोड किंवा इम्प्लिमेंटेशन डिटेल्सची कोणतीही माहिती नसते. या पद्धतीत टेस्टर फक्त इनपुट देतो आणि त्यानुसार मिळणारे आउटपुट तपासतो, जेणेकरून सॉफ्टवेअर स्पेसिफाईड रिक्वायरमेंट्स पूर्ण करते का हे व्हेरिफाय करता येईल. या अप्रोचमध्ये सॉफ्टवेअरला "ब्लॅक बॉक्स" म्हणून पाहिले जाते, म्हणजेच त्याचे अंतर्गत लॉजिक लपलेले असते, आणि फक्त त्याचे बाह्य वर्तन (external behaviour) टेस्ट केले जाते.

उदाहरण (Example) : ATM मधील विथड्रॉ फंक्शन ची टेस्टिंग करताना खालील इनपुट्स देणे :

- किमान मर्यादा – ₹500
- कमाल मर्यादा – ₹10,000
- मर्यादित असलेली रक्कम – उदा. ₹700
- मर्यादेबाहेरील रक्कम – उदा. ₹100, ₹10,500
- अवैध फॉरमॅट – उदा. ₹750 (दशांश/अयोग्य फॉरमॅट)

यामुळे सिस्टीम फक्त वैध इनपुट्स स्वीकारते आणि अवैध इनपुट्स नाकारते का हे तपासता येते, तेही आतील प्रोग्राम कोड तपासल्याशिवाय.

ब्लॅक बॉक्स टेस्टिंगच्या पद्धती (Methods of Black Box Testing)

ब्लॅक बॉक्स टेस्टिंग मेथड्स म्हणजे अशा तंत्रांचा संच, ज्यांचा वापर इनपुट्स आणि आउटपुट्सवर लक्ष केंद्रित करून टेस्ट केसेस डिझाईन करण्यासाठी केला जातो, आणि त्यामध्ये अंतर्गत कोडची कोणतीही माहिती आवश्यक नसते.

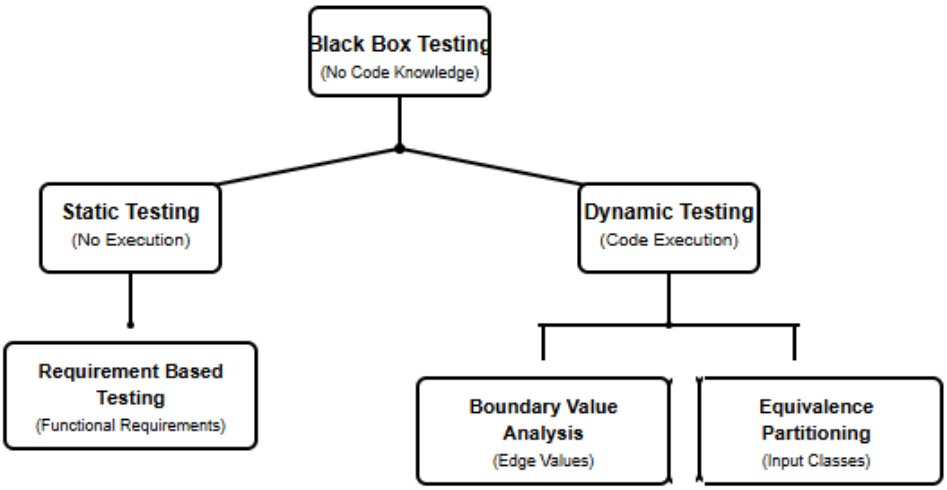

Fig 1.11: ब्लॅक बॉक्स टेस्टिंग मेथड्स (Black Box Testing Methods)

1.6.1 रिक्वायरमेंट-बेस्ड टेस्टिंग (Requirement-Based Testing)

रिक्वायरमेंट-बेस्ड टेस्टिंग ही ब्लॅक बॉक्स टेस्टिंग ची एक पद्धत आहे, ज्यामध्ये सिस्टीमच्या डॉक्युमेंटेड रिक्वायरमेंट्स (उदा. Software Requirement Specification – SRS, युजर स्टोरीज किंवा युज केसेस) यांच्या आधारे टेस्ट केसेस डिझाईन केल्या जातात, जेणेकरून सॉफ्टवेअर नेमके स्पेसिफिकेशननुसार वागत आहे का हे तपासता येईल. या पद्धतीत प्रत्येक रिक्वायरमेंटला टेस्ट करता येण्याजोगी अट (testable condition) म्हणून पाहिले जाते. याचा मुख्य उद्देश म्हणजे 100% रिक्वायरमेंट कव्हरेज सुनिश्चित करणे, म्हणजेच सर्व फंक्शनल आणि नॉन-फंक्शनल रिक्वायरमेंट्स किमान एकदा तरी टेस्ट झाल्या आहेत याची खात्री करणे. रिक्वायरमेंट्स आणि त्यांच्याशी संबंधित टेस्ट केसेस यांचा संबंध दर्शवण्यासाठी सहसा रिक्वायरमेंट ट्रेसेबिलिटी मॅट्रिक्स (RTM) वापरली जाते. ही टेस्टिंग टेक्निक अंतर्गत कोड स्ट्रक्चरपासून स्वतंत्र आहे आणि सिस्टीम टेस्टिंग, ॲक्सेप्टन्स टेस्टिंग तसेच इंटिग्रेशन टेस्टिंग अशा विविध टेस्टिंग लेव्हल्सवर लागू केली जाते.

रिक्वायरमेंट-बेस्ड टेस्टिंग प्रोसेस (Requirement-Based Testing Process)

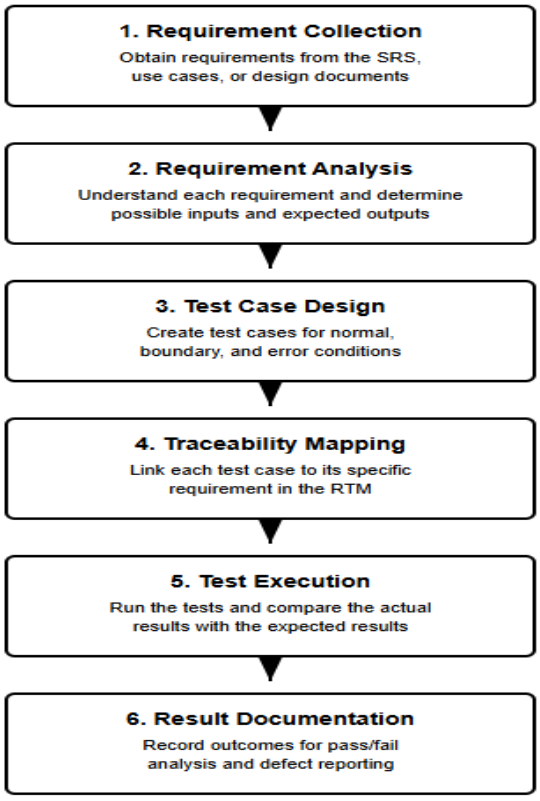

Fig 1.12: रिक्वायरमेंट-बेस्ड टेस्टिंग प्रोसेस (Requirement-Based Testing Process)

1. **रिक्वायरमेंट कलेक्शन (Requirement Collection):** SRS, युज केसेस किंवा डिझाईन डॉक्युमेंट्स मधून रिक्वायरमेंट्स मिळवणे.
2. **रिक्वायरमेंट ॲनालिसिस (Requirement Analysis):** प्रत्येक रिक्वायरमेंट समजून घेणे आणि संभाव्य इनपुट्स व अपेक्षित आउटपुट्स निश्चित करणे.
3. **टेस्ट केस डिझाईन (Test Case Design):** नॉर्मल, बाउंडरी आणि एरर कंडिशन्स साठी टेस्ट केसेस तयार करणे.
4. **ट्रेसेबिलिटी मॅपिंग (Traceability Mapping):** प्रत्येक टेस्ट केसला त्याच्या संबंधित रिक्वायरमेंटशी RTM मध्ये लिंक करणे.
5. **टेस्ट एक्झिक्यूशन (Test Execution):** टेस्ट्स रन करून प्रत्यक्ष निकालांची अपेक्षित निकालांशी तुलना करणे.
6. **रिझल्ट डॉक्युमेंटेशन (Result Documentation):** Pass/Fail ॲनालिसिस आणि डिफेक्ट रिपोर्टिंग साठी निकाल नोंदवणे.

उदाहरण (Example): रिक्वायरमेंट: ATM ने फक्त तेव्हाच रोख रक्कम द्यावी, जेव्हा खात्यातील शिल्लक रक्कम मागितलेल्या विथड्रॉ रकमेपेक्षा जास्त किंवा समान असेल.

टेस्ट केसेस:

- पॉझिटिव्ह टेस्ट: बॅलन्स ₹10,000, विथड्रॉ ₹5,000 → कॅश डिस्पेन्स होईल
- बाउंडरी टेस्ट: बॅलन्स ₹5,000, विथड्रॉ ₹5,000 → कॅश डिस्पेन्स होईल
- नेगेटिव्ह टेस्ट: बॅलन्स ₹4,000, विथड्रॉ ₹5,000 → ट्रान्झॅक्शन नाकारले जाईल
- इनव्हॅलिड इनपुट टेस्ट: विथड्रॉ रक्कम ₹0 → एरर मेसेज अपेक्षित

1.6.2 बाउंडरी व्हॅल्यू ॲनालिसिस (Boundary Value Analysis – BVA)

बाउंडरी व्हॅल्यू ॲनालिसिस (BVA) ही एक ब्लॅक बॉक्स टेस्टिंग टेक्निक आहे, ज्यामध्ये टेस्ट केसेस इनपुट डोमेनच्या कडील (boundary) व्हॅल्यूज वर आधारित तयार केल्या जातात, कारण बहुतेक चुका या मधल्या व्हॅल्यूजपेक्षा कडील व्हॅल्यूजवर जास्त प्रमाणात आढळतात. BVA मागील मुख्य तत्त्व असे आहे की सॉफ्टवेअर डिफेक्ट्स सहसा किमान (minimum) किंवा कमाल (maximum) इनपुट व्हॅल्यूज हाताळताना उद्भवतात.

सर्व इनपुट्स टेस्ट करण्याऐवजी, BVA मध्ये खालील व्हॅल्यूजवर लक्ष केंद्रित केले जाते:

- किमान (Minimum)
- किमानपेक्षा थोडी कमी (Just below minimum)
- किमानपेक्षा थोडी जास्त (Just above minimum)
- कमालपेक्षा थोडी कमी (Just below maximum)
- कमालपेक्षा थोडी जास्त (Just above maximum)

ही पद्धत विशेषतः पुढील बाबींमध्ये प्रभावी ठरते: न्यूमेरिक रेंजेस, डेट रेंजेस आणि ऑर्डर्ड सिक्वेन्सेस.

बाउंडरी व्हॅल्यू ॲनालिसिस प्रोसेस (Boundary Value Analysis Process):

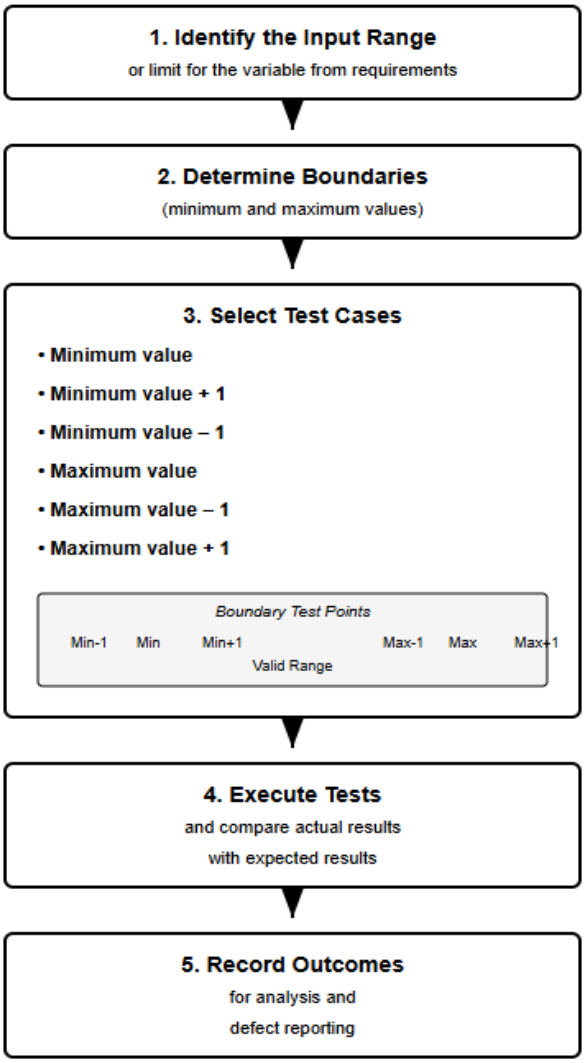

Fig 1.13: बाउंडरी व्हॅल्यू ॲनालिसिस प्रोसेस (Boundary Value Analysis (BVA) Process)

1. **इनपुट रेंज ओळखणे (Identify Input Range):** रिक्वायरमेंट्समधून संबंधित व्हेरिएबलसाठी इनपुटची मर्यादा ओळखणे.
2. **बाउंडरीज निश्चित करणे (Determine Boundaries):** किमान (minimum) आणि कमाल (maximum) व्हॅल्यूज ठरवणे.
3. **टेस्ट केसेस निवडणे (Select Test Cases)**
   - किमान – 1
   - किमान
   - किमान + 1
   - कमाल – 1
   - कमाल
   - कमाल + 1
4. **टेस्ट एक्झिक्यूशन (Execute Tests):** टेस्ट केसेस रन करून प्रत्यक्ष निकालांची अपेक्षित निकालांशी तुलना करणे.
5. **रिझल्ट्स रेकॉर्ड करणे (Record Outcomes):** ॲनालिसिस आणि डिफेक्ट रिपोर्टिंग साठी निकाल नोंदवणे.

उदाहरण: रिक्वायरमेंट: पासवर्डची लांबी 8 ते 12 कॅरेक्टर्स दरम्यान असली पाहिजे.
टेस्ट केसेस:

- किमान − 1: 7 कॅरेक्टर्स → एरर अपेक्षित
- किमान: 8 कॅरेक्टर्स → यशस्वी (Success)
- किमान + 1: 9 कॅरेक्टर्स → यशस्वी
- कमाल − 1: 11 कॅरेक्टर्स → यशस्वी
- कमाल: 12 कॅरेक्टर्स → यशस्वी
- कमाल + 1: 13 कॅरेक्टर्स → एरर अपेक्षित

1.6.3 इक्विव्हॅलन्स पार्टिशनिंग (Equivalence Partitioning – EP)

इक्विव्हॅलन्स पार्टिशनिंग (EP) ही एक ब्लॅक बॉक्स टेस्टिंग टेक्निक आहे, ज्यामध्ये प्रोग्रामच्या इनपुट डोमेनला विविध पार्टिशन्स (क्लासेस) मध्ये विभागले जाते. प्रत्येक पार्टिशनमधील सर्व इनपुट व्हॅल्यूज समान प्रकारे वागतील अशी अपेक्षा असते, म्हणून प्रत्येक पार्टिशनमधून फक्त एक प्रतिनिधी टेस्ट केस एक्झिक्यूट केला जातो. या तंत्राचा मुख्य सिद्धांत असा आहे की, जर एखाद्या पार्टिशनमधील एक टेस्ट केस योग्यरित्या काम करत असेल, तर त्याच पार्टिशनमधील इतर सर्व व्हॅल्यूज देखील तशीच वागतील असे गृहीत धरले जाते. व्हॅलिड आणि इनव्हॅलिड पार्टिशन्स मध्ये इनपुट डेटा गटबद्ध केल्यामुळे, टेस्टर्स टेस्ट केसेसची संख्या कमी करू शकतात आणि तरीही चांगले कव्हरेज साध्य करू शकतात.
ही टेक्निक विशेषतः पुढील बाबींसाठी उपयुक्त आहे:

- न्यूमेरिक रेंजेस
- कॅटेगॉरिकल इनपुट्स
- डेटा फॉरमॅट्स (उदा. ई-मेल, फोन नंबर)

इक्विव्हॅलन्स पार्टिशनिंग प्रोसेस (Equivalence Partitioning Process):

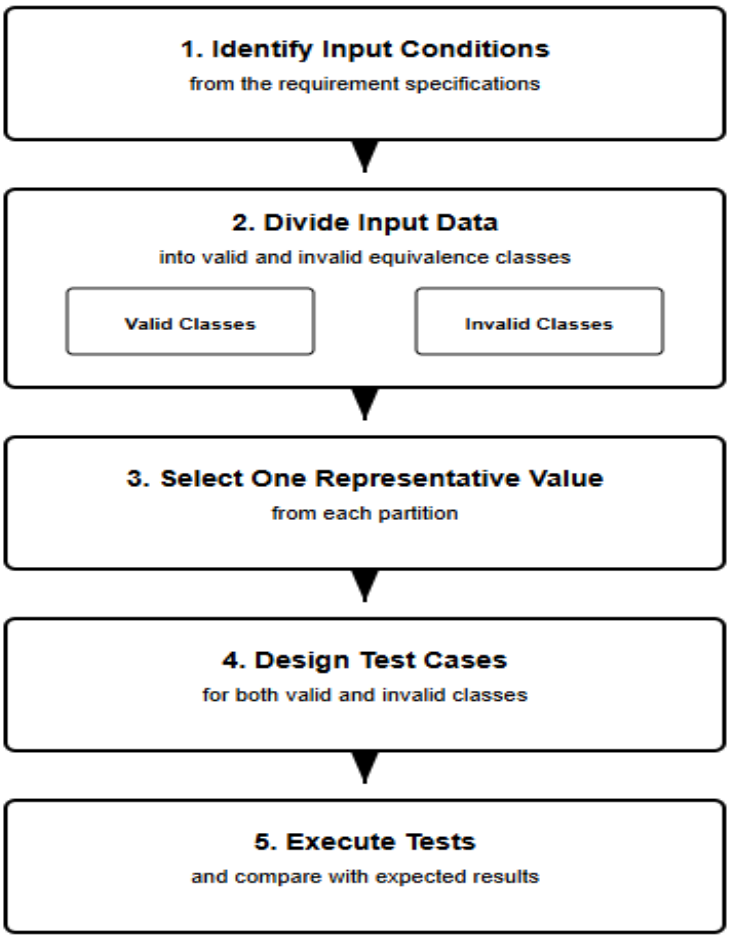

Fig 1.14: इक्विव्हॅलन्स पार्टिशनिंग प्रोसेस (Equivalence Partitioning (EP) Process)

इक्विव्हॅलन्स पार्टिशनिंग प्रोसेस (Equivalence Partitioning Process)

1. इनपुट कंडिशन्स ओळखणे (Identify Input Conditions): रिक्वायरमेंट स्पेसिफिकेशन्समधून इनपुट कंडिशन्स ओळखणे.
2. इक्विव्हॅलन्स क्लासेसमध्ये विभागणी (Divide into Equivalence Classes): इनपुट डेटा व्हॅलिड आणि इनव्हॅलिड इक्विव्हॅलन्स पार्टिशन्समध्ये विभागणे.
3. प्रत्येक पार्टिशनमधून प्रतिनिधी व्हॅल्यू निवडणे (Select Representative Value): प्रत्येक पार्टिशनमधून एक प्रतिनिधी इनपुट व्हॅल्यू निवडणे.
4. टेस्ट केसेस डिझाईन करणे (Design Test Cases): व्हॅलिड आणि इनव्हॅलिड दोन्ही क्लासेससाठी टेस्ट केसेस तयार करणे.
5. टेस्ट एक्झिक्यूशन (Execute Tests): टेस्ट्स रन करून प्रत्यक्ष निकाल अपेक्षित निकालांशी तुलना करणे.

उदाहरण: रिक्वायरमेंट: Age इनपुट फील्ड 18 ते 60 वर्षे (inclusive) स्वीकारते.

पार्टिशन्स:

- व्हॅलिड पार्टिशन: $18 \leq age \leq 60$ → टेस्ट व्हॅल्यू: 30
- इनव्हॅलिड पार्टिशन 1 : $age < 18$ → टेस्ट व्हॅल्यू : 15
- इनव्हॅलिड पार्टिशन 2 : $age > 60$ → टेस्ट व्हॅल्यू : 65

टेस्ट केसेस:

- इनपुट: 30 → Accepted (व्हॅलिड)
- इनपुट: 15 → Rejected (इनव्हॅलिड)
- इनपुट: 65 → Rejected (इनव्हॅलिड)

Table 1.2: व्हाइट बॉक्स टेस्टिंग वि. ब्लॅक बॉक्स टेस्टिंग (Comparison between White Box Testing and Black Box Testing)

निकष (Criteria)	व्हाईट बॉक्स टेस्टिंग (White Box Testing)	ब्लॅक बॉक्स टेस्टिंग (Black Box Testing)
व्याख्या (Definition)	कोडच्या अंतर्गत लॉजिक आणि स्ट्रक्चरवर आधारित टेस्टिंग.	अंतर्गत कोडची माहिती नसताना, फक्त फंक्शनॅलिटीवर आधारित टेस्टिंग.
डायग्राम (Diagram)	कंट्रोल फ्लो ग्राफवर आधारित	इनपुट–आउटपुट बिहेवियरवर आधारित
आवश्यक ज्ञान (Knowledge Required)	प्रोग्रामिंग ज्ञान आवश्यक असते.	प्रोग्रामिंग ज्ञान आवश्यक नाही.
कोड ॲक्सेस (Access to Code)	सोर्स कोडचा ॲक्सेस आवश्यक असतो.	सोर्स कोडचा ॲक्सेस आवश्यक नाही.
मुख्य फोकस (Main Focus)	कोड कव्हरेज – स्टेटमेंट्स, ब्रँचेस, पाथ्स, लूप्स.	रिक्वायरमेंट कव्हरेज – इनपुट्स आणि आउटपुट्स.
कोण करतो (Performed By)	प्रामुख्याने डेव्हलपर्स.	प्रामुख्याने टेस्टर किंवा QA टीम.
वापरली जाणारी तंत्रे (Techniques Used)	स्टेटमेंट कव्हरेज, ब्रँच कव्हरेज, पाथ कव्हरेज, लूप टेस्टिंग.	रिक्वायरमेंट-बेस्ड टेस्टिंग, बाउंडरी व्हॅल्यू ॲनालिसिस, इक्विव्हॅलन्स पार्टिशनिंग, डिसिजन टेबल टेस्टिंग.
उद्दिष्ट (Goal)	कोडची इम्प्लिमेंटेशन योग्य आहे का हे तपासणे.	सॉफ्टवेअर युजर रिक्वायरमेंट्स पूर्ण करते का हे तपासणे.
सापडणाऱ्या चुका (Error Types Found)	लॉजिकल एरर्स, कोडिंग मिस्टेक्स, डेड कोड.	मिसिंग फंक्शनॅलिटी, चुकीचे आउटपुट, युजर इंटरफेस डिफेक्ट्स.
टेस्टिंग लेव्हल (Testing Level)	प्रामुख्याने युनिट टेस्टिंग आणि इंटिग्रेशन टेस्टिंग.	प्रामुख्याने सिस्टीम टेस्टिंग आणि ॲक्सेप्टन्स टेस्टिंग.
उदाहरण (Example)	सॉर्टिंग अल्गोरिदममधील प्रत्येक ब्रँच आणि लूप एक्झिक्यूट होतो का ते तपासणे.	सॉर्टिंग अल्गोरिदमला इनपुट डेटा देऊन आउटपुट योग्यरीत्या सॉर्ट झाले आहे का ते तपासणे.

References:

1. Desikan, S., & Ramesh, G. (2016). *Software testing: Principles and practices*. Pearson India. ISBN: 9788177581218
2. Limaye, M. G. (2012). *Software testing: Principles, techniques and tools*. Tata McGraw Hill Education. ISBN: 9780070139909
3. Chauhan, N. (2016). *Software testing: Principles and practices*. Oxford University Press. ISBN: 9780198061847
4. Rahman, K. (2019). *Science of Selenium: Master Web UI automation and create your own test automation framework*. BPB Publications. ISBN: 9789389423242, 9389423244
5. Singh, Y. (2012). *Software testing*. Cambridge University Press. ISBN: 9781107652781
6. Infosys Springboard – Software Testing Fundamentals Course: https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_013841792861315072425 4_shared/overview
7. W3Schools – Software Testing Tutorials: https://www.w3schools.in/software-testing/tutorials/
8. NPTEL – Software Testing Course: https://nptel.ac.in/courses/106101163

युनिट-2
टाईप्स अॅण्ड लेव्हल्स ऑफ टेस्टिंग
(Types and Levels of Testing)

विषय निष्पत्ती (Course Outcome):

CO2: टेस्टिंगच्या विविध स्तरांसाठी टेस्ट केसेस तयार करा.

घटक निष्पत्ती (Theory Learning Outcome):

1. युनिट चाचणीच्या संकल्पना लागू करा.
2. एकत्रित चाचणीच्या विविध रणनीती स्पष्ट करा.
3. प्रणाली चाचणीची तत्त्वे आणि पद्धती लागू करा.
4. स्वीकार्यता चाचणीचा उद्देश आणि प्रक्रिया स्पष्ट करा.
5. विविध विशेष चाचणी तंत्रे लागू करा.
6. दिलेल्या अनुप्रयोगासाठी टेस्ट केस तयार करा...

2.0 सॉफ्टवेअर टेस्टिंग लेव्हल्स

सॉफ्टवेअर डेव्हलपमेंट प्रक्रियेमध्ये टेस्टिंग विविध स्तरांवर (levels) केली जाते, जेणेकरून प्रत्येक कॉम्पोनंट तसेच संपूर्ण सिस्टीम अपेक्षित क्वालिटी स्टँडर्ड्स पूर्ण करते का हे सुनिश्चित करता येईल. हा लेयर्ड अप्रोच डिफेक्ट्स लवकर ओळखण्यास, चुका दुरुस्त करण्याचा खर्च कमी करण्यास आणि प्रॉडक्टची रिलायबिलिटी वाढवण्यास मदत करतो.
सॉफ्टवेअर टेस्टिंगचे प्रमुख लेव्हल्स पुढीलप्रमाणे आहेत:

- युनिट टेस्टिंग (Unit Testing)
- इंटिग्रेशन टेस्टिंग (Integration Testing)
- सिस्टीम टेस्टिंग (System Testing)
- अॅक्सेप्टन्स टेस्टिंग (Acceptance Testing)

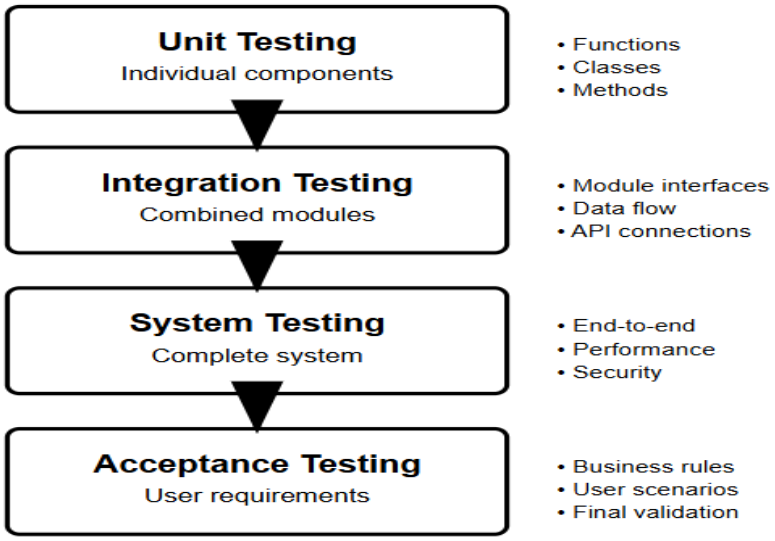

Fig 2.1: सॉफ्टवेअर टेस्टिंग लेव्हल्स (Software Testing Levels)

2.1 युनिट टेस्टिंग: ड्रायव्हर, स्टब (Unit Testing: Driver, Stub)

2.1.1 युनिट टेस्टिंग (Unit Testing)

युनिट टेस्टिंग ही सॉफ्टवेअर टेस्टिंगची एक प्रक्रिया आहे, ज्यामध्ये सॉफ्टवेअर अॅप्लिकेशनचा सर्वात लहान स्वतंत्र भाग (unit) योग्य प्रकारे कार्य करतो का हे तपासले जाते. युनिट म्हणजे फंक्शन, मेथड, प्रोसीजर, क्लास किंवा मॉड्यूल असू शकतो. युनिट टेस्टिंगचा मुख्य उद्देश म्हणजे प्रत्येक भाग इतर सिस्टीमपासून वेगळा ठेवून (isolation मध्ये) तपासणे आणि नंतरच तो मोठ्या सिस्टीममध्ये समाविष्ट करणे. युनिट टेस्टिंग हा सॉफ्टवेअर टेस्टिंगचा मूलभूत (fundamental) स्तर आहे, जो सॉफ्टवेअरमधील सर्वात लहान टेस्ट करता येणाऱ्या घटकांची अचूकता (correctness) पडताळतो. प्रत्येक

युनिटने त्यासाठी ठरवलेली फंक्शनॅलिटी योग्य प्रकारे पार पाडली पाहिजे याची खात्री केली जाते. युनिट टेस्टिंगचा मुख्य हेतू म्हणजे कोडिंग फेजमध्येच डिफेक्ट्स ओळखणे. सुरुवातीच्या टप्प्यात चुका सापडल्यास, त्या दुरुस्त करणे सोपे, कमी खर्चिक आणि कमी वेळखाऊ ठरते. यामुळे पुढील डेव्हलपमेंट टप्प्यांमध्ये होणारे मोठे आणि महागडे डिफेक्ट्स टाळता येतात. युनिट टेस्टिंगचा स्कोप मर्यादित असतो आणि तो फक्त स्वतंत्र युनिट्सच्या टेस्टिंगपर्यंत सीमित असतो. डेटाबेस, फाईल सिस्टीम किंवा API सारख्या एक्सटर्नल डिपेन्डन्सीज सहसा मॉक ऑब्जेक्ट्स किंवा स्टब्स वापरून बदलल्या जातात, जेणेकरून इतर घटकांचा हस्तक्षेप न होता युनिटची चाचणी करता येईल. साधारणपणे युनिट टेस्टिंगची जबाबदारी डेव्हलपर्सकडे असते, कारण त्यांना कोडचे अंतर्गत लॉजिक आणि अपेक्षित वर्तन सर्वाधिक चांगल्या प्रकारे माहिती असते. त्यामुळे ते प्रभावी आणि अर्थपूर्ण टेस्ट केसेस डिझाईन करू शकतात.

युनिट टेस्टिंगसाठी विविध टूल्स आणि फ्रेमवर्क्स उपलब्ध आहेत. त्यामध्ये:

- JUnit – Java ॲप्लिकेशन्ससाठी
- NUnit – .NET ॲप्लिकेशन्ससाठी
- pytest – Python ॲप्लिकेशन्ससाठी

ही टूल्स टेस्ट केस एक्झिक्यूशन, रिझल्ट रिपोर्टिंग आणि ऑटोमेशन यासाठी उपयुक्त सुविधा पुरवतात.

युनिट टेस्टिंगची प्रक्रिया (Process of Unit Testing)

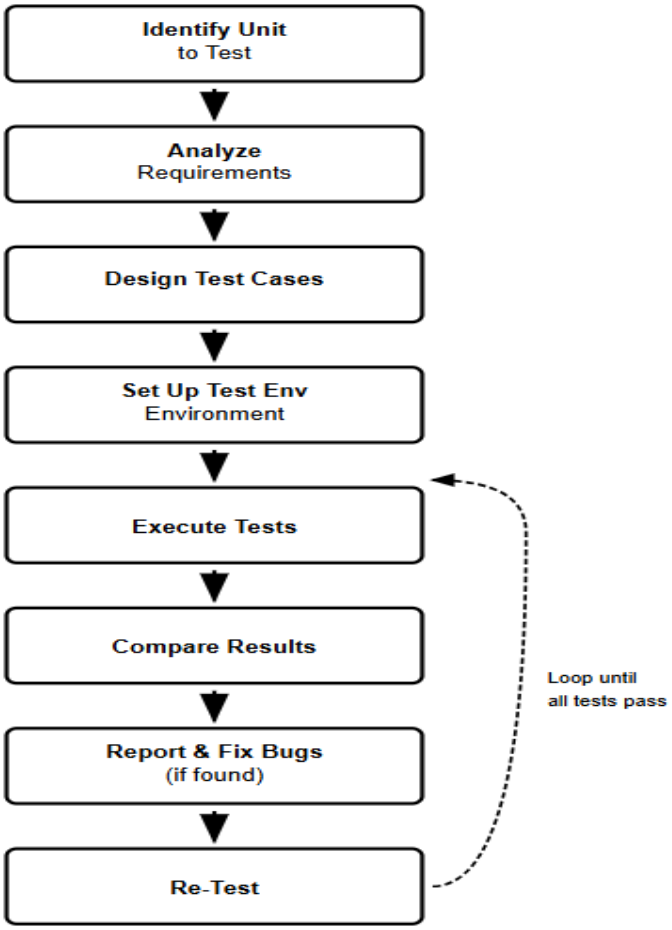

Fig 2.2: युनिट टेस्टिंगची प्रक्रिया (Process of Unit Testing)

1. **टेस्ट करावयाचा युनिट ओळखणे (Identify the Unit to be Tested):** सॉफ्टवेअरमधील सर्वात लहान फंक्शनल कॉम्पोनंट (फंक्शन / मेथड / क्लास) निश्चित करणे.
2. **रिक्वायरमेंट ॲनालिसिस (Analyze Requirements):** त्या युनिटचे अपेक्षित वर्तन (expected behavior) नीट समजून घेणे.
3. **टेस्ट केस डिझाईन (Design Test Cases):** इनपुट्स, अपेक्षित आउटपुट्स आणि एक्झिक्यूशन स्टेप्स निश्चित करणे.

4. **टेस्ट एन्व्हायर्नमेंट सेटअप (Set Up the Test Environment):** आवश्यक टूल्स तयार करणे आणि डिपेन्डन्सीज आयसोलेट करणे. (गरज असल्यास ड्रायव्हर / स्टब वापरणे).
5. **टेस्ट एक्झिक्यूशन (Execute Tests):** डिझाईन केलेल्या टेस्ट केसेस रन करणे.
6. **रिझल्ट्सची तुलना (Compare Results):** प्रत्यक्ष निकाल आणि अपेक्षित निकाल यांची तुलना करणे.
7. **डिफेक्ट रिपोर्टिंग (Report Defects):** फेल झालेल्या टेस्ट केसेससाठी डिफेक्ट्स लॉग आणि डॉक्युमेंट करणे.
8. **री-टेस्टिंग (Re-test after Fixes):** दुरुस्त्या केल्यानंतर पुन्हा टेस्ट रन करणे, जोपर्यंत सर्व टेस्ट केसेस Pass होत नाहीत.

उदाहरण – स्टँडर्ड कॅल्क्युलेटरसाठी युनिट टेस्टिंग

परिस्थिती (Scenario): आपल्याला Standard Calculator ॲप्लिकेशनमधील add() फंक्शन योग्य प्रकारे कार्य करते का हे तपासायचे आहे.

ही फंक्शन:

- दोन संख्यांची बेरीज करेल (पूर्णांक किंवा दशांश)
- निगेटिव्ह नंबर्स आणि शून्य योग्य प्रकारे हाताळेल
- अवैध (invalid) इनपुट दिल्यास एरर देईल

टेस्ट होणारी फंक्शन (Function under Test – FUT)

```
add (a, b):
  return a + b
```

add () फंक्शनसाठी टेस्ट केस डिझाईन (Test Case Design – add () Function)

Test Case ID (टेस्ट केस आयडी)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
TC_ADD_001	Addition of two positive integers दोन सकारात्मक पूर्णांकांची बेरीज	add(5, 3)	8	8	Pass (यशस्वी)
TC_ADD_002	Addition of positive and negative integer सकारात्मक व नकारात्मक पूर्णांकांची बेरीज	add(7, -2)	5	6 (bug) 6 (बग)	Fail (अयशस्वी)

2.1.2 ड्रायव्हर्स आणि स्टब्स (Drivers and Stubs)

युनिट टेस्टिंग किंवा इंटिग्रेशन टेस्टिंग करताना अनेकदा अशी परिस्थिती येते की सिस्टीममधील सर्व मॉड्यूल्स अजून तयार झालेली नसतात, पण तरीही एखादे मॉड्यूल स्वतंत्रपणे (isolation मध्ये) टेस्ट करणे आवश्यक असते. अशा वेळी ड्रायव्हर्स (Drivers) आणि स्टब्स (Stubs) हे तात्पुरते पर्यायी मॉड्यूल्स (temporary stand-ins) म्हणून वापरले जातात.

a. ड्रायव्हर (Driver)

ड्रायव्हर हा एक तात्पुरता प्रोग्राम असतो, जो टेस्ट होणाऱ्या युनिटला (Unit Under Test – UUT) कॉल करतो आणि त्याला आवश्यक इनपुट डेटा पुरवतो. ड्रायव्हर मॉड्यूल टेस्ट होणाऱ्या युनिटचा आउटपुट कॅप्चर करून तो डिस्प्ले किंवा व्हेरिफिकेशनसाठी इंटरप्रेट देखील करू शकतो.

ड्रायव्हरचा मुख्य उद्देश :

- UUT ला आवश्यक इनपुट देणे.
- टेस्टचे एक्झिक्यूशन कंट्रोल करणे.
- तयार होणारा आउटपुट कॅप्चर करून ॲनालिसिस करणे

यामुळे, ज्या वेळी मूळ कॉलिंग प्रोग्राम किंवा हायर-लेव्हल मॉड्यूल उपलब्ध नसते, त्या वेळीही एखादे मॉड्यूल स्वतंत्रपणे टेस्ट करता येते.

ड्रायव्हर कधी वापरला जातो:

- प्रामुख्याने बॉटम-अप इंटिग्रेशन टेस्टिंग मध्ये.
- जेव्हा हायर-लेव्हल मॉड्यूल (calling program) अजून डेव्हलप झालेले नसते

b. स्टब (Stub)

स्टब हा एक डमी (dummy) प्रोग्राम असतो, जो टेस्ट होणाऱ्या युनिटद्वारे कॉल होणाऱ्या लोअर-लेव्हल मॉड्यूलचे वर्तन सिम्युलेट करतो. स्टब प्रत्यक्ष फंक्शनॅलिटी चालवण्याऐवजी पूर्वनिश्चित (predefined) व्हॅल्यूज किंवा सोपे प्रतिसाद (responses) परत करतो.

स्टबचा मुख्य उद्देश:

- अनुपलब्ध असलेल्या लोअर-लेव्हल मॉड्यूलचा पर्याय म्हणून काम करणे.
- टेस्ट होणाऱ्या युनिटला एररशिवाय एक्झिक्यूट होऊ देणे.
- पूर्वानुमान करता येण्याजोगे (predictable) आउटपुट्स मिळवणे

स्टब कधी वापरला जातो: प्रामुख्याने टॉप-डाऊन इंटिग्रेशन टेस्टिंग मध्ये जेव्हा टेस्ट होणारे मॉड्यूल इतर मॉड्यूल्सना कॉल करते, पण ती मॉड्यूल्स अजून तयार नसतात

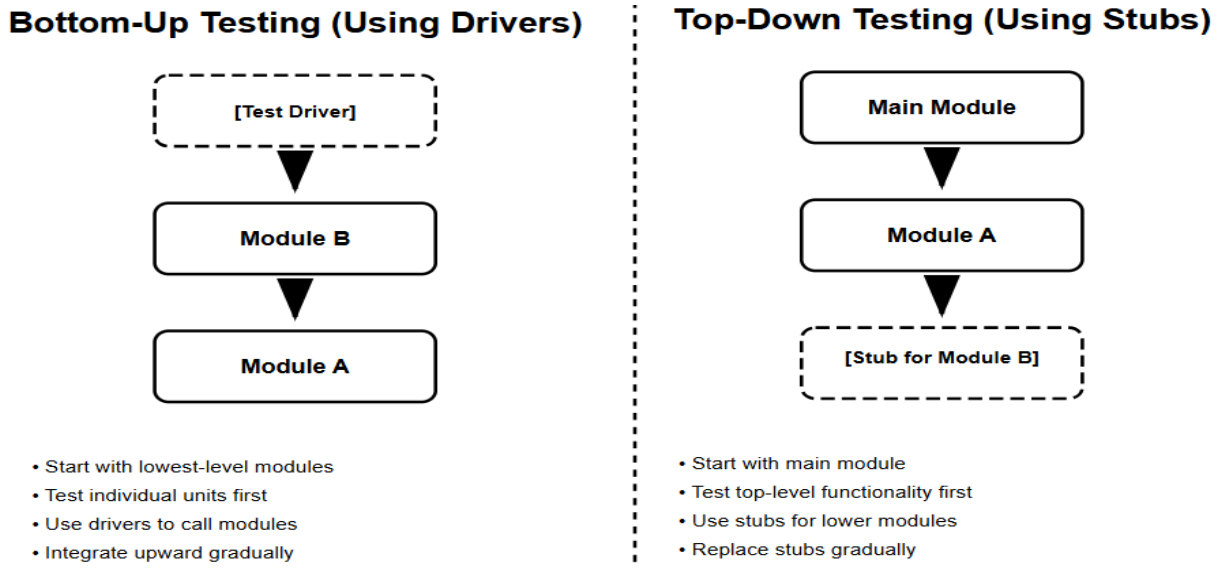

Fig 2.3: ड्रायव्हर्स आणि स्टब्स यांची संकल्पना (Concept of Drivers and Stubs)

बॉटम-अप आणि टॉप-डाऊन टेस्टिंगमध्ये ड्रायव्हर आणि स्टब्सचा वापर:

1. बॉटम-अप टेस्टिंग (Bottom-Up Testing)

बॉटम-अप टेस्टिंग मध्ये टेस्ट ड्रायव्हर हा एक तात्पुरता प्रोग्राम म्हणून वापरला जातो, जो लोअर-लेव्हल मॉड्यूल (डायग्राममध्ये Module B) ला कॉल करतो. Module B चे स्वतंत्रपणे (isolation मध्ये) टेस्टिंग केले जाते, जेणेकरून ते अपेक्षेप्रमाणे कार्य करत आहे का हे तपासता येईल. Module B चे टेस्टिंग यशस्वी झाल्यानंतर ते हायर-लेव्हल मॉड्यूल Module A सोबत इंटिग्रेट करून पुढील टेस्टिंग केली जाते. ही प्रक्रिया क्रमाक्रमाने सुरू राहते, जोपर्यंत सर्व हायर-लेव्हल मॉड्यूल्स इंटिग्रेट होऊन टेस्ट होत नाहीत.

2. टॉप-डाऊन टेस्टिंग (Top-Down Testing)

टॉप-डाऊन टेस्टिंग मध्ये प्रक्रिया Main Module पासून सुरू होते, जो Module A ला कॉल करतो. Module A ने Module B ला कॉल करणे अपेक्षित असते; परंतु जर Module B उपलब्ध नसेल, तर त्याच्या जागी स्टब (Stub) तयार केला जातो. हा स्टब Module B चे वर्तन सिम्युलेट करतो आणि पूर्वनिश्चित आउटपुट्स देतो. त्यामुळे खालच्या स्तरावरील मॉड्यूलची प्रत्यक्ष अंमलबजावणी नसतानाही, हायर-लेव्हल मॉड्यूल्सचे टेस्टिंग करता येते. या पद्धतीमुळे सिस्टीमचे कंट्रोल लॉजिक, तसेच इंटिग्रेशन पाथ्स यांची लवकर पडताळणी (early validation) करता येते, जरी सर्व कॉम्पोनंट्स पूर्ण झालेले नसले तरी.

उदाहरण – स्टँडर्ड कॅल्क्युलेटर ॲप्लिकेशनमध्ये ड्रायव्हर आणि स्टब

परिस्थिती (Scenario): आपण एक स्टँडर्ड कॅल्क्युलेटर ॲप्लिकेशन विकसित करत आहोत.

- add() फंक्शन इम्प्लिमेंट झालेले आहे, पण मेन प्रोग्राम (UI) अजून तयार नाही →अशा वेळी add() फंक्शनची टेस्टिंग करण्यासाठी Driver वापरतो.
- getInput() फंक्शन (लोअर-लेव्हल मॉड्यूल) अजून तयार नाही → अशा वेळी Stub वापरून getInput() चे वर्तन सिम्युलेट केले जाते.

Test Case ID (टेस्ट केस आयडी)	Type (प्रकार)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
TC_ADD_001	Driver (ड्रायव्हर)	Addition of two positive integers दोन सकारात्मक पूर्णांकांची बेरीज	add(5, 3)	8	8	Pass (यशस्वी)
TC_ADD_002	Driver (ड्रायव्हर)	Addition of positive and negative integer सकारात्मक व नकारात्मक पूर्णांकांची बेरीज	add(7, -2)	5	6 (bug) 6 (बग)	Fail (अयशस्वी)
TC_INPUT_001	Stub (स्टब)	Simulate input for addition बेरीजसाठी इनपुट सिम्युलेट करणे	getInput() → returns (5, 3)	(5, 3)	(5, 3)	Pass (यशस्वी)
TC_INPUT_002	Stub (स्टब)	Simulate input for subtraction वजाबाकीसाठी इनपुट सिम्युलेट करणे	getInput() → returns (9, 4)	(9, 4)	(9, 4)	Pass (यशस्वी)

Table 2.1: ड्रायव्हर वि. स्टब (Comparison between Driver and Stub)

निकष (Criteria)	ड्रायव्हर (Driver)	स्टब (Stub)
व्याख्या (Definition)	हायर-लेव्हल मॉड्यूलचे सिम्युलेशन करणारा तात्पुरता प्रोग्राम/मॉड्यूल, जो टेस्ट होणाऱ्या युनिटला कॉल करतो. वरचा मॉड्यूल तयार नसताना लोअर मॉड्यूल टेस्ट करण्यासाठी वापरला जातो.	लोअर-लेव्हल मॉड्यूलचे सिम्युलेशन करणारा तात्पुरता प्रोग्राम/मॉड्यूल, जो टेस्ट होणाऱ्या युनिटद्वारे कॉल केला जातो. खालचा मॉड्यूल तयार नसताना हायर मॉड्यूल टेस्ट करण्यासाठी वापरला जातो.
उद्देश (Purpose)	कॉल करणारा (हायर-लेव्हल) मॉड्यूल उपलब्ध नसताना मॉड्यूलला कॉल करून टेस्ट करणे.	कॉल होणारा (लोअर-लेव्हल) मॉड्यूल उपलब्ध नसताना प्रतिसाद देणे.
टेस्टिंग ॲप्रोच (Testing Approach)	बॉटम-अप इंटिग्रेशन टेस्टिंग.	टॉप-डाऊन इंटिग्रेशन टेस्टिंग.

दिशा (Direction)	लोअर मॉड्यूल्सचे टेस्टिंग कंट्रोल आणि इनिशिएट करतो.	हायर मॉड्यूल्सना कंट्रोल्ड प्रतिसाद परत करतो.
कोणाची जागा घेतो (Who Replaces Who)	कॉलिंग मॉड्यूलची जागा घेतो.	कॉल होणाऱ्या मॉड्यूलची जागा घेतो.
डेव्हलपमेंट स्टेज (Development Stage)	जेव्हा लोअर-लेव्हल मॉड्यूल्स तयार आहेत पण हायर-लेव्हल मॉड्यूल्स तयार नाहीत.	जेव्हा हायर-लेव्हल मॉड्यूल्स तयार आहेत पण लोअर-लेव्हल मॉड्यूल्स तयार नाहीत.
उदाहरण (Example Use)	UI तयार नसताना add() फंक्शन टेस्ट करणे.	getInput() फंक्शन इम्प्लिमेंट नसताना त्याचे सिम्युलेशन करणे.
प्रभाव (Impact)	लोअर-लेव्हल मॉड्यूल्सची लवकर पडताळणी करता येते.	हायर-लेव्हल मॉड्यूल्सची लवकर पडताळणी करता येते.

स्टब्स आणि ड्रायव्हर्सची गरज (Needs of Stubs and Drivers)

1. लवकर टेस्टिंग शक्य करणे (Enable Early Testing): सर्व मॉड्यूल्स पूर्ण होण्यापूर्वीच टेस्टिंग करता येते.
2. इन्क्रिमेंटल इंटिग्रेशनला समर्थन (Support Incremental Integration): टॉप-डाऊन आणि बॉटम-अप टेस्टिंग ॲप्रोच सुलभ करतात.
3. अनुपलब्ध मॉड्यूल्सचे सिम्युलेशन (Simulate Missing Modules): उपलब्ध नसलेल्या मॉड्यूल्ससाठी तात्पुरते पर्याय उपलब्ध करून देतात.
4. पॅरॅलल डेव्हलपमेंट (Parallel Development): वेगवेगळ्या टीम्सना स्वतंत्र मॉड्यूल्सवर एकाच वेळी काम करता येते.
5. डिफेक्ट्सची लवकर ओळख (Early Defect Detection): स्वतंत्र मॉड्यूल्समधील चुका लवकर सापडतात.
6. टेस्टिंगमध्ये होणारा विलंब कमी करणे (Reduce Testing Delays): पूर्ण सिस्टीम इंटिग्रेशनची वाट पाहावी लागत नाही.
7. खर्चात बचत (Cost Efficiency): डेव्हलपमेंटच्या सुरुवातीच्या टप्प्यात चुका दुरुस्त केल्यामुळे खर्च कमी होतो.
8. टेस्टिंगमध्ये लवचिकता (Flexibility in Testing): मॉड्यूल्स स्वतंत्रपणे (isolation मध्ये) टेस्ट करता येतात.

युनिट टेस्टिंगचे फायदे आणि तोटे (Advantages and Disadvantages of Unit Testing)

युनिट टेस्टिंगचे फायदे (Advantages of Unit Testing):

1. डेव्हलपमेंट प्रक्रियेत डिफेक्ट्स लवकर सापडतात.
2. चुका दुरुस्त करण्याचा खर्च कमी होतो.
3. कोड क्वालिटी सुधारते आणि मॉड्युलर डिझाईन ला प्रोत्साहन मिळते.
4. सॉफ्टवेअरची मेंटेनन्स आणि अपडेट्स सोपी होतात.
5. सेफ कोड रिफॅक्टरिंग शक्य होते.
6. कंटिन्युअस इंटिग्रेशन आणि ऑटोमेशन ला समर्थन मिळते.
7. सॉफ्टवेअरची रिलायबिलिटी वाढते.
8. एक्झिक्यूटेबल डॉक्युमेंटेशन म्हणून काम करते.

युनिट टेस्टिंगचे तोटे (Disadvantages of Unit Testing)

1. सुरुवातीला जास्त वेळ आणि प्रयत्न लागतात.
2. इंटिग्रेशनशी संबंधित समस्या शोधू शकत नाही.
3. टेस्ट केसेसच्या मेंटेनन्सचा अतिरिक्त भार पडतो.
4. सॉफ्टवेअरच्या क्वालिटीबाबत खोटी खात्री (false sense of quality) निर्माण होऊ शकते.
5. कुशल टेस्टर किंवा डेव्हलपर्सची आवश्यकता असते.
6. मर्यादित स्कोप – परफॉर्मन्स, सिक्युरिटी किंवा युजेबिलिटी टेस्टिंग करू शकत नाही.

2.2 इंटिग्रेशन टेस्टिंग (Integration Testing):

इंटिग्रेशन टेस्टिंग ही सॉफ्टवेअर टेस्टिंगची एक प्रक्रिया आहे, ज्यामध्ये इंटिग्रेट केलेल्या मॉड्यूल्स किंवा युनिट्समधील इंटरफेस आणि परस्परसंवाद (interaction) योग्य प्रकारे कार्य करतो का हे तपासले जाते. यामध्ये युनिट टेस्टिंगनंतर मॉड्यूल्स एकत्र केल्यानंतर त्यांची चाचणी केली जाते आणि विशेषतः डेटा फ्लो, कंट्रोल फ्लो तसेच मॉड्यूल्समधील कम्युनिकेशन यावर लक्ष केंद्रित केले जाते. इंटिग्रेशन टेस्टिंग ही युनिट टेस्टिंगनंतर आणि सिस्टीम टेस्टिंगपूर्वी केली जाते. याचा मुख्य उद्देश म्हणजे स्वतंत्रपणे योग्यरीत्या काम करणारी मॉड्यूल्स एकत्र आल्यावर देखील योग्य प्रकारे कार्य करतात का हे सुनिश्चित करणे. अनेक वेळा मॉड्यूल्स एकमेकांशी संवाद साधताना इंटरफेस-संबंधित चुका निर्माण होतात; अशा चुका शोधणे हे इंटिग्रेशन टेस्टिंगचे प्रमुख ध्येय आहे. इंटिग्रेशन टेस्टिंगमध्ये पुढील बाबी तपासल्या जातात: मॉड्यूल्समधील इंटरफेस आणि डेटा एक्सचेंज योग्य आहे का, कम्युनिकेशन प्रोटोकॉल्स आणि मॉड्यूल्समधील कम्पॅटिबिलिटी योग्य आहे का. ही टेस्टिंग सहसा इंटिग्रेशन टेस्टर किंवा डेव्हलपर्स करतात, कारण त्यांना सिस्टीम आर्किटेक्चर आणि मॉड्यूल्समधील परस्परसंबंधांची चांगली माहिती असते.

इंटिग्रेशन टेस्टिंगचे ॲप्रोचेस (Approaches of Integration Testing)

1. बिग बँग इंटिग्रेशन (Big Bang Integration): या पद्धतीत सर्व मॉड्यूल्स एकाच वेळी इंटिग्रेट करून टेस्टिंग केली जाते. अंमलबजावणी सोपी असते परंतु डिफेक्ट्स शोधणे आणि डीबगिंग करणे कठीण असते.
2. इन्क्रिमेंटल इंटिग्रेशन (Incremental Integration): या पद्धतीत मॉड्यूल्स टप्प्याटप्प्याने (step-by-step) इंटिग्रेट केली जातात. प्रत्येक टप्प्यानंतर टेस्टिंग केली जाते आणि चुका दुरुस्त केल्यावर पुढील टप्प्याकडे जातात.

इन्क्रिमेंटल इंटिग्रेशनचे तीन प्रमुख प्रकार आहेत:

1. टॉप-डाऊन इंटिग्रेशन (Top-Down Integration): या पद्धतीत हायर-लेव्हल मॉड्यूल्स प्रथम टेस्ट केली जातात. लोअर-लेव्हल मॉड्यूल्स उपलब्ध नसतील तर स्टब्स (Stubs) वापरले जातात. सिस्टीमचे कंट्रोल फ्लो आणि लॉजिक लवकर व्हेरिफाय करता येते.
   फायदे: लवकर सिस्टीम बिहेवियर तपासता येतो.
   तोटे: लोअर-लेव्हल मॉड्यूल्सची सखोल टेस्टिंग उशिरा होते.
2. बॉटम-अप इंटिग्रेशन (Bottom-Up Integration):या पद्धतीत लोअर-लेव्हल मॉड्यूल्स प्रथम टेस्ट केली जातात. हायर-लेव्हल मॉड्यूल्स उपलब्ध नसतील तर ड्रायव्हर्स (Drivers) वापरले जातात, बेसिक फंक्शनॅलिटी आधीच मजबूत केली जाते.
   फायदे: लोअर-लेव्हल मॉड्यूल्सची सखोल टेस्टिंग होते.
   तोटे: संपूर्ण सिस्टीमचे वर्तन उशिरा दिसते.
3. बाय-डायरेक्शनल / सँडविच इंटिग्रेशन (Bi-Directional / Sandwich Integration): ही पद्धत टॉप-डाऊन आणि बॉटम-अप या दोन्हींचा संगम आहे. मधल्या लेव्हलपासून टेस्टिंग सुरू होते, वरच्या बाजूस स्टब्स आणि खालच्या बाजूस ड्रायव्हर्स वापरले जातात.
   फायदे: दोन्ही पद्धतींचे फायदे मिळतात, इंटिग्रेशन अधिक संतुलित होते

इंटिग्रेशन टेस्टिंगसाठी टूल्स (Tools for Integration Testing)

इंटिग्रेशन टेस्टिंगसाठी खालील टूल्स वापरली जातात:

- सेलेनियम (Selenium)
- जेयुनिट (JUnit)
- टेस्टएनजी (TestNG)
- सिट्रस फ्रेमवर्क (Citrus Framework)
- फिटनेस (FitNesse)

ही टूल्स टेस्ट ऑटोमेशन, टेस्ट केस मॅनेजमेंट आणि मॉड्यूल इंटरॲक्शन व्हेरिफिकेशन साठी उपयुक्त ठरतात.

इंटिग्रेशन टेस्टिंग प्रोसेस (Process of Integration Testing)

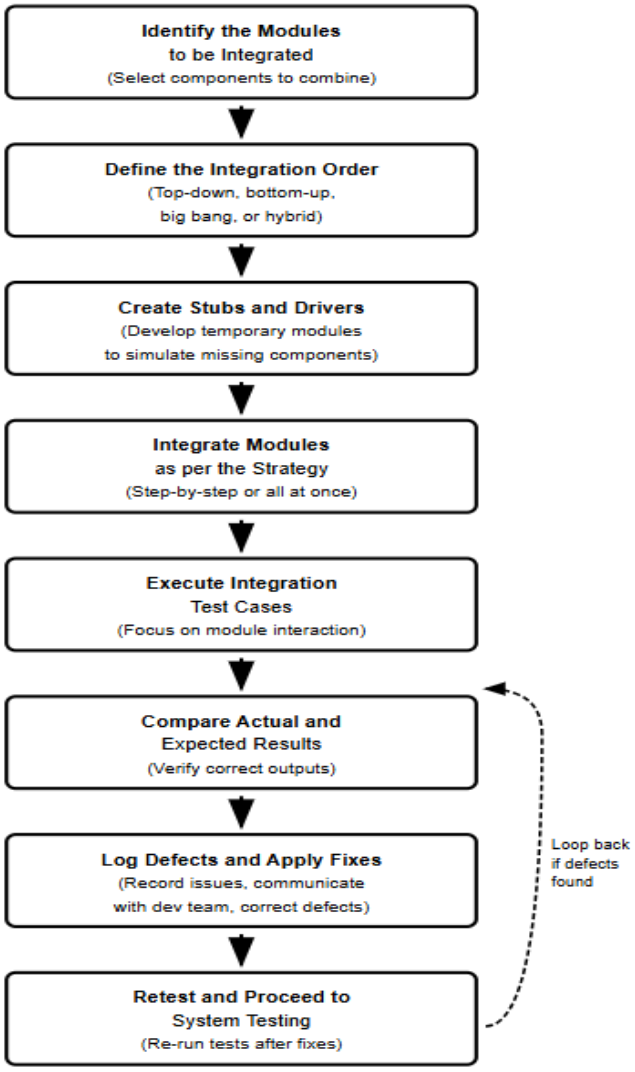

Fig 2.4: इंटिग्रेशन टेस्टिंग प्रोसेस (Process of Integration Testing)

1. **इंटिग्रेट करावयाची मॉड्यूल्स ओळखणे (Identify the Modules to be Integrated):** टेस्टिंगसाठी एकत्र केली जाणारी युनिट्स किंवा कॉम्पोनंट्स निवडणे.
2. **इंटिग्रेशनचा क्रम ठरवणे (Define the Integration Order):** टॉप-डाऊन, बॉटम-अप, बिग बँग किंवा हायब्रिड (सँडविच) ॲप्रोच यापैकी कोणती पद्धत वापरायची ते ठरवणे.
3. **स्टब्स आणि ड्रायव्हर्स तयार करणे (Create Stubs and Drivers):** अनुपलब्ध असलेल्या हायर-लेव्हल किंवा लोअर-लेव्हल मॉड्यूल्सचे सिम्युलेशन करण्यासाठी तात्पुरती मॉड्यूल्स तयार करणे.
4. **रणनीतीनुसार मॉड्यूल्स इंटिग्रेट करणे (Integrate Modules as per the Strategy):** निवडलेल्या ॲप्रोचनुसार मॉड्यूल्स टप्प्याटप्प्याने किंवा एकाच वेळी एकत्र करणे.
5. **इंटिग्रेशन टेस्ट केसेस एक्झिक्यूट करणे (Execute Integration Test Cases):** मॉड्यूल इंटरॲक्शन आणि डेटा एक्सचेंज वर लक्ष केंद्रित करणाऱ्या टेस्ट केसेस रन करणे.
6. **प्रत्यक्ष आणि अपेक्षित निकालांची तुलना (Compare Actual and Expected Results):** इंटिग्रेट केलेली मॉड्यूल्स योग्य आणि सुसंगत आउटपुट्स देत आहेत का ते तपासणे.
7. **डिफेक्ट्स लॉग करणे आणि दुरुस्त्या करणे (Log Defects and Apply Fixes):** सापडलेल्या समस्यांची नोंद (log) करणे, डेव्हलपमेंट टीमशी संवाद साधणे आणि डिफेक्ट्स दुरुस्त करणे.
8. **री-टेस्टिंग आणि सिस्टीम टेस्टिंगकडे वाटचाल (Retest and Proceed to System Testing):** दुरुस्तीनंतर फेल झालेल्या टेस्ट्स पुन्हा रन करणे आणि इंटिग्रेशन स्थिर झाल्यावर सिस्टीम टेस्टिंग सुरू करणे.

उदाहरण – कॅल्क्युलेटरसाठी इंटिग्रेशन टेस्टिंग

परिस्थिती (Scenario): आपल्याकडे आधीच टेस्ट केलेली दोन युनिट्स आहेत:

1. add() – दोन संख्यांची बेरीज करते
2. subtract() – दोन संख्यांची वजाबाकी करते

आता ही दोन्ही युनिट्स calculate () या एकाच फंक्शनमध्ये इंटिग्रेट केली आहेत, जे युजर इनपुटनुसार ऑपरेशन निवडते. इंटिग्रेशन टेस्टिंगचा उद्देश: दोन्ही फंक्शन्स एकत्र योग्यरित्या कार्य करतात का, फंक्शन्सदरम्यान डेटा योग्य प्रकारे पास होत आहे का.

इंटिग्रेशन टेस्ट केसेस (Integration Test Cases)

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
TC_INT_001	Verify addition through calculate() कॅल्क्युलेट() द्वारे बेरीज तपासणे	calculate(5, 3, "add")	8	8	Pass (यशस्वी)
TC_INT_002	Verify subtraction through calculate() कॅल्क्युलेट() द्वारे वजाबाकी तपासणे	calculate(10, 4, "subtract")	6	6	Pass (यशस्वी)
TC_INT_003	Verify mixed operation sequence मिश्र ऑपरेशन सिक्वेन्स तपासणे	result = calculate(5, 3, "add");result = calculate(result, 2, "subtract")	6	5 (bug) 5 (बग)	Fail (अयशस्वी)
TC_INT_004	Invalid operation handling अवैध ऑपरेशन हँडलिंग	calculate(5, 3, "multiply")	Error: Operation not supported त्रुटी: ऑपरेशन समर्थित नाही	Error: Operation not supported	Pass (यशस्वी)

2.2.1 टॉप-डाऊन इंटिग्रेशन (Top-Down Integration)

टॉप-डाऊन इंटिग्रेशन टेस्टिंग ही एक अशी पद्धत आहे, ज्यामध्ये टेस्टिंगची सुरुवात टॉप-लेव्हल (हायर-लेव्हल) मॉड्यूल्सपासून केली जाते आणि नंतर हळूहळू लोअर-लेव्हल मॉड्यूल्सकडे प्रगती केली जाते. प्रत्येक टप्प्यावर मॉड्यूल्स इंटिग्रेट करून टेस्टिंग केली जाते.जर काही लोअर-लेव्हल मॉड्यूल्स अजून विकसित झालेले नसतील, तर त्यांची जागा स्टब्स (Stubs) घेऊन त्यांचे वर्तन सिम्युलेट केले जाते, जोपर्यंत प्रत्यक्ष मॉड्यूल्स उपलब्ध होत नाहीत.

टॉप-डाऊन इंटिग्रेशन का उपयुक्त आहे?

- जेव्हा वर्किंग सिस्टीमचे लवकर डेमो देणे आवश्यक असते
- जेव्हा मेन कंट्रोल लॉजिक आधीच तपासायचे असते

टॉप-डाऊन इंटिग्रेशन कसे केले जाते?

- प्रथम मेन कंट्रोल मॉड्यूल टेस्ट केले जाते
- त्यानंतर सब-मॉड्यूल्स एक-एक करून इंटिग्रेट केली जातात
- इंटिग्रेशन डेप्थ-फर्स्ट किंवा ब्रेड्थ-फर्स्ट पद्धतीने केली जाऊ शकते

- टेस्टिंग पुढे सरकत असताना स्टब्सच्या जागी प्रत्यक्ष मॉड्यूल्स बसवले जातात

फायदे (Advantages):

1. हायर-लेव्हल लॉजिक आणि डेटा फ्लोची लवकर पडताळणी करता येते
2. कंट्रोल स्ट्रक्चरमधील चुका लवकर सापडतात

योग्य वापर कधी करावा?

1. जेव्हा टॉप-लेव्हल डिझाईन स्थिर (stable) असते
2. जेव्हा महत्त्वाचे कंट्रोल फ्लोज लवकर टेस्ट करणे आवश्यक असते

उदाहरण – स्टँडर्ड कॅल्क्युलेटरसाठी टॉप-डाऊन इंटिग्रेशन टेस्टिंग

परिस्थिती (Scenario):आपल्याकडे calculate () हा मेन मॉड्यूल आहे, ज्यामध्ये मेन्यू सिलेक्शन आहे.

- Addition आणि Subtraction मॉड्यूल्स उपलब्ध आहेत
- Multiplication आणि Division मॉड्यूल्स अजून इम्प्लिमेंट झालेले नाहीत

सुरुवातीला Multiplication आणि Division साठी स्टब्स वापरले जातात. पुढे, ही मॉड्यूल्स तयार झाल्यावर स्टब्सच्या जागी प्रत्यक्ष मॉड्यूल्स बसवली जातात आणि पुन्हा टेस्टिंग केली जाते.

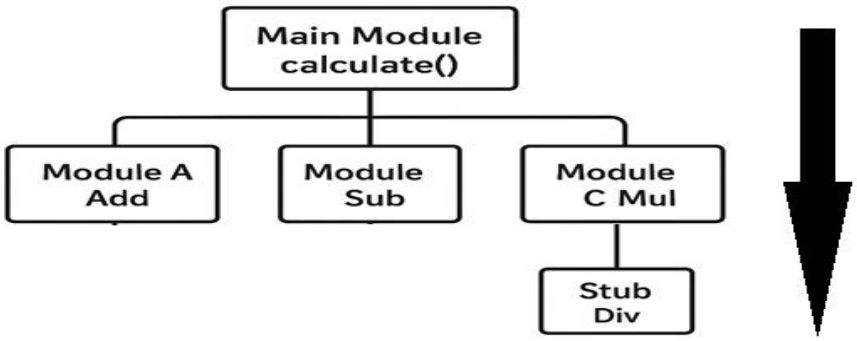

Fig 2.5: टॉप-डाऊन इंटिग्रेशन (Top-Down Integration Testing)

मेन मॉड्यूल (Main Module) – सर्वात वरचे कंट्रोल युनिट (कॅल्क्युलेटर उदाहरणात calculate()).

- मॉड्यूल A (Add) – सर्वप्रथम इम्प्लिमेंट आणि टेस्ट केले जाते.
- मॉड्यूल B (Sub) – पुढील टप्प्यात इम्प्लिमेंट आणि टेस्ट केले जाते.
- मॉड्यूल C (Mul) – उपलब्ध आहे, परंतु Division मॉड्यूल तयार होईपर्यंत Stub (Div) वर अवलंबून असते.
- स्टब्स (Stubs) – सुरुवातीच्या टेस्टिंग फेजमध्ये अनुपलब्ध मॉड्यूल्सचे सिम्युलेशन करतात.

टॉप-डाऊन इंटिग्रेशनसाठी टेस्ट केसेस (Test Cases):

Test Case ID (टेस्ट केस आयडी)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
TC_TD_001	Verify addition via main module मुख्य मॉड्यूलद्वारे बेरीज तपासणे	calculate(5, 3, "add")	8	8	Pass (यशस्वी)
TC_TD_002	Verify subtraction via main module मुख्य मॉड्यूलद्वारे वजाबाकी तपासणे	calculate(9, 4, "subtract")	5	5	Pass (यशस्वी)

TC_TD_003	Verify multiplication stub response गुणाकार स्टबचा प्रतिसाद तपासणे	calculate(6, 2, "multiply")	"Multiplication module not implemented" "गुणाकार मॉड्यूल अंमलात आणलेले नाही"	"Multiplication module not implemented"	Pass (यशस्वी)
TC_TD_004	Verify division stub response भागाकार स्टबचा प्रतिसाद तपासणे	calculate(8, 4, "divide")	"Division module not implemented" "भागाकार मॉड्यूल अंमलात आणलेले नाही"	"Division module not implemented"	Pass (यशस्वी)

2.2.2 बॉटम-अप इंटिग्रेशन (Bottom-Up Integration)

बॉटम-अप इंटिग्रेशन टेस्टिंग ही इंटिग्रेशन टेस्टिंगची एक इन्क्रिमेंटल पद्धत आहे, ज्यामध्ये टेस्टिंगची सुरुवात सॉफ्टवेअर हायरार्कीमधील सर्वात खालच्या स्तरावरील (lowest-level) मॉड्यूल्सपासून केली जाते आणि नंतर हळूहळू वरच्या स्तरावरील मॉड्यूल्सकडे प्रगती केली जाते. मॉड्यूल्स क्लस्टर्स स्वरूपात एकत्र करून टेस्ट केली जातात आणि नंतर हायर-लेव्हल मॉड्यूल्स इंटिग्रेट केली जातात. बॉटम-अप इंटिग्रेशन टेस्टिंगची सुरुवात लीफ-लेव्हल (leaf-level) मॉड्यूल्स पासून होते. हायर-लेव्हल मॉड्यूल्स अजून उपलब्ध नसतील, तर त्यांच्या कार्यक्षमतेचे सिम्युलेशन करण्यासाठी ड्रायव्हर्स (Drivers) वापरले जातात. टेस्ट केलेली मॉड्यूल्स क्लस्टर्समध्ये गटबद्ध केली जातात आणि त्यांचे परस्परसंवाद योग्य आहे का हे तपासले जाते. क्लस्टर टेस्टिंगनंतर, टप्प्याटप्प्याने वरच्या स्तरावरील मॉड्यूल्स इंटिग्रेट करून टेस्टिंग केली जाते, जोपर्यंत संपूर्ण सिस्टीम तयार होऊन व्हेरिफाय होत नाही. ही पद्धत विशेषतः तेव्हा उपयुक्त ठरते, जेव्हा सिस्टीमची महत्त्वाची फंक्शनॅलिटी खालच्या स्तरावर असते. याचा एक मोठा फायदा म्हणजे युटिलिटी मॉड्यूल्समधील चुका लवकर सापडतात. मात्र, यातील एक महत्त्वाची मर्यादा अशी आहे की सिस्टीम-लेव्हल डिझाईनमधील त्रुटी उशिरा लक्षात येऊ शकतात. या पद्धतीत प्रामुख्याने ड्रायव्हर्सची आवश्यकता असते, तर स्टब्स सहसा आवश्यक नसतात.

उदाहरण – स्टँडर्ड कॅल्क्युलेटरसाठी बॉटम-अप इंटिग्रेशन टेस्टिंग

परिस्थिती (Scenario): आपल्याकडे एक कॅल्क्युलेटर ॲप्लिकेशन आहे, ज्यामध्ये calculate() हा मेन मॉड्यूल चार मूलभूत ऑपरेशन्ससाठी मेन्यू पुरवतो : ॲडिशन, सब्ट्रॅक्शन, मल्टिप्लिकेशन आणि डिव्हिजन(Addition, Subtraction, Multiplication and Division).

बॉटम-अप इंटिग्रेशन टेस्टिंगमध्ये:

- सर्वप्रथम Division (सर्वात खालचा मॉड्यूल) टेस्ट केला जातो
- त्यानंतर त्याचा पॅरेंट मॉड्यूल Multiplication टेस्ट केला जातो
- शेवटी Addition, Subtraction आणि calculate () (हायर-लेव्हल मॉड्यूल्स) इंटिग्रेट करून टेस्ट केले जातात

सुरुवातीच्या टप्प्यात मेन प्रोग्राम उपलब्ध नसल्यामुळे, त्याचे वर्तन सिम्युलेट करण्यासाठी टेस्ट ड्रायव्हर्स वापरले जातात. संपूर्ण इंटिग्रेशन पूर्ण झाल्यावर ड्रायव्हर्स काढून टाकले जातात.

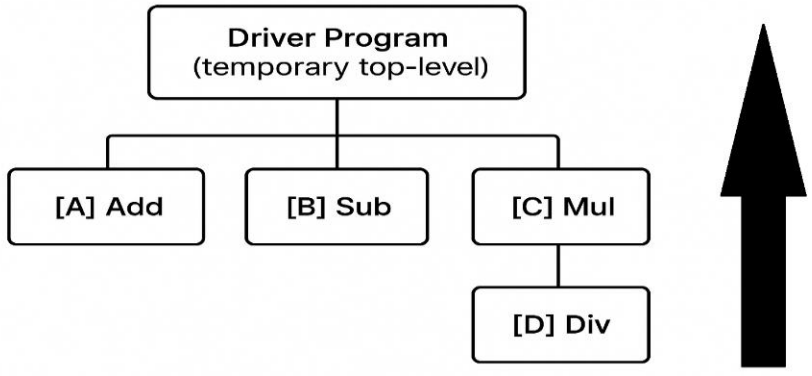

Fig 2.6: बॉटम-अप इंटिग्रेशन टेस्टिंगमध्ये (Bottom-Up Integration Testing)

मेन मॉड्यूल (calculate()) – हे कॅल्क्युलेटरचे सर्वोच्च कंट्रोल युनिट आहे, जे Add, Sub, Mul, Div अशा ऑपरेशन्ससाठी मेन्यू सिलेक्शन करते.बॉटम-अप इंटिग्रेशनमध्ये हे सुरुवातीला इम्प्लिमेंट केले जात नाही. त्याऐवजी, त्याचे वर्तन सिम्युलेट करण्यासाठी ड्रायव्हर वापरला जातो.

1. **मॉड्यूल D (Div)** – हे सर्वात खालचे (lowest-level) मॉड्यूल आहे. बॉटम-अप टेस्टिंग खालच्या स्तरापासून सुरू होत असल्यामुळे Division मॉड्यूल सर्वप्रथम टेस्ट केले जाते. ड्रायव्हर थेट Division ला कॉल करतो.
2. **मॉड्यूल C (Mul)** – Division यशस्वीपणे टेस्ट झाल्यानंतर Multiplication मॉड्यूल इंटिग्रेट करून टेस्ट केले जाते. या टप्प्यावरही ड्रायव्हर टेस्ट फ्लो कंट्रोल करतो.
3. **मॉड्यूल A (Add)** – त्यानंतर Addition मॉड्यूल ड्रायव्हर प्रोग्रामद्वारे इंटिग्रेट करून टेस्ट केले जाते.
4. **मॉड्यूल B (Sub)** – शेवटी Subtraction मॉड्यूल ड्रायव्हरद्वारे इंटिग्रेट आणि टेस्ट केले जाते.
5. **ड्रायव्हर प्रोग्राम (Driver Program)** – हा एक तात्पुरता प्रोग्राम असतो, जो मेन मॉड्यूल (calculate()) उपलब्ध नसताना त्याची जागा घेतो. हा ड्रायव्हर इनपुट पुरवतो आणि Add, Sub, Mul, Div या लोअर-लेव्हल मॉड्यूल्सना कॉल करून टेस्टिंग करतो. सर्व मॉड्यूल्स योग्यरीत्या व्हेरिफाय झाल्यानंतर ड्रायव्हर काढून टाकला जातो आणि त्याच्या जागी प्रत्यक्ष मेन मॉड्यूल (calculate()) वापरले जाते.

बॉटम-अप इंटिग्रेशनसाठी टेस्ट केसेस (Test Cases)

Test Case ID (टेस्ट केस आयडी)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
TC_BU_001	Verify division module independently using driver ड्रायव्हर वापरून भागाकार मॉड्यूल स्वतंत्रपणे तपासणे	Input: 20 ÷ 5	4	4	Pass (यशस्वी)
TC_BU_002	Verify multiplication module independently using driver ड्रायव्हर वापरून गुणाकार मॉड्यूल स्वतंत्रपणे तपासणे	Input: 6 × 7	42	42	Pass (यशस्वी)
TC_BU_003	Verify addition module independently using driver ड्रायव्हर वापरून बेरीज मॉड्यूल स्वतंत्रपणे तपासणे	Input: 5 + 3	8	8	Pass (यशस्वी)
TC_BU_004	Verify subtraction module independently using driver ड्रायव्हर वापरून वजाबाकी मॉड्यूल स्वतंत्रपणे तपासणे	Input: 10 - 4	6	6	Pass (यशस्वी)

2.2.3 बाय-डायरेक्शनल इंटिग्रेशन (Bi-Directional Integration / Hybrid / Sandwich Integration)

बाय-डायरेक्शनल इंटिग्रेशन टेस्टिंग ही एक हायब्रिड पद्धत आहे, जी टॉप-डाऊन आणि बॉटम-अप या दोन्ही इंटिग्रेशन स्ट्रॅटेजीजचा संगम करते. या पद्धतीमध्ये टेस्टिंगची सुरुवात महत्त्वाच्या मधल्या स्तरावरील (middle-level) मॉड्यूलपासून केली जाते आणि नंतर टेस्टिंग दोन दिशांनी पुढे जाते.

- वरच्या दिशेने (Upward) – हायर-लेव्हल मॉड्यूल्स अजून उपलब्ध नसतील तर त्यांचे वर्तन सिम्युलेट करण्यासाठी स्टब्स (Stubs) वापरले जातात.
- खालच्या दिशेने (Downward) – लोअर-लेव्हल मॉड्यूल्सचे कंट्रोल लॉजिक सिम्युलेट करण्यासाठी ड्रायव्हर्स (Drivers) वापरले जातात.

या दुहेरी ॲप्रोचमुळे कंट्रोल लॉजिक (control logic) आणि कम्प्युटेशनल लॉजिक (computational logic) या दोन्ही प्रकारच्या त्रुटी फक्त टॉप-डाऊन किंवा बॉटम-अप वापरण्यापेक्षा लवकर ओळखता येतात.

बाय-डायरेक्शनल इंटिग्रेशनचे फायदे

- टॉप-डाऊन घटक: हायर-लेव्हल मॉड्यूल्समधील कंट्रोल फ्लो आणि डिसिजन-मेकिंग लॉजिक ची पडताळणी होते.
- बॉटम-अप घटक: लोअर-लेव्हल मॉड्यूल्समधील कम्प्युटेशनल अचूकता तपासली जाते.

मधल्या मॉड्यूलपासून टेस्टिंग सुरू केल्यामुळे, ही पद्धत स्टब्स आणि ड्रायव्हर्सवरील अवलंबन संतुलित ठेवते आणि महत्त्वाची मॉड्यूल्स डेव्हलपमेंटच्या सुरुवातीच्या टप्प्यातच टेस्ट करता येतात. यामुळे बेहतर कव्हरेज मिळते, इंटिग्रेशन इश्यूज लवकर सापडतात आणि डेव्हलपमेंट सायकल अधिक कार्यक्षम बनते.

उदाहरण – स्टँडर्ड कॅल्क्युलेटरसाठी बाय-डायरेक्शनल इंटिग्रेशन टेस्टिंग

परिस्थिती (Scenario): आपल्याकडे एक कॅल्क्युलेटर ॲप्लिकेशन आहे, ज्यामध्ये calculate () हा मेन मॉड्यूल चार मूलभूत ऑपरेशन्ससाठी मेन्यू देतो: ॲडिशन, सब्ट्रॅक्शन, मल्टिप्लिकेशन आणि डिव्हिजन (Addition, Subtraction, Multiplication and Division).

बाय-डायरेक्शनल इंटिग्रेशन टेस्टिंगमध्ये टेस्टिंगची सुरुवात मधल्या स्तरावरील मॉड्यूल (उदा. Multiplication) पासून केली जाते आणि एकाच वेळी दोन दिशांनी पुढे जाते. वरच्या दिशेने (Upward Testing) –calculate () सारखी हायर-लेव्हल मॉड्यूल्स उपलब्ध नसतील तर स्टब्स वापरले जातात. खालच्या दिशेने (Downward Testing) –Division सारखी लोअर-लेव्हल मॉड्यूल्स उपलब्ध नसतील तर ड्रायव्हर्स वापरले जातात. यामुळे बॉटम-अपमधील कम्प्युटेशनल अचूकता, टॉप-डाऊनमधील कंट्रोल फ्लो व्हॅलिडेशन हे दोन्ही घटक डेव्हलपमेंट सायकलच्या सुरुवातीलाच तपासले जातात.

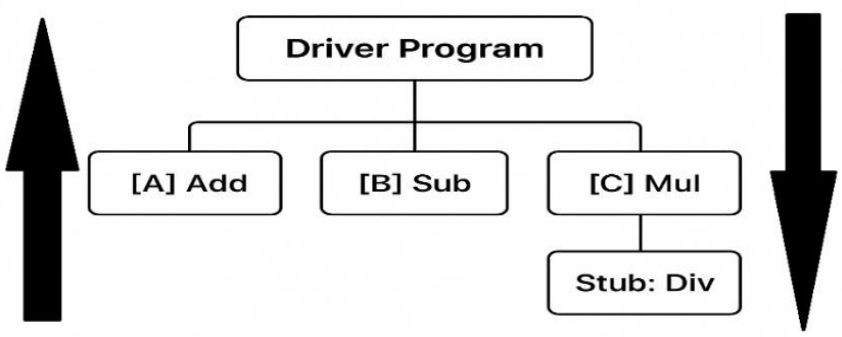

Fig 2.7: बाय-डायरेक्शनल इंटिग्रेशन टेस्टिंगमध्ये (Bi-Directional Integration Testing)

दिलेल्या आकृतीत बाय-डायरेक्शनल इंटिग्रेशन टेस्टिंगची संकल्पना स्पष्ट केली आहे, जिथे Top-Down आणि Bottom-Up या दोन्ही पद्धती एकाच वेळी वापरल्या जातात.

- Driver Program: वरच्या बाजूस असलेला *Driver Program* खालच्या स्तरावरील मॉड्युल्सना कॉल करून चाचणी करण्यासाठी वापरला जातो. हे Bottom-Up Testing दर्शवते.
- [A] Add, [B] Sub, [C] Mul: ही स्वतंत्र फंक्शनल मॉड्युल्स आहेत जी प्रत्यक्ष अंमलबजावणी (actual implementation) म्हणून तपासली जातात.
- Stub: Div: *Div* मॉड्युल अजून विकसित नसेल किंवा उपलब्ध नसेल, तर त्याच्या जागी Stub वापरले जाते. हे Top-Down Testingचे उदाहरण आहे.

Bi-Directional Integration साठी टेस्ट केसेस (Test Cases)

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
TC_BD_001	Verify Multiplication module independently (starting point) गुणाकार मॉड्यूल स्वतंत्रपणे तपासणे (सुरुवातीचा बिंदू)	Input: 6 × 7	42	42	Pass (यशस्वी)
TC_BD_002	Verify downward integration of Multiplication with Division (using driver) ड्रायव्हर वापरून गुणाकार व भागाकार मॉड्यूल्सची डाउनवर्ड इंटिग्रेशन तपासणे	Inputs: (6 × 7), (20 ÷ 5)	42, 4	42, 4	Pass (यशस्वी)
TC_BD_003	Verify upward integration with main module via stub (Multiplication) स्टब वापरून मुख्य मॉड्यूलसोबत गुणाकाराचे अपवर्ड इंटिग्रेशन तपासणे	calculate(8, 2, "multiply")	16	16	Pass (यशस्वी)
TC_BD_004	Verify combined integration (Division with main module) भागाकार व मुख्य मॉड्यूलचे संयुक्त इंटिग्रेशन तपासणे	calculate(12, 4, "divide")	3	3	Pass (यशस्वी)

Table 2.2: टॉप-डाउन, बॉटम-अप आणि बाय-डायरेक्शनल इंटिग्रेशनमधील तुलना (Comparison between Top-Down, Bottom-Up, Bi-Directional Integration)

निकष (Criteria)	**टॉप-डाऊन इंटिग्रेशन (Top-Down Integration)**	**बॉटम-अप इंटिग्रेशन (Bottom-Up Integration)**	**बाय-डायरेक्शनल इंटिग्रेशन (Bi-Directional Integration)**
व्याख्या (Definition)	मेन कंट्रोलर (टॉप-लेव्हल मॉड्यूल) पासून टेस्टिंग सुरू करून खालच्या मॉड्यूल्सकडे प्रगती केली जाते. लोअर मॉड्यूल्ससाठी स्टब्स वापरले जातात.	लीफ (लोअर-लेव्हल) मॉड्यूल्स पासून टेस्टिंग सुरू करून वरच्या मॉड्यूल्सकडे प्रगती केली जाते. हायर मॉड्यूल्ससाठी ड्रायव्हर्स वापरले जातात.	मधल्या (क्रिटिकल) मॉड्यूलपासून टेस्टिंग सुरू करून वर आणि खाली दोन्ही दिशांनी प्रगती केली जाते.
सुरुवातीचा बिंदू (Starting Point)	मेन मॉड्यूल (calculate())	लोअर-लेव्हल मॉड्यूल्स (Div, Mul)	मधला क्रिटिकल मॉड्यूल (Mul)

तात्पुरते प्रोग्राम (Temporary Programs)	स्टब्स आवश्यक	ड्रायव्हर्स आवश्यक	स्टब्स आणि ड्रायव्हर्स दोन्ही लागू शकतात (परंतु कमी प्रमाणात)
क्रिटिकल मॉड्यूल टेस्टिंग(Critical Module Tested)	क्रिटिकल लोअर-लेव्हल मॉड्यूल्स उशिरा टेस्ट होतात.	क्रिटिकल हायर-लेव्हल मॉड्यूल्स उशिरा टेस्ट होतात.	क्रिटिकल मधल्या स्तरावरील मॉड्यूल्स लवकर टेस्ट होतात.
डायग्राम (Diagram)	टॉप → डाऊन फ्लो	बॉटम → अप फ्लो	मिडल → अप + डाऊन
उदाहरण (Example)	कॅल्क्युलेटर: calculate () → Add, Sub, Mul. Div साठी Stub.	कॅल्क्युलेटर: Div, Mul प्रथम (Drivers) → नंतर Add, Sub → शेवटी calculate().	कॅल्क्युलेटर: Mul पासून सुरुवात. Div कडे ड्रायव्हर्सने, Add/Sub कडे स्टब्सने, नंतर calculate().
फायदे (Advantages)	•टॉप-लेव्हल डिझाईनची लवकर पडताळणी • लवकर वर्किंग प्रोटोटाइप	• युटिलिटी मॉड्यूल्स लवकर टेस्ट होतात • लोअर-लेव्हल फंक्शन्सची मजबूत पडताळणी	• क्रिटिकल मॉड्यूल्स आधी टेस्ट होतात • वर-खाली दोन्ही बाजूंनी बॅलन्स्ड एरर डिटेक्शन
तोटे (Disadvantages)	• अनेक स्टब्सची गरज • युटिलिटी एरर्स उशिरा सापडतात	• अनेक ड्रायव्हर्सची गरज • टॉप-लेव्हल डिझाईन फ्लॉज उशिरा सापडतात	• नियोजन व मॅनेजमेंट अधिक क्लिष्ट • स्टब्स व ड्रायव्हर्सची गरज राहते

2.3 सिस्टीम टेस्टिंग (System Testing)

सिस्टीम टेस्टिंग हा सॉफ्टवेअर टेस्टिंगचा एक स्तर आहे, ज्यामध्ये पूर्णपणे इंटिग्रेट केलेल्या सॉफ्टवेअर सिस्टीमची संपूर्ण स्वरूपात (as a whole) चाचणी केली जाते. या टप्प्यावर सॉफ्टवेअर प्रॉडक्ट दिलेल्या आवश्यकता पूर्ण करते का आणि त्याच्या अपेक्षित वातावरणात योग्यरीत्या कार्य करते का हे तपासले जाते. युनिट टेस्टिंग आणि इंटिग्रेशन टेस्टिंग मध्ये स्वतंत्र मॉड्यूल्स किंवा मॉड्यूल्समधील परस्परसंवादावर लक्ष दिले जाते, तर सिस्टीम टेस्टिंगमध्ये संपूर्ण ॲप्लिकेशनचे वर्तन तपासले जाते. यामध्ये फंक्शनल आणि नॉन-फंक्शनल अशा दोन्ही पैलूंचा समावेश असतो. सिस्टीम टेस्टिंग ही इंटिग्रेशन टेस्टिंगनंतर आणि ॲक्सेप्टन्स टेस्टिंगपूर्वी केली जाते. याचा मुख्य उद्देश म्हणजे सिस्टीमची एंड-टू-एंड फंक्शनॅलिटी व्हॅलिडेट करणे आणि सॉफ्टवेअरने फंक्शनल तसेच बिझनेस रिक्वायरमेंट्स पूर्ण केल्या आहेत का हे सुनिश्चित करणे. या स्तरावर सिस्टीमकडे एक संपूर्ण घटक (single entity) म्हणून पाहिले जाते. टेस्टिंगमध्ये पुढील बाबींचा समावेश असतो: विविध इंटरफेसेस, मॉड्यूल्समधील परस्परसंवाद, युजेबिलिटी, परफॉर्मन्स, सिक्युरिटी, रिलायबिलिटी आणि संपूर्ण सिस्टीमचे वर्तन.

सिस्टीम टेस्टिंगची प्रमुख वैशिष्ट्ये (Key Characteristics)

1. पूर्णपणे इंटिग्रेट केलेल्या ॲप्लिकेशनवर टेस्टिंग केली जाते
2. सहसा स्वतंत्र टेस्टिंग टीम द्वारे केली जाते
3. मुख्यतः ब्लॅक-बॉक्स टेस्टिंग टेक्निक्स वापरल्या जातात
4. इनपुट आणि आउटपुटवर लक्ष केंद्रित केले जाते, इंटरनल कोडवर नाही
5. फंक्शनल टेस्टिंग (फिचर्स, वर्कफ्लोज)
6. नॉन-फंक्शनल टेस्टिंग (परफॉर्मन्स, लोड, स्ट्रेस, सिक्युरिटी) दोन्हींचा समावेश होतो.

सिस्टीम टेस्टिंग प्रोसेस (System Testing Process)

1. रिक्वायरमेंट ॲनालिसिस (Requirement Analysis): सिस्टीम आणि बिझनेस रिक्वायरमेंट्स समजून घेणे.
2. टेस्ट प्लॅनिंग (Test Planning): टेस्ट स्कोप, स्ट्रॅटेजी, रिसोर्सेस आणि शेड्यूल ठरवणे.
3. टेस्ट केस डिझाईन (Test Case Design): एंड-टू-एंड सिस्टीम बिहेवियरसाठी टेस्ट केसेस तयार करणे.

4. टेस्ट एन्व्हायर्नमेंट सेटअप (Test Environment Setup): हार्डवेअर, सॉफ्टवेअर, नेटवर्क आणि टेस्ट डेटा तयार करणे.
5. टेस्ट एक्झिक्युशन (Test Execution): टेस्ट केसेस रन करून प्रत्यक्ष निकाल नोंदवणे.
6. डिफेक्ट रिपोर्टिंग (Defect Reporting): सापडलेल्या त्रुटींची नोंद व ट्रॅकिंग करणे.
7. री-टेस्टिंग आणि रिग्रेशन टेस्टिंग (Re-testing & Regression Testing): दुरुस्त्या तपासणे आणि आधीची फंक्शनॅलिटी बिघडली नाही याची खात्री करणे.
8. टेस्ट क्लोजर (Test Closure): टेस्ट रिपोर्ट्स तयार करणे आणि सिस्टीम पुढील टप्प्यासाठी तयार असल्याची पुष्टी करणे.

उदाहरण (Example): ऑनलाईन बँकिंग सिस्टीमचे सिस्टीम टेस्टिंग, जिथे लॉगिन, अकाउंट बॅलन्स तपासणे, फंड ट्रान्सफर, लॉगआउट हे सर्व फिचर्स एकत्रितपणे एंड-टू-एंड योग्यरीत्या कार्य करत आहेत का हे तपासले जाते.

सिस्टीम टेस्टिंग प्रोसेस (System Testing Process)

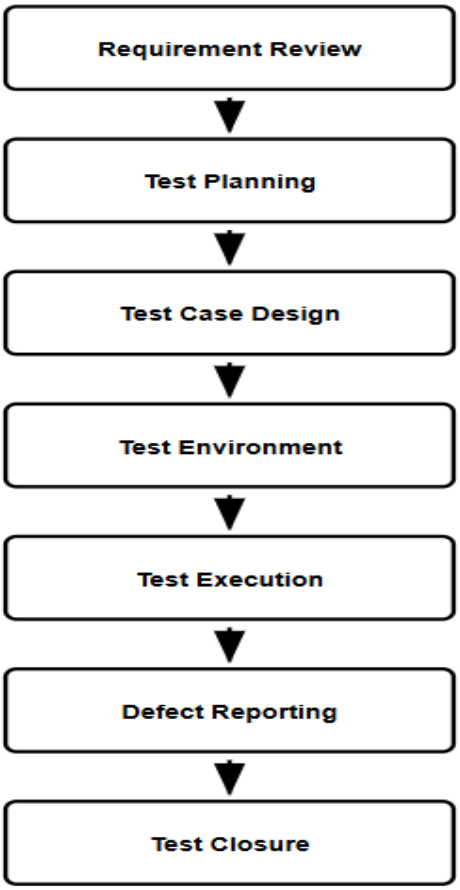

Fig 2.8: सिस्टीम टेस्टिंग प्रोसेस (System Testing Process)

1. **रिक्वायरमेंट ॲनालिसिस (Requirement Analysis):** सिस्टीमच्या आवश्यकतांचा आढावा घेऊन टेस्ट उद्दिष्टे ठरवणे.
2. **टेस्ट प्लॅनिंग (Test Planning):** टेस्ट स्ट्रॅटेजी, रिसोर्सेस, टूल्स आणि शेड्यूल तयार करणे.
3. **टेस्ट केस डिझाईन (Test Case Design):** सविस्तर टेस्ट केसेस आणि अपेक्षित निकाल तयार करणे.
4. **टेस्ट एन्व्हायर्नमेंट सेटअप (Test Environment Setup):** एक्झिक्युशनसाठी आवश्यक हार्डवेअर, सॉफ्टवेअर, डेटाबेस आणि नेटवर्क तयार करणे.
5. **टेस्ट एक्झिक्युशन (Test Execution):** पूर्णपणे इंटिग्रेट केलेल्या सिस्टीमवर टेस्ट केसेस रन करणे.
6. **डिफेक्ट रिपोर्टिंग आणि ट्रॅकिंग (Defect Reporting & Tracking):** सापडलेल्या समस्यांची नोंद करणे, दुरुस्त्या करून री-टेस्टिंग करणे.
7. **टेस्ट क्लोजर (Test Closure):** निकालांचे मूल्यांकन करून टेस्ट समरी रिपोर्ट तयार करणे.

उदाहरण – स्टँडर्ड कॅल्क्युलेटरसाठी सिस्टीम टेस्टिंग

परिस्थिती (Scenario): आपल्याकडे एक कॅल्क्युलेटर ॲप्लिकेशन आहे, ज्यामध्ये calculate () हा मेन मॉड्यूल चार ऑपरेशन्सची परवानगी देतो: ॲडिशन, सब्ट्रॅक्शन, मल्टिप्लिकेशन आणि डिव्हिजन (Addition, Subtraction, Multiplication and Division)

सिस्टीम टेस्टिंगचा उद्देश:

- सर्व ऑपरेशन्स एकत्रितपणे योग्यरित्या कार्य करतात का
- विविध प्रकारच्या इनपुट्ससाठी अपेक्षित आउटपुट मिळते का
- एरर कंडिशन्स योग्यरीत्या हँडल होतात का

सिस्टीम टेस्ट केसेस (System Test Cases)

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
TC_SYS_001	Verify Addition function बेरीज फंक्शन तपासणे	Input: 5 + 3	8	8	Pass (यशस्वी)
TC_SYS_002	Verify Subtraction function वजाबाकी फंक्शन तपासणे	Input: 10 - 4	6	5	Fail (अयशस्वी)
TC_SYS_003	Verify Multiplication function गुणाकार फंक्शन तपासणे	Input: 6 × 7	42	42	Pass (यशस्वी)
TC_SYS_004	Verify Division function भागाकार फंक्शन तपासणे	Input: 20 ÷ 5	4	0	Fail (अयशस्वी)
TC_SYS_005	Verify Division by zero (error handling) शून्याने भागाकार (एरर हँडलिंग) तपासणे	Input: 10 ÷ 0	Error: Division by Zero त्रुटी: शून्याने भागाकार	Error: Division by Zero	Pass (यशस्वी)
TC_SYS_006	Verify multiple operations sequence अनेक ऑपरेशन्सचा क्रम तपासणे	Inputs: (5 + 3) × 2	16	16	Pass (यशस्वी)

2.4 ॲक्सेप्टन्स टेस्टिंग (Acceptance Testing)

ॲक्सेप्टन्स टेस्टिंग हा सॉफ्टवेअर टेस्टिंगचा अंतिम टप्पा आहे. ही टेस्टिंग सिस्टीम टेस्टिंगनंतर आणि प्रॉडक्शनमध्ये रिलीज करण्यापूर्वी केली जाते. याचा मुख्य उद्देश म्हणजे सॉफ्टवेअर बिझनेस रिक्वायरमेंट्स आणि युजरच्या अपेक्षा पूर्ण करते का हे तपासणे. इंटिग्रेशन टेस्टिंग किंवा सिस्टीम टेस्टिंग मध्ये तांत्रिक अचूकतेवर भर दिला जातो, तर ॲक्सेप्टन्स टेस्टिंगमध्ये सॉफ्टवेअर प्रत्यक्ष वापरासाठी योग्य आहे का हे पडताळले जाते. यामध्ये डिलिव्हर केलेले सॉफ्टवेअर कॉन्ट्रॅक्टमधील अटी पूर्ण करते का आणि ऑपरेशनल वापरासाठी तयार आहे का हे निश्चित केले जाते. ॲक्सेप्टन्स टेस्टिंगमुळे सिस्टीम प्रत्यक्ष वापराच्या (real-world) वातावरणात आपला उद्देश पूर्ण करते याचा औपचारिक पुरावा मिळतो. ही टेस्टिंग सिस्टीम टेस्टिंगनंतर आणि प्रॉडक्ट डिलिव्हरीपूर्वी केली जाते, जेणेकरून सॉफ्टवेअर एंड-युजर्स किंवा क्लायंट्सच्या अपेक्षांनुसार आहे का हे सुनिश्चित करता येते. ही टेस्टिंग सहसा एंड-युजर्स, क्लायंट्स किंवा स्वतंत्र क्वालिटी ॲश्युरन्स टीम कडून केली

जाते. यात मुख्यतः ब्लॅक-बॉक्स टेस्टिंग टेक्निक्स वापरल्या जातात, ज्यामध्ये रिअल-वर्ल्ड वर्कफ्लोजनुसार सिस्टीमचे वर्तन तपासले जाते.

ॲक्सेप्टन्स टेस्टिंगमध्ये खालील दोन प्रकारांचा समावेश होतो:

- फंक्शनल ॲक्सेप्टन्स टेस्टिंग – फिचर्स आणि वर्कफ्लोज योग्य आहेत का हे तपासते.
- नॉन-फंक्शनल ॲक्सेप्टन्स टेस्टिंग – युजेबिलिटी, परफॉर्मन्स, रिलायबिलिटी आणि सिक्युरिटी यांचे मूल्यमापन करते.

ॲक्सेप्टन्स टेस्टिंग प्रोसेस (Acceptance Testing Process)

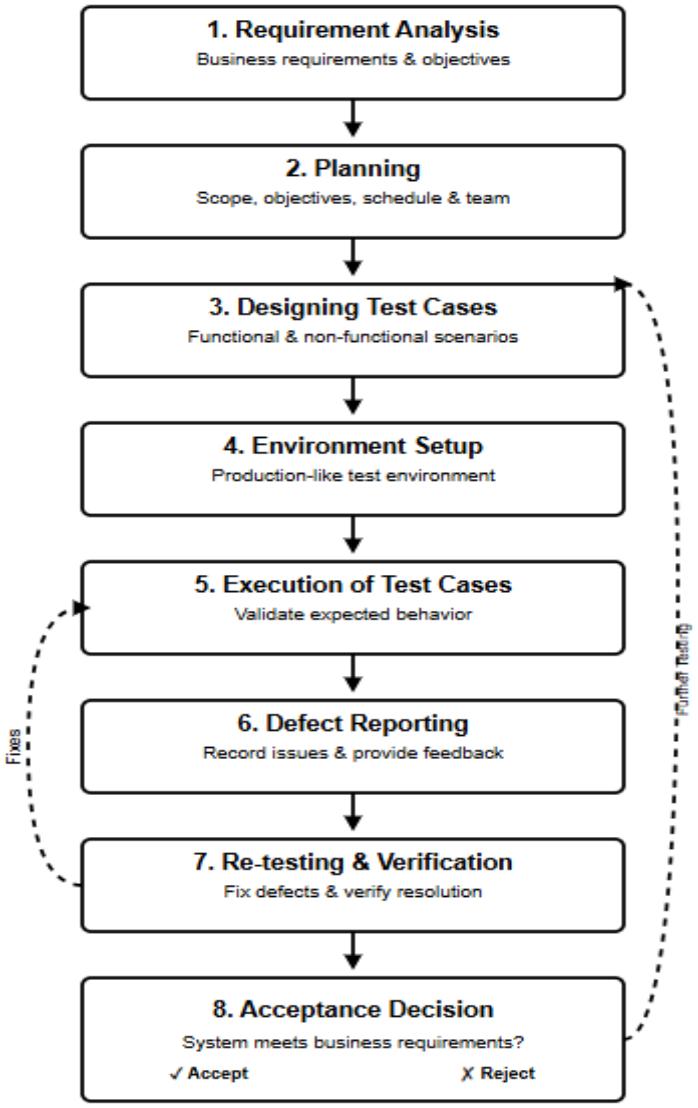

Fig 2.9: ॲक्सेप्टन्स टेस्टिंग प्रोसेस (Acceptance Testing Process)

1. **रिक्वायरमेंट ॲनालिसिस (Requirement Analysis):** बिझनेस रिक्वायरमेंट्स आणि उद्दिष्टे समजून घेणे. टेस्ट करावयाच्या महत्त्वाच्या वर्कफ्लोज, फिचर्स आणि परफॉर्मन्स निकष ओळखणे.
2. **प्लॅनिंग (Planning):** ॲक्सेप्टन्स टेस्टिंगचा स्कोप, उद्दिष्टे आणि वेळापत्रक ठरवणे. टेस्टिंग कोण करणार हे ठरवणे (एंड-युजर्स, क्लायंट्स किंवा QA टीम).
3. **टेस्ट केस डिझाईन करणे (Designing Test Cases):** रिअल-वर्ल्ड वर्कफ्लोजवर आधारित टेस्ट सिनारिओ आणि टेस्ट केसेस तयार करणे. खालील प्रकारच्या टेस्ट केसेस समाविष्ट करणे:
   - फंक्शनल टेस्ट केसेस (फिचर्स, वर्कफ्लोज)
   - नॉन-फंक्शनल टेस्ट केसेस (परफॉर्मन्स, युजेबिलिटी, सिक्युरिटी)
4. **एन्व्हायर्नमेंट सेटअप (Environment Setup):** सिस्टीम ज्या वातावरणात वापरली जाणार आहे, त्यासारखेच रिअल-वर्ल्ड टेस्टिंग एन्व्हायर्नमेंट तयार करणे.

5. **टेस्ट केस एक्झिक्युशन (Execution of Test Cases):** एंड-युजर्स किंवा QA टीम टेस्ट केसेस रन करतात. सिस्टीम सामान्य वापराच्या परिस्थितीत अपेक्षेप्रमाणे वागत आहे का हे तपासले जाते.
6. **डिफेक्ट रिपोर्टिंग (Defect Reporting):** आढळलेल्या समस्या, बग्स किंवा अपेक्षित निकालांपासून झालेल्या विचलनांची नोंद करणे व डेव्हलपमेंट टीमला सुधारण्यासाठी फीडबॅक देणे.
7. **री-टेस्टिंग आणि व्हेरिफिकेशन (Re-testing and Verification):** डेव्हलपर्स डिफेक्ट्स दुरुस्त करतात. टेस्टर पुन्हा टेस्ट करून सर्व समस्या पूर्णपणे सुटल्या आहेत का हे तपासतात.
8. **ॲक्सेप्टन्स निर्णय (Acceptance Decision):** सर्व टेस्ट केसेस पास झाल्या आणि सिस्टीम बिझनेस रिक्वायरमेंट्स पूर्ण करत असल्यास सिस्टीम औपचारिकरित्या स्वीकारली जाते (Accepted). अन्यथा, आवश्यक दुरुस्त्या करून पुन्हा टेस्टिंग केली जाते.

उदाहरण (Example):स्टँडर्ड कॅल्क्युलेटर ॲप्लिकेशनमध्ये ॲक्सेप्टन्स टेस्टिंग करताना खालील बाबी तपासल्या जातात:

• युजर्सना Addition, Subtraction, Multiplication आणि Division करता येते का
• Memory functions वापरता येतात का
• Entries clear करणे अपेक्षेप्रमाणे होते का
• निकालांची अचूकता, वापरण्याची सुलभता (Ease of Use) आणि रिस्पॉन्स टाइम (Response Time) योग्य आहे का.

हे सर्व क्लायंटच्या गरजांनुसार तपासले जाते.

ॲक्सेप्टन्स टेस्टिंगचे प्रकार (Types of Acceptance Testing)

1. **अल्फा टेस्टिंग (Alpha Testing):** डेव्हलपरच्या साईटवर नियंत्रित वातावरणात केली जाते. एंड-युजर्स सहभागी होऊ शकतात, पण डेव्हलपर्स निरीक्षण करतात आणि इश्यूज लॉग करतात.
2. **बीटा टेस्टिंग (Beta Testing):** ग्राहक किंवा एंड-युजरच्या वातावरणात केली जाते. रिअल-वर्ल्ड डेटा आणि परिस्थिती वापरून टेस्टिंग केली जाते. ऑफिशियल रिलीजपूर्वी फायनल व्हॅलिडेशन देते.

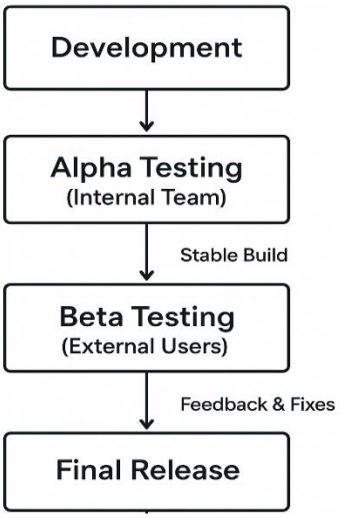

Fig 2.10: अल्फा टेस्टिंग ते बीटा टेस्टिंग प्रक्रिया (Alpha Testing to Beta Testing Process)

2.4.1 अल्फा टेस्टिंग (Alpha Testing)

अल्फा टेस्टिंग हा इन-हाऊस ॲक्सेप्टन्स टेस्टिंगचा प्रकार आहे, जो सॉफ्टवेअर एंड-युजर्सकडे रिलीज करण्यापूर्वी डेव्हलपरच्या साईटवर केला जातो. ही टेस्टिंग डेव्हलपर्स किंवा समर्पित टेस्टिंग (QA) टीम कडून केली जाते. याचा मुख्य उद्देश म्हणजे नियंत्रित वातावरणात (controlled environment) सॉफ्टवेअरमधील बग्स आणि समस्या लवकर शोधणे. अल्फा टेस्टिंगमध्ये सॉफ्टवेअरचा रिअल-वर्ल्ड वापर सिम्युलेट केला जातो, पण ही टेस्टिंग नियंत्रित परिस्थितीत केली जाते. यामुळे सॉफ्टवेअर बिझनेस रिक्वायरमेंट्स पूर्ण करते का आणि अपेक्षेप्रमाणे वर्तन करते का हे तपासले जाते. यामध्ये फंक्शनल टेस्टिंग आणि नॉन-फंक्शनल टेस्टिंग (परफॉर्मन्स, युजेबिलिटी, रिलायबिलिटी) दोन्हींचा समावेश असतो. अल्फा टेस्टिंग सहसा बीटा टेस्टिंगपूर्वी केली जाते, ज्यामुळे डेव्हलपमेंट प्रक्रियेच्या सुरुवातीच्या टप्प्यातच समस्या ओळखून त्या दुरुस्त करता येतात.

अल्फा टेस्टिंगची प्रक्रिया (Alpha Testing Process)

1. **प्लॅनिंग (Planning):** टेस्टिंगचा स्कोप, उद्दिष्टे आणि टेस्ट सिनारिओज निश्चित करणे.
2. **टेस्ट केस डिझाईन (Designing Test Cases):** इनपुट्स, अपेक्षित आउटपुट्स आणि टेस्ट स्टेप्स निश्चित करणे.
3. **टेस्ट एक्झिक्युशन (Execution):** टेस्टर्स नियंत्रित वातावरणात टेस्ट केसेस रन करतात.
4. **बग रिपोर्टिंग (Bug Reporting):** टेस्टिंगदरम्यान आढळलेले डिफेक्ट्स आणि इश्यूज लॉग करणे.
5. **दुरुस्ती आणि री-टेस्टिंग (Fixing and Retesting):** डेव्हलपर्स बग्स दुरुस्त करतात आणि टेस्टर्स पुन्हा टेस्ट करून सिस्टीम स्थिर (stable) झाली आहे का हे तपासतात.

उदाहरण – स्टँडर्ड कॅल्क्युलेटरसाठी अल्फा टेस्टिंग

परिस्थिती (Scenario): समजा, एक कंपनी स्टँडर्ड कॅल्क्युलेटर ॲप्लिकेशन विकसित करते.

अल्फा टेस्टिंगदरम्यान: इंटरनल टेस्टर्स (कर्मचारी किंवा QA टीम) खालील फिचर्स तपासतात:

- ॲडिशन, सब्ट्रॅक्शन, मल्टिप्लिकेशन, डिव्हिजन (Addition, Subtraction, Multiplication, Division)
- परसेंटेज कॅल्क्युलेशन (Percentage calculation)
- मेमरी फंक्शन्स (Memory functions)

तसेच खालील नॉन-फंक्शनल बाबी तपासल्या जातात:

- परफॉर्मन्स (गणनांचा वेग)
- युजेबिलिटी (बटणांची मांडणी, इनपुट देण्याची सोय)
- रिलायबिलिटी (मोठ्या संख्यांचे हँडलिंग, अवैध इनपुट)

काही बग्स जसे की काही ऑपरेशन्ससाठी चुकीचे निकाल आणि सलग अनेक गणना करताना response time मध्ये विलंब हे बग्स अल्फा टेस्टिंगमध्येच नोंदवले जातात आणि दुरुस्त केले जातात.

अल्फा टेस्टिंगसाठी नमुना टेस्ट केसेस (Test Cases)

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
AT001	Test addition बेरीज तपासणे	Input: 5 + 3, press "="	8	8	Pass (यशस्वी)
AT002	Test division by zero शून्याने भागाकार तपासणे	Input: 10 ÷ 0, press "="	Error / “Cannot divide by zero ”त्रुटी / “शून्याने भागाकार करता येत नाही”	Error / “Cannot divide by zero”	Pass (यशस्वी)
AT003	Test multiplication गुणाकार तपासणे	Input: 6 × 7, press "="	42	41	Fail (अयशस्वी)
AT004	Test usability वापर सुलभता तपासणे	Check button layout and ease of input बटण लेआउट आणि इनपुटची सुलभता तपासणे	Easy to use, all buttons accessible वापरण्यास सोपे, सर्व बटणे सहज उपलब्ध	Easy to use, all buttons accessible	Pass (यशस्वी)

2.4.2 बीटा टेस्टिंग (Beta Testing)

बीटा टेस्टिंग हा सॉफ्टवेअर टेस्टिंगचा एक प्रकार आहे, जो अल्फा टेस्टिंगनंतर आणि अंतिम रिलीजपूर्वी केला जातो. ही टेस्टिंग संस्थेबाहेरील प्रत्यक्ष युजर्सकडून (real users) आणि प्रत्यक्ष वापराच्या वातावरणात (real environment) केली जाते. याचा मुख्य उद्देश म्हणजे अल्फा टेस्टिंगमध्ये न सापडलेल्या बग्स किंवा समस्या शोधणे. बीटा टेस्टिंगमध्ये सॉफ्टवेअरचा वापर खऱ्या परिस्थितीत केला जातो. त्यामुळे सॉफ्टवेअरची फंक्शनॅलिटी, परफॉर्मन्स आणि युजर एक्सपीरियन्स यांचे योग्य मूल्यमापन करता येते. बीटा टेस्टर्सकडून मिळणारा फीडबॅक वापरून डेव्हलपर्स अंतिम सुधारणा (final improvements) करतात. बीटा टेस्टिंगमध्ये सहसा: फंक्शनल टेस्टिंग, नॉन-फंक्शनल टेस्टिंग (युजेबिलिटी, परफॉर्मन्स, रिलायबिलिटी) यांचा समावेश असतो, परंतु हे सर्व प्रत्यक्ष युजरच्या वातावरणात केले जाते.

बीटा टेस्टिंगचा उद्देश (Purpose):

- अल्फा टेस्टिंगमध्ये न सापडलेल्या त्रुटी शोधणे
- प्रत्यक्ष वापरात सॉफ्टवेअरचे वर्तन तपासणे
- युजर्सकडून फीडबॅक, सूचना आणि अनुभव मिळवणे
- सॉफ्टवेअर व्यावसायिक रिलीजसाठी तयार आहे का हे सुनिश्चित करणे

बीटा टेस्टिंगची प्रक्रिया (Process of Beta Testing)

1. प्लॅनिंग (Planning): उद्दिष्टे, टार्गेट युजर ग्रुप आणि टेस्टिंग कालावधी ठरवणे.
2. बीटा व्हर्जन रिलीज करणे (Release Beta Version): जवळजवळ पूर्ण झालेले सॉफ्टवेअर (Beta version) निवडक बाह्य युजर्सना उपलब्ध करून देणे.
3. युजर्सद्वारे टेस्ट एक्झिक्युशन (Execution by Users): युजर्स प्रत्यक्ष परिस्थितीत सॉफ्टवेअर वापरतात. फंक्शनल ऑपरेशन्स करतात आणि युजेबिलिटी अनुभवतात.
4. फीडबॅक संकलन (Feedback Collection): युजर्स बग्स, एरर्स किंवा सुधारणा सूचना नोंदवतात.
5. बग फिक्सिंग आणि सुधारणा (Bug Fixing and Improvements): डेव्हलपर्स फीडबॅकचे विश्लेषण करून महत्त्वाच्या समस्या दुरुस्त करतात.
6. अंतिम रिलीज (Final Release): सर्व महत्त्वाच्या समस्या सोडवल्यानंतर सॉफ्टवेअर ऑफिशियल लॉन्चसाठी तयार केले जाते.

उदाहरण – स्टँडर्ड कॅल्क्युलेटरसाठी बीटा टेस्टिंग

परिस्थिती (Scenario):अल्फा टेस्टिंग पूर्ण झाल्यानंतर कॅल्क्युलेटर ॲप्लिकेशन चे बीटा व्हर्जन काही निवडक बाह्य युजर्सना दिले जाते, जे ते घरी, ऑफिसमध्ये किंवा मोबाईल/पीसीवर प्रत्यक्ष वापरतात.

बीटा टेस्टिंगमध्ये युजर्स तपासतात: ॲडिशन, सब्ट्रॅक्शन, मल्टिप्लिकेशन, डिव्हिजन, मोठ्या संख्यांवरील गणना, सतत वापर केल्यावर परफॉर्मन्स, ॲप वापरण्याची सुलभता,युजर्सकडून मिळालेल्या फीडबॅकनुसार अंतिम सुधारणा करून सॉफ्टवेअर रिलीज केले जाते.

बीटा टेस्टिंगसाठी नमुना टेस्ट केसेस (Test Cases)

Test Case ID (टेस्ट केस आयडी)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
BT001	Test addition बेरीज तपासणे	45 + 55 =	100	100	Pass (यशस्वी)
BT002	Test division by zero शून्याने भागाकार तपासणे	12 ÷ 0 =	Error / “Cannot divide by zero “त्रुटी / “शून्याने भागाकार करता येत नाही”	Error / “Cannot divide by zero”	Pass (यशस्वी)

BT003	Test large number calculation मोठ्या संख्येची गणना तपासणे	999999 × 999999 =	999998000001	999998000001	Pass (यशस्वी)
BT004	Test usability वापर सुलभता तपासणे	Check button layout and ease of input बटण लेआउट आणि इनपुटची सुलभता तपासणे	Easy to use, all buttons accessible वापरण्यास सोपे, सर्व बटणे सहज उपलब्ध	Some buttons hard to read काही बटणे वाचायला कठीण	Fail (अयशस्वी)

Table 2.3: अल्फा टेस्टिंग वि. बीटा टेस्टिंग (Comparison between Alpha Testing, Beta Testing)

निकष (Criteria)	**अल्फा टेस्टिंग (Alpha Testing)**	**बीटा टेस्टिंग (Beta Testing)**
व्याख्या (Definition)	युजर्सकडे रिलीज करण्यापूर्वी सॉफ्टवेअरची गुणवत्ता आंतरिकरित्या तपासण्यासाठी केली जाणारी टेस्टिंग.	प्रत्यक्ष वापराच्या वातावरणात बाह्य युजर्सकडून सॉफ्टवेअरचे मूल्यमापन करण्यासाठी केली जाणारी टेस्टिंग.
कोण करते (Performed By)	आंतरिक टीम (डेव्हलपर्स, QA टेस्टर्स, कर्मचारी).	बाह्य युजर्स / ग्राहक.
पर्यावरण (Environment)	नियंत्रित वातावरण (डेव्हलपमेंट / टेस्टिंग लॅब).	रिअल-वर्ल्ड वातावरण (युजरची उपकरणे, नेटवर्क्स, OS व्हर्जन्स).
फोकस (Focus)	बग्स लवकर शोधणे, बिझनेस रिक्वायरमेंट्स आणि फंक्शनॅलिटी तपासणे.	परफॉर्मन्स, युजेबिलिटी, रिलायबिलिटी आणि युजर समाधान तपासणे.
टेस्टिंगचा प्रकार (Type of Testing)	फंक्शनल + नॉन-फंक्शनल (परफॉर्मन्स, युजेबिलिटी, रिलायबिलिटी).	फंक्शनल + नॉन-फंक्शनल (प्रत्यक्ष वापराच्या परिस्थितीत).
सापडणारे डिफेक्ट्स (Defects Found)	मोठ्या फंक्शनल चुका, मिसिंग फिचर्स, परफॉर्मन्स बॉटलनेक्स.	उरलेले डिफेक्ट्स, युजेबिलिटी इश्यूज, क्रॅशेस, कम्पॅटिबिलिटी प्रॉब्लेम्स.
युजर सहभाग (User Involvement)	बाह्य युजर्सचा सहभाग नाही.	प्रत्यक्ष बाह्य युजर्स सक्रियपणे फीडबॅक देतात.
उद्देश (Goal)	सॉफ्टवेअर बीटा रिलीजसाठी पुरेसे स्थिर आहे का हे सुनिश्चित करणे.	सॉफ्टवेअर व्यावसायिक / अधिकृत रिलीजसाठी तयार आहे का हे सुनिश्चित करणे.
वेळ (Timing)	बीटा टेस्टिंगपूर्वी केली जाते.	अल्फा टेस्टिंगनंतर केली जाते.
उदाहरण (Example)	कॅल्क्युलेटर अॅप आंतरिकरित्या Add, Subtract, Multiply, Divide आणि परफॉर्मन्ससाठी टेस्ट केला जातो.	कॅल्क्युलेटर अॅप बाह्य युजर्सकडून विविध डिव्हाइसेस/OS वर वापरला जातो; युजेबिलिटी, क्रॅशेस आणि स्पीडवर फीडबॅक घेतला जातो.

2.5 स्पेसिअल टेस्टिंग (Special Testing):

2.5.1 परफॉर्मन्स टेस्टिंग (Performance Testing) – लोड टेस्टिंग (Load Testing) आणि स्ट्रेस टेस्टिंग (Stress Testing)

परफॉर्मन्स टेस्टिंग (Performance Testing) ही नॉन-फंक्शनल टेस्टिंगची एक श्रेणी आहे, जी विविध परिस्थितींमध्ये सिस्टीमचा वेग, स्केलेबिलिटी, स्थिरता आणि विश्वासार्हता तपासण्यावर लक्ष केंद्रित करते. परफॉर्मन्स टेस्टिंगमधील दोन महत्त्वाचे प्रकार म्हणजे लोड टेस्टिंग (Load Testing) आणि स्ट्रेस टेस्टिंग (Stress Testing). या चाचण्या सिस्टीम वास्तविक (real-world) वापराच्या गरजा कार्यक्षमतेने हाताळू शकते का हे सुनिश्चित करतात.

परफॉर्मन्स टेस्टिंगची वैशिष्ट्ये (Characteristics of Performance Testing):

1. वेग (Speed): सिस्टीम वापरकर्त्यांच्या विनंत्यांना (user requests) किती लवकर प्रतिसाद देते (response time) हे मोजते.
2. स्केलेबिलिटी (Scalability): वाढत्या वापरकर्त्यांची संख्या किंवा व्यवहार (transactions) सिस्टीम कार्यक्षमतेने हाताळू शकते का हे तपासते.
3. स्थिरता (Stability): दीर्घकाळ वापरात असतानाही सिस्टीम सातत्याने आणि सुरळीत चालू राहते का याची खात्री करते.
4. विश्वासार्हता (Reliability): जास्त लोडसह विविध परिस्थितींमध्ये ॲप्लिकेशन योग्य प्रकारे कार्य करते का हे तपासते.

परफॉर्मन्स टेस्टिंगचे प्रकार (Types of Performance Testing):

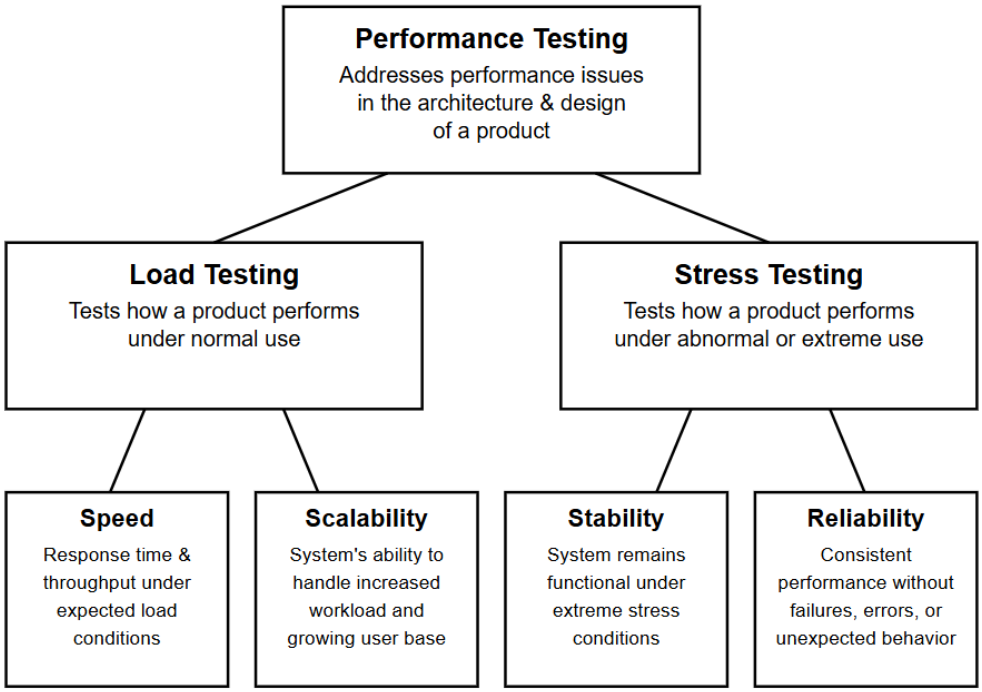

Fig 2.11: परफॉर्मन्स टेस्टिंगचे प्रकार (Types of Performance Testing)

1. **लोड टेस्टिंग (Load Testing):** अपेक्षित कामाच्या लोडखाली (Expected Workload) सिस्टीमचे वर्तन तपासते. यामध्ये एकाच वेळी (concurrent) वापरकर्त्यांची संख्या किंवा व्यवहार (transactions) हळूहळू वाढवले जातात, जोपर्यंत सिस्टीमची मर्यादा (threshold) गाठली जात नाही.
2. **स्ट्रेस टेस्टिंग (Stress Testing):** सिस्टीमची सामान्य क्षमतेपेक्षा जास्त लोड देऊन तिचा ब्रेकिंग पॉईंट (breaking point) शोधला जातो. तसेच सिस्टीम अपयशानंतर (failure) कशी सावरते (recovery) हेही तपासले जाते.
3. **एन्ड्युरन्स / सोक टेस्टिंग (Endurance / Soak Testing):** दीर्घ कालावधीसाठी सिस्टीमची स्थिरता (stability) आणि परफॉर्मन्स तपासते. यामुळे मेमरी लीक (memory leak) किंवा परफॉर्मन्स घट यासारख्या समस्या ओळखता येतात.

4. **स्पाइक टेस्टिंग (Spike Testing):** अचानक आणि खूप जास्त प्रमाणात लोड वाढल्यास सिस्टीम कशी प्रतिक्रिया देते हे तपासते. उदा. अचानक मोठ्या प्रमाणावर वापरकर्ते लॉगिन झाल्यास सिस्टीम कशी वागते.
5. **स्केलेबिलिटी टेस्टिंग (Scalability Testing):** लोड वाढवला किंवा कमी केला असता ॲप्लिकेशन कार्यक्षमतेने स्केल अप किंवा स्केल डाउन होते का हे ठरवते.

a. लोड टेस्टिंग (Load Testing)

लोड टेस्टिंग हा परफॉर्मन्स टेस्टिंगचा एक प्रकार आहे, ज्यामध्ये सिस्टीमवर हळूहळू लोड वाढवून ती तिच्या मर्यादेपर्यंत (Threshold Value) पोहोचते का हे तपासले जाते. यासाठी एकाच वेळी वापरकर्त्यांची संख्या (Concurrent Users), व्यवहार (Transactions) किंवा डेटा व्हॉल्यूम सातत्याने वाढवले जातात आणि त्या परिस्थितीत ॲप्लिकेशनचे वर्तन निरीक्षण केले जाते. लोड टेस्टिंगमुळे ॲप्लिकेशन अपेक्षित ग्राहक लोडखाली योग्य प्रकारे कार्य करते की नाही हे सुनिश्चित होते.

या चाचणीदरम्यान खालील घटक तपासले जातात:

- प्रतिसाद वेळ (Response Time)
- थ्रूपुट (Throughput)
- CPU वापर
- मेमरी वापर
- सिस्टीमची स्थिरता (Stability)

जेव्हा अनेक वापरकर्ते किंवा प्रोसेसेस एकाच वेळी सिस्टीम वापरतात तेव्हा ती कशी वागते हे लोड टेस्टिंगद्वारे समजते. ही चाचणी सहसा नियंत्रित वातावरणात (Controlled Environment) केली जाते. ठरवलेली सर्व टेस्ट केस दिलेल्या वेळेत आणि कोणतीही त्रुटी न येता पूर्ण झाल्यास लोड टेस्टिंग यशस्वी मानली जाते. अनेक वापरकर्ते आणि व्यवहार सिम्युलेट करण्यासाठी ऑटोमेशन टूल्सचा मोठ्या प्रमाणावर वापर केला जातो.

उदाहरण: सणासुदीच्या सेलदरम्यान ई-कॉमर्स वेबसाईट

परिस्थिती (Scenario):सणासुदीच्या सेलदरम्यान एका ई-कॉमर्स वेबसाईटवर एकाच वेळी 10,000 ग्राहक खरेदी करतील अशी अपेक्षा आहे.

प्रक्रिया (Process): QA इंजिनिअर्स 10,000 समकालीन (Concurrent) वापरकर्ते सिम्युलेट करतात, जे:

- वस्तू कार्टमध्ये टाकतात
- डिस्काउंट कूपन वापरतात
- पेमेंट करतात

निरीक्षण (Observation):

- वेबसाईट सुरळीत प्रतिसाद देते
- पेजेस 3 सेकंदांच्या आत लोड होतात
- पेमेंट कोणत्याही त्रुटीशिवाय पूर्ण होते

निकाल (Result):अपेक्षित ग्राहक लोडखाली सिस्टीम नीट कार्य करत असल्यामुळे लोड टेस्टिंग यशस्वी ठरते.

टेस्ट केस (Test Cases):

Test Case ID (टेस्ट केस आयडी)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
LT01	Validate website under expected user load	Simulate 10,000 concurrent users browsing and adding items to cart	Pages load within ≤ 3 sec, no errors	Pages loaded within 2.8 sec	Pass (यशस्वी)

	अपेक्षित युजर लोडखाली वेबसाईट तपासणे	10,000 एकत्रित युजर्स ब्राउझिंग व कार्टमध्ये आयटम जोडणे सिम्युलेट करणे	पेजेस ≤ 3 सेकंदात लोड होणे, कोणतीही त्रुटी नाही	पेजेस 2.8 सेकंदात लोड झाले	
LT02	Verify coupon application under load लोडखाली कूपन ॲप्लिकेशन तपासणे	5,000 users apply discount coupons simultaneously 5,000 युजर्स एकाच वेळी डिस्काउंट कूपन वापरणे	Coupons applied correctly without errors कूपन योग्यरित्या लागू होणे, त्रुटी नाही	Coupons applied successfully कूपन यशस्वीपणे लागू झाले	Pass (यशस्वी)
LT03	Check checkout stability चेकआउट प्रक्रियेची स्थिरता तपासणे	3,000 users proceed to checkout at the same time 3,000 युजर्स एकाच वेळी चेकआउटकडे जाणे	Checkout completes smoothly चेकआउट प्रक्रिया सुरळीत पूर्ण होणे	Checkout process completed चेकआउट प्रक्रिया पूर्ण झाली	Pass. (यशस्वी)
LT04	Validate payment under concurrent users एकत्रित युजर्सखाली पेमेंट तपासणे	2,000 users make payments simultaneously 2,000 युजर्स एकाच वेळी पेमेंट करणे	Payments processed without failure or duplication पेमेंट कोणतीही अयशस्वीता किंवा डुप्लिकेशन शिवाय पूर्ण होणे	3% transactions delayed but recovered 3% व्यवहार उशिरा झाले पण पुनर्प्राप्त झाले	Pass (यशस्वी)

b. स्ट्रेस टेस्टिंग (Stress Testing)

स्ट्रेस टेस्टिंग हा नॉन-फंक्शनल परफॉर्मन्स टेस्टिंगचा एक प्रकार आहे, ज्यामध्ये सिस्टीमची स्थिरता (Stability) आणि मजबुती (Robustness) तपासण्यासाठी तिला अत्यंत जड लोड किंवा संसाधनांच्या कमतरतेखाली (Resource Exhaustion) चालवले जाते. ही चाचणी सिस्टीमच्या सामान्य कार्यक्षमतेपेक्षा (Normal Capacity) जास्त परिस्थितीत केली जाते, जेणेकरून सिस्टीमचा ब्रेकिंग पॉइंट (Breaking Point) ओळखता येईल.

लोड टेस्टिंगपेक्षा वेगळे म्हणून, स्ट्रेस टेस्टिंगमध्ये सिस्टीमची मर्यादा ओलांडून तिला "ताण" दिला जातो. उदा.: मेमरी कमी पडणे, डिस्क स्पेस संपणे, नेटवर्क कोंडी (Network Congestion), अतिशय जास्त ट्रॅफिक या चाचणीचा मुख्य उद्देश सामान्य परफॉर्मन्स तपासणे नसून खालील गोष्टी तपासणे हा असतो:

- एरर हँडलिंग (Error Handling)
- सिस्टीमची मजबुती (Robustness)
- उपलब्धता (Availability)
- फेल्युअरनंतर रिकव्हरी क्षमता (Recovery Capability)

स्ट्रेस टेस्टिंगमुळे सिस्टीम अचानक क्रॅश न होता ग्रेसफुली फेल (Graceful Failure) होते का, तसेच महत्त्वाचा डेटा हरवत नाही ना, हे सुनिश्चित केले जाते. म्हणूनच स्ट्रेस टेस्टिंगला कधी कधी फॅटिग टेस्टिंग (Fatigue Testing) असेही म्हणतात, कारण सिस्टीम "तुटेपर्यंत" तिच्यावर ताण दिला जातो.

उदाहरण: सणासुदीच्या सेलदरम्यान ई-कॉमर्स वेबसाईट

परिस्थिती (Scenario): याच ई-कॉमर्स वेबसाईटवर आता एकाच वेळी 50,000 वापरकर्ते येतील अशी चाचणी केली जाते, जी अपेक्षित लोडपेक्षा खूप जास्त आहे.

प्रक्रिया (Process): QA इंजिनिअर्स खालील क्रिया मोठ्या प्रमाणावर सिम्युलेट करतात: ब्राउझिंग, चेकआउट, पेमेंट

सर्व्हरवर अतिशय जास्त रिक्वेस्ट्स पाठवून त्याची क्षमता तपासली जाते.

निरीक्षण (Observation):

- वेबसाईट हळू प्रतिसाद देऊ लागते
- काही पेमेंट व्यवहार अयशस्वी होतात
- 55,000 वापरकर्त्यांनंतर सिस्टीम पूर्णपणे क्रॅश होते

निकाल (Result):

यामुळे सिस्टीमचा ब्रेकिंग पॉइंट समजतो आणि ती संसाधनांच्या कमतरतेत कशी वागते हे स्पष्ट होते. या माहितीनुसार डेव्हलपर्स:

- एरर हँडलिंग सुधारू शकतात
- ऑटो-स्केलिंग लागू करू शकतात
- सर्व्हर ऑप्टिमायझेशन करू शकतात

टेस्ट केस (Test Cases):

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
ST01	Identify system breaking point सिस्टीमचा ब्रेकिंग पॉइंट ओळखणे	Increase users from 10,000 → 55,000 युजर्स 10,000 पासून 55,000 पर्यंत वाढवणे	System should degrade gracefully before crash क्रॅश होण्यापूर्वी सिस्टीम हळूहळू स्लो व्हावी	System slowed at 50,000, crashed at 55,000 50,000 वर सिस्टीम स्लो झाली, 55,000 वर क्रॅश झाली	Fail (अयशस्वी)
ST02	Stress test payment gateway. पेमेंट गेटवेवर स्ट्रेस टेस्ट करणे	Simulate 10,000+ payments simultaneously 10,000 पेक्षा जास्त पेमेंट्स एकाच वेळी सिम्युलेट करणे	Minor failures acceptable but no data loss किरकोळ अपयश चालतील पण डेटा लॉस नको	Some failures, no data loss काही अपयश, पण डेटा लॉस नाही	Pass (यशस्वी)
ST03	Test system at extreme resource usage अत्यंत रिसोर्स वापरावर सिस्टीम तपासणे	Push CPU & memory to 95%+ CPU व मेमरी 95% पेक्षा जास्त वापरणे	System should remain stable or deny requests gracefully सिस्टीम स्थिर राहावी किंवा विनंत्या योग्यरित्या नाकाराव्यात	Server slowed down, queued requests सर्व्हर स्लो झाला, रिक्वेस्ट्स क्यू मध्ये गेल्या	Pass (यशस्वी)
ST04	Verify error handling under sudden spike	Increase traffic 5x suddenly ट्रॅफिक अचानक 5 पट वाढवणे	System should log errors & show user-friendly message	Error logs captured, friendly	Pass (यशस्वी)

	अचानक ट्रॅफिक वाढल्यावर एरर हँडलिंग तपासणे		एरर लॉग होणे व युजर-फ्रेंडली मेसेज दाखवणे	message displayed एरर लॉग झाले, युजर-फ्रेंडली मेसेज दाखवला	
ST05	Test recovery after crash क्रॅशनंतर सिस्टीम रिकव्हरी तपासणे	Restart system after overload ओव्हरलोडनंतर सिस्टीम रीस्टार्ट करणे	System should recover and resume normal operations सिस्टीम पुन्हा नॉर्मल ऑपरेशनमध्ये यावी	System recovered in 2 mins सिस्टीम 2 मिनिटांत रिकव्हर झाली	Pass (यशस्वी)

Table 2.4: लोड टेस्टिंग वि. स्ट्रेस टेस्टिंग (Comparison Between Load Testing and Stress Testing)

Criteria(निकष)	Load Testing(लोड टेस्टिंग)	Stress Testing(स्ट्रेस टेस्टिंग)
Definition (व्याख्या)	Verifies system performance under expected load conditions. अपेक्षित लोड परिस्थितीत सिस्टीमची कार्यक्षमता तपासते.	Tests system performance under extreme or beyond-limit load conditions. अत्यंत किंवा मर्यादेपेक्षा जास्त लोडखाली सिस्टीमची कार्यक्षमता तपासते.
Objective (उद्दिष्ट)	To measure response time, throughput, and stability under normal workload. सामान्य वर्कलोडखाली प्रतिसाद वेळ, थ्रूपुट आणि स्थिरता मोजणे.	To find the breaking point, robustness, and error-handling ability of the system. सिस्टीमचा ब्रेकिंग पॉइंट, मजबुती आणि एरर-हँडलिंग क्षमता शोधणे.
Focus (मुख्य लक्ष)	Ensures system can handle expected concurrent users, transactions, or data. अपेक्षित युजर्स, ट्रान्झॅक्शन्स किंवा डेटा हाताळता येतो का हे पाहते.	Ensures system remains stable under unexpected, peak, or resource-limited situations. अनपेक्षित, पीक किंवा मर्यादित संसाधन परिस्थितीत सिस्टीम स्थिर राहते का हे पाहते.
Environment (पर्यावरण)	Conducted in a controlled environment with gradually increasing load. हळूहळू वाढणाऱ्या लोडसह नियंत्रित वातावरणात केली जाते.	Conducted by pushing the system beyond its operational limits. सिस्टीमला तिच्या कार्यक्षमतेच्या मर्यादेपेक्षा जास्त ताण देऊन केली जाते.
Failure Tolerance (अपयश सहनशीलता)	Test passes if the system runs smoothly without errors under expected load. अपेक्षित लोडखाली त्रुटीशिवाय सिस्टीम सुरळीत चालली तर टेस्ट पास होते.	Test passes if the system fails gracefully without crashing or data loss. सिस्टीम क्रॅश किंवा डेटा लॉस न होता योग्यरित्या फेल झाली तर टेस्ट पास होते.
Key Metrics (मुख्य मेट्रिक्स)	Response time, throughput, resource utilization, scalability. प्रतिसाद वेळ, थ्रूपुट, संसाधन वापर, स्केलेबिलिटी.	Robustness, availability, error-handling, recovery after failure. मजबुती, उपलब्धता, एरर-हँडलिंग, अपयशानंतरची रिकव्हरी.

Examples (उदाहरणे)	10,000 users accessing an e-commerce website simultaneously; 1,000 documents print queue. 10,000 युजर्स एकाच वेळी ई-कॉमर्स वेबसाईट वापरणे.	50,000 users accessing the same website at once; Running software with almost zero memory. 50,000 युजर्स एकाच वेळी वेबसाईट वापरणे.
Outcome (निकाल)	Ensures the software meets performance requirements under normal usage. सामान्य वापरात सॉफ्टवेअर परफॉर्मन्स गरजा पूर्ण करते याची खात्री देते.	Identifies weak points and ensures software stability in extreme conditions. कमकुवत भाग ओळखते आणि अत्यंत परिस्थितीत सॉफ्टवेअरची स्थिरता सुनिश्चित करते.

2.5.2 रिग्रेशन टेस्टिंग (Regression Testing)

रिग्रेशन टेस्टिंग ही एक सॉफ्टवेअर टेस्टिंग तंत्र आहे, ज्याचा वापर कोडमध्ये केलेले अलीकडील बदल जसे की एन्हान्समेंट्स, पॅचेस किंवा बग फिक्सेस यांचा आधीपासून अस्तित्वात असलेल्या फंक्शनॅलिटीजवर नकारात्मक परिणाम झाला नाही ना, हे तपासण्यासाठी केला जातो. म्हणजेच, सॉफ्टवेअरमध्ये बदल केल्यानंतरही आधी विकसित व टेस्ट केलेली वैशिष्ट्ये योग्यरित्या कार्य करत आहेत का, याची खात्री रिग्रेशन टेस्टिंगद्वारे केली जाते. रिग्रेशन टेस्टिंग हे सॉफ्टवेअर टेस्टिंगमधील अत्यंत महत्त्वाचे प्रकारांपैकी एक आहे, विशेषतः इटरेटिव्ह (Iterative) आणि ॲजाईल (Agile) डेव्हलपमेंट वातावरणात. सॉफ्टवेअरमध्ये जेव्हा जेव्हा बदल केले जातात, तेव्हा त्या बदलांमुळे आधीचे कार्यरत फिचर्स बिघडण्याचा धोका असतो. रिग्रेशन टेस्टिंग हा धोका कमी करते, कारण यामध्ये आधी चालवलेल्या टेस्ट केसेस पुन्हा एकदा चालवून सिस्टीम स्थिर (Stable) आणि विश्वासार्ह (Reliable) आहे का, हे तपासले जाते. रिग्रेशन टेस्टिंग मॅन्युअली करता येते, परंतु प्रत्यक्षात कार्यक्षमता आणि कव्हरेज वाढवण्यासाठी (Selenium, JUnit, TestNG) यांसारखी ऑटोमेशन टूल्स मोठ्या प्रमाणावर वापरली जातात. ही प्रक्रिया साधारणपणे युनिट टेस्टिंगनंतर सुरू होते आणि सिस्टीम इंटिग्रेशन पूर्ण होईपर्यंत चालू राहते. सतत अपडेट होणाऱ्या CI/CD पाइपलाईन्समध्ये रिग्रेशन टेस्टिंग फारच आवश्यक असते. रिग्रेशन टेस्टिंगचे मुख्य उद्दिष्ट म्हणजे वारंवार कोड बदल होत असतानाही सॉफ्टवेअरची स्थिरता, विश्वासार्हता आणि गुणवत्ता कायम राखणे.

रिग्रेशन टेस्टिंगची वैशिष्ट्ये (Characteristics of Regression Testing)

1. सिस्टीमची स्थिरता सुनिश्चित करते: बग फिक्सेस, पॅचेस किंवा नवीन फिचर्स जोडल्यावरही सिस्टीम स्थिर राहते आणि नवीन त्रुटी निर्माण होत नाहीत, याची खात्री करते.
2. विश्वासार्हता वाढवते: आधीच्या फंक्शनॅलिटीज पुन्हा पुन्हा तपासल्यामुळे ॲप्लिकेशन वेगवेगळ्या वातावरणात आणि परिस्थितीत सातत्याने योग्यरित्या कार्य करेल, यावर विश्वास निर्माण होतो.
3. क्वालिटी ॲश्युरन्सला समर्थन देते: अपडेट्समुळे सॉफ्टवेअरची परफॉर्मन्स, युजेबिलिटी किंवा फंक्शनॅलिटी कमी होत नाही ना, हे तपासून एकूण गुणवत्तेचे संरक्षण करते.

उदाहरण: ऑनलाइन शॉपिंग ॲप्लिकेशन

परिस्थिती (Scenario):"Apply Coupon" या फिचरमधील एक बग डेव्हलपर्सने दुरुस्त केला आहे.

प्रक्रिया (Process):QA टीम रिग्रेशन टेस्टिंग चालवते आणि फक्त कूपन फिचरच नाही तर खालील गोष्टीही तपासते:

1. कार्टमध्ये आयटम जोडणे/काढणे
2. चेकआउट प्रक्रियेकडे जाणे
3. पेमेंट गेटवे इंटिग्रेशन
4. ऑर्डर समरी तयार होणे

निरीक्षण (Observation): कूपन फिचर योग्यरित्या कार्य करते आणि आधीपासून चालू असलेले सर्व मॉड्यूल्स अपेक्षेप्रमाणे काम करत राहतात.

टेस्ट केसेस (Test Cases):

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
RT-01	Verify adding/removing items from cart कार्टमध्ये आयटम जोडणे/काढणे तपासणे	Add 2 items, remove 1 2 आयटम जोडा, 1 काढा	Cart updates correctly कार्ट योग्यरीत्या अपडेट होणे	Cart updated correctly कार्ट योग्यरीत्या अपडेट झाले	Pass (यशस्वी)
RT-02	Verify coupon application कूपन लागू होणे तपासणे	Apply “DISCOUNT10” coupon “DISCOUNT10” कूपन वापरणे	10% discount applied 10% सूट लागू होणे	Coupon applied successfully कूपन यशस्वीपणे लागू झाले	Pass (यशस्वी)
RT-03	Verify checkout process चेकआउट प्रक्रिया तपासणे	Proceed to checkout after applying coupon कूपननंतर चेकआउटला जाणे	Checkout continues smoothly चेकआउट सुरळीत सुरू राहणे	Checkout stuck at “Payment” screen चेकआउट “Payment” स्क्रीनवर अडकले	Fail (अयशस्वी)
RT-04	Verify payment gateway integration पेमेंट गेटवे इंटिग्रेशन तपासणे	Make payment with valid credit card वैध क्रेडिट कार्डने पेमेंट करणे	Payment successful, receipt generated पेमेंट यशस्वी, पावती तयार होणे	Payment successful, receipt shown पेमेंट यशस्वी, पावती दाखवली	Pass (यशस्वी)
RT-05	Verify order summary generation ऑर्डर सारांश तयार होणे तपासणे	View final order summary after payment पेमेंटनंतर अंतिम ऑर्डर सारांश पाहणे	Displays item details, discount, total amount आयटम तपशील, सूट व एकूण रक्कम दाखवणे	Missing discount in summary सारांशात सूट दिसत नाही	Fail (अयशस्वी)

Table 2.5: रिग्रेशन टेस्टिंग वि. री-टेस्टिंग (Comparison between Regression Testing and Re-Testing)

Criteria(निकष)	Regression Testing(रिग्रेशन टेस्टिंग)	Re-Testing(री-टेस्टिंग)
Definition (व्याख्या)	Testing done to ensure that recent changes (bug fixes, enhancements, updates) have not broken the existing functionality. नवीन बदलांमुळे (बग फिक्स, एन्हान्समेंट, अपडेट्स) आधीची फंक्शनॅलिटी बिघडली नाही ना हे तपासणे.	Testing done to verify whether a specific defect that was fixed has been successfully resolved. दुरुस्त केलेला विशिष्ट बग योग्यरीत्या सुटला आहे का हे तपासणे.
Objective (उद्दिष्ट)	To ensure stability, reliability, and quality assurance of the entire application after modifications. बदलांनंतर संपूर्ण अॅप्लिकेशनची स्थिरता, विश्वसनीयता आणि गुणवत्ता सुनिश्चित करणे.	To check the correctness of the specific bug fix. विशिष्ट बग फिक्स योग्य आहे का ते तपासणे.
Scope (व्याप्ती)	Broad – includes testing of previously working features as well as indirectly affected modules. व्यापक – आधी चालणारी फीचर्स व अप्रत्यक्षरीत्या प्रभावित झालेले मॉड्यूल्स तपासले जातात.	Narrow – limited to the failed test cases or defect areas that were fixed. मर्यादित – फक्त फेल झालेल्या टेस्ट केसेस किंवा बग एरियावर लक्ष केंद्रित.
Performed On (कोणावर केली जाते)	Previously passed test cases (to confirm they still work after updates).आधी पास झालेल्या टेस्ट केसेसवर.	Previously failed test cases (to confirm defect resolution). आधी फेल झालेल्या टेस्ट केसेसवर.
Automation Suitability (ऑटोमेशन योग्यतेबाबत)	Highly suitable – regression suites are automated in Agile/DevOps environments. अत्यंत योग्य – ॲजाईल / डेव्हऑप्स मध्ये रिग्रेशन सूट ऑटोमेट केली जाते.	Less suitable – usually executed manually for specific fixes. कमी योग्य बहुतेक वेळा मॅन्युअली केली जाते.
Time & Effort (वेळ व प्रयत्न)	More time-consuming due to wider coverage. जास्त वेळखाऊ – कारण टेस्टिंगचा आवाका मोठा असतो.	Less time-consuming since it focuses on specific defects. कमी वेळखाऊ – कारण फक्त ठराविक बग तपासले जातात.
Dependency (अवलंबित्व)	Not dependent only on defect fixes; also done for new features or system changes फक्त बग फिक्सवर अवलंबून नसते; नवीन फीचर्स किंवा बदलांसाठीही केली जाते.	Always dependent on previously reported defect fixes. नेहमी आधी रिपोर्ट केलेल्या बग फिक्सवर अवलंबून असते.
Example (उदाहरण)	After updating the login module, ensure dashboard, profile, and logout still work correctly. लॉगिन मॉड्यूल अपडेट केल्यानंतर डॅशबोर्ड, प्रोफाइल, लॉगआउट तपासणे.	After fixing invalid-login bug, verify the bug is resolved. अवैध लॉगिन बग दुरुस्त केल्यानंतर तो खरंच सुटला आहे का ते तपासणे.

2.5.3 सिक्युरिटी टेस्टिंग (Security Testing)

सिक्युरिटी टेस्टिंग हा सॉफ्टवेअर टेस्टिंगचा एक प्रकार आहे, ज्यामध्ये ॲप्लिकेशन किंवा सिस्टीममध्ये कोणत्याही प्रकारच्या सुरक्षा त्रुटी (Vulnerabilities) नाहीत ना, डेटा सुरक्षित आहे ना, आणि दुष्ट हल्ल्यांच्या (Malicious Attacks) परिस्थितीतही सिस्टीम योग्यरित्या कार्य करते का, हे तपासले जाते. याचा मुख्य उद्देश म्हणजे अनधिकृत प्रवेश

(Unauthorized Access), डेटा ब्रिचेस (Data Breaches) आणि सिस्टीमच्या गैरवापराला प्रतिबंध करणे. सिक्युरिटी टेस्टिंग ही एक अत्यंत महत्त्वाची नॉन-फंक्शनल टेस्टिंग पद्धत आहे, जी सिस्टीम विविध प्रकारच्या धोक्यांपासून किती सुरक्षित आहे, हे तपासते. यामध्ये कॉन्फिडेन्शिअलिटी (Confidentiality), इंटेग्रिटी (Integrity), ऑथेंटिकेशन (Authentication), ऑथरायझेशन (Authorization), अव्हेलेबिलिटी (Availability) आणि नॉन-रिप्युडिएशन (Non-repudiation) यांची पडताळणी केली जाते. विविध प्रकारचे हल्ले सिम्युलेट करून आणि सिस्टीममधील एंट्री पॉइंट्स तपासून एसक्यूएल इंजेक्शन, क्रॉस-साइट स्क्रिप्टिंग (एक्सएसएस)(SQL Injection, Cross-Site Scripting (XSS)), कमकुवत ऑथेंटिकेशन, असुरक्षित APIs, सेशन मॅनेजमेंटमधील त्रुटी अशा अनेक कमतरता शोधल्या जातात. सिक्युरिटी टेस्टिंगमुळे संवेदनशील माहितीचे संरक्षण होते, युजरचा विश्वास टिकून राहतो आणि आयएसओ, ओडब्ल्यूएएसपी, जीडीपीआर (ISO, OWASP, GDPR) यांसारख्या सुरक्षा मानकांचे पालन सुनिश्चित होते. सिक्युरिटी टेस्टिंगमध्ये पेनिट्रेशन टेस्टिंग, व्हल्नरेबिलिटी स्कॅनिंग, रिस्क असेसमेंट, सिक्युरिटी ऑडिट्स आणि एथिकल हॅकिंग यांसारख्या तंत्रांचा समावेश असतो. अंतिम उद्दिष्ट म्हणजे सिस्टीमला अंतर्गत आणि बाह्य दोन्ही प्रकारच्या धोक्यांपासून सुरक्षित, स्थिर आणि विश्वासार्ह बनवणे.

सिक्युरिटी टेस्टिंगची वैशिष्ट्ये (Characteristics of Security Testing)

1. **कॉन्फिडेन्शिअलिटी (Confidentiality):** लॉगिन डिटेल्स, आर्थिक माहिती, वैयक्तिक डेटा यांसारखी संवेदनशील माहिती अनधिकृत प्रवेशापासून सुरक्षित ठेवते.
   उदाहरण: युजर पासवर्ड एन्क्रिप्ट करणे आणि डेटा लीक होण्यापासून संरक्षण करणे.
2. **इंटेग्रिटी (Integrity):** सिस्टीममधील माहितीमध्ये अनधिकृत बदल होणार नाही, याची खात्री करते.
   उदाहरण: बँकिंग सिस्टीममधील ट्रान्झॅक्शन रेकॉर्ड्समध्ये छेडछाड होऊ नये.
3. **ऑथेंटिकेशन (Authentication):** सिस्टीममध्ये प्रवेश देण्यापूर्वी युजरची ओळख तपासते.
   उदाहरण: युजरनेम–पासवर्ड, ओटीपी, बायोमेट्रिक्स (फिंगरप्रिंट / फेस आयडी), टू-फॅक्टर ऑथेंटिकेशन (2एफए).
4. **ऑथरायझेशन (Authorization):** युजरला फक्त त्याला परवानगी असलेल्या संसाधनांनाच प्रवेश मिळतो, याची खात्री करते.
   उदाहरण: विद्यार्थी स्वतःचा निकाल पाहू शकतो, पण सर्व विद्यार्थ्यांचा डेटाबेस पाहू शकत नाही.
5. **अव्हेलेबिलिटी (Availability):** हल्ले किंवा अनपेक्षित परिस्थितीतही सिस्टीम आणि सेवा उपलब्ध राहतात का, हे तपासते.
   उदाहरण: डिस्ट्रिब्युटेड डिनायल ऑफ सर्व्हिस (Distributed Denial of Service (DDoS)) हल्ल्यांपासून संरक्षण.
6. **नॉन-रिप्युडिएशन (Non-Repudiation):** केलेली कृती युजर किंवा सिस्टीम नाकारू शकत नाही, याची खात्री करते.
   उदाहरण: ऑनलाइन बँकिंगमध्ये ट्रान्झॅक्शन झाल्यानंतर ग्राहक किंवा बँक ती नाकारू शकत नाही.
7. **हल्ल्यांविरुद्ध प्रतिकारशक्ती (Resilience Against Attacks):** एसक्यूएल इंजेक्शन, एक्सएसएस, ब्रूट फोर्स अटॅक्स, मालवेअर यांसारख्या हल्ल्यांना सिस्टीम तोंड देऊ शकते का, हे तपासते.
8. **कॉम्प्लायन्स-आधारित (Compliance-Oriented):** सिक्युरिटी टेस्टिंगमुळे आंतरराष्ट्रीय मानकांचे पालन होते:
   - आयएसओ – इंटरनॅशनल ऑर्गनायझेशन फॉर स्टँडर्डायझेशन (उदा. ISO/IEC 27001)
   - ओडब्ल्यूएएसपी – ओपन वेब ॲप्लिकेशन सिक्युरिटी प्रोजेक्ट (OWASP टॉप 10 व्हल्नरेबिलिटीज)
   - जीडीपीआर – जनरल डेटा प्रोटेक्शन रेग्युलेशन (GDPR – General Data Protection Regulation) डेटा संरक्षणासाठी युरोपियन कायदा.
9. **प्रोॲक्टिव्ह व्हल्नरेबिलिटी डिटेक्शन (Proactive Vulnerability Detection):** हॅकर्स exploit करण्याआधीच सिस्टीममधील कमतरता शोधते. पेनिट्रेशन टेस्टिंग आणि व्हल्नरेबिलिटी स्कॅनिंग(Penetration Testing and Vulnerability Scanning) यांसारखी साधने वापरली जातात.
10. **रिस्क रिडक्शन आणि ट्रस्ट ॲश्युरन्स (Risk Reduction & Trust Assurance):** डेटा ब्रिचचा धोका कमी करते, सिस्टीमची विश्वासार्हता वाढवते आणि युजर्सचा विश्वास दृढ करते.

उदाहरण: बँकिंग वेब ॲप्लिकेशन (Banking Web Application)

परिस्थिती (Scenario): लॉगिन, व्यवहार (Transactions) आणि डेटा सिक्युरिटी यासाठी बँकिंग वेब ॲप्लिकेशनची सिक्युरिटी टेस्टिंग केली जाते.

प्रक्रिया (Process)

1. लॉगिन टेस्ट – वैध (Valid) आणि अवैध (Invalid) क्रेडेन्शियल्स वापरून तपासणी.
2. हल्ल्यांचे सिम्युलेशन (Attack Simulation) –
   - SQL इंजेक्शन (SQL Injection)
   - ब्रूट फोर्स अटॅक (Brute Force Attack)
   - सेशन हायजॅकिंग (Session Hijacking)
3. ट्रान्झॅक्शन टेस्ट – एन्क्रिप्शन वापरून सुरक्षित फंड ट्रान्सफरची तपासणी.
4. कॉम्प्लायन्स तपासणी (Compliance Check) –
   - आयएसओ/आयईसी 27001(ISO/IEC 27001)
   - ओडब्ल्यूएएसपी टॉप 10(OWASP Top 10)
   - जीडीपीआर (GDPR)

निरीक्षण (Observation)

1. योग्य क्रेडेन्शियल्सने सुरक्षित लॉगिन होते.
2. अनेक वेळा चुकीचा पासवर्ड टाकल्यास अकाउंट लॉक होते.
3. एसक्यूएल इंजेक्शन (SQL Injection) हल्ले ब्लॉक केले जातात.
4. अनधिकृत अकाउंट ॲक्सेस नाकारला जातो.
5. सर्व ट्रान्झॅक्शन्स एन्क्रिप्टेड आणि लॉग केलेले असतात.
6. जर कोणतीही सुरक्षा त्रुटी (Flaws) आढळली → दुरुस्ती (Fix) आवश्यक.

Test Cases:

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
ST-01	Verify login with valid credentials वैध लॉगिन क्रेडेन्शियल्स तपासणे	Enter correct username & password योग्य युजरनेम व पासवर्ड टाकणे	User logged in successfully युजर यशस्वीपणे लॉगिन होतो	Login success लॉगिन यशस्वी	Pass (यशस्वी)
ST-02	Verify login with invalid credentials अवैध क्रेडेन्शियल्ससह लॉगिन तपासणे	Enter wrong password 3 times चुकीचा पासवर्ड 3 वेळा टाकणे	Account locked after 3 failed attempts 3 अयशस्वी प्रयत्नांनंतर खाते लॉक होणे	Account locked खाते लॉक झाले	Pass (यशस्वी)
ST-03	SQL Injection protection	Enter ' OR '1'='1 in login field लॉगिन फील्डमध्ये ' OR '1'='1 टाकणे	Login attempt blocked लॉगिन प्रयत्न ब्लॉक होणे	Injection blocked Injection ब्लॉक झाले	Pass (यशस्वी)

	SQL Injection पासून संरक्षण तपासणे				
ST-04	Session hijacking prevention सेशन हायजॅकिंग प्रतिबंध तपासणे	Try accessing account without session token सेशन टोकनशिवाय खाते उघडण्याचा प्रयत्न	Access denied, redirected to login ॲक्सेस नाकारला जावा व लॉगिनकडे रिडायरेक्ट	Unauthorized access denied अनधिकृत प्रवेश नाकारला	Pass (यशस्वी)
ST-05	Secure transaction सुरक्षित ट्रान्झॅक्शन तपासणे	Perform fund transfer between accounts खात्यांदरम्यान निधी हस्तांतरण करणे	Transaction encrypted & logged ट्रान्झॅक्शन एन्क्रिप्ट व लॉग झालेले	Successful secure transfer सुरक्षित ट्रान्सफर यशस्वी	Pass (यशस्वी)
ST-06	Data privacy compliance डेटा गोपनीयता नियमांचे पालन तपासणे	Check user data storage and logs युजर डेटा स्टोरेज व लॉग्स तपासणे	Data encrypted, GDPR compliant डेटा एन्क्रिप्टेड व GDPR अनुरूप	Secure data handling सुरक्षित डेटा हाताळणी	Pass (यशस्वी)

2.5.4 क्लायंट–सर्व्हर टेस्टिंग (Client-Server Testing)

क्लायंट–सर्व्हर टेस्टिंग ही सॉफ्टवेअर टेस्टिंगची एक पद्धत आहे, जी क्लायंट–सर्व्हर आर्किटेक्चरवर आधारित ॲप्लिकेशन्ससाठी वापरली जाते. या आर्किटेक्चरमध्ये क्लायंट (Front-end) सर्व्हरकडे विनंत्या (Requests) पाठवतो आणि सर्व्हर (Back-end) त्या विनंत्यांवर प्रक्रिया करून प्रतिसाद (Responses) परत पाठवतो. या टेस्टिंगचा मुख्य उद्देश क्लायंट आणि सर्व्हर यांच्यातील योग्य कम्युनिकेशन, फंक्शनॅलिटी, परफॉर्मन्स आणि सिक्युरिटी यांची पडताळणी करणे हा आहे. ही टेस्टिंग पद्धत सुनिश्चित करते की अनेक क्लायंट्सकडून येणाऱ्या विनंत्या सर्व्हर योग्य प्रकारे हाताळतो, कोणतीही चूक, विलंब किंवा डेटा लॉस होत नाही. तसेच, वेगवेगळ्या परिस्थितींमध्ये (सामान्य लोड, जड लोड, अपयशाची स्थिती) सर्व्हर योग्य प्रतिसाद देतो का हे तपासले जाते. यामध्ये क्लायंट साइड (UI, इनपुट व्हॅलिडेशन, वापर सुलभता) आणि सर्व्हर साइड (डेटा हँडलिंग, सिक्युरिटी, डेटाबेस ऑपरेशन्स, कॉन्करन्सी) दोन्हींची चाचणी केली जाते.

क्लायंट–सर्व्हर टेस्टिंग मध्ये समाविष्ट बाबी:

- फंक्शनल टेस्टिंग → क्लायंट–सर्व्हर Request/Response फ्लोची तपासणी
- परफॉर्मन्स टेस्टिंग → वेग, स्केलेबिलिटी आणि लोडखालील स्थिरता मोजणे
- सिक्युरिटी टेस्टिंग → डेटा गोपनीयता, ऑथेंटिकेशन आणि ॲक्सेस कंट्रोलची खात्री
- कम्पॅटिबिलिटी टेस्टिंग → वेगवेगळ्या डिव्हाइसेस आणि ब्राउझर्सवरील सपोर्ट तपासणे

ही टेस्टिंग पद्धत बँकिंग ॲप्स, ई-कॉमर्स वेबसाइट्स, सोशल मीडिया प्लॅटफॉर्म्स आणि एंटरप्राइझ ॲप्लिकेशन्समध्ये मोठ्या प्रमाणावर वापरली जाते, जिथे अनेक क्लायंट्स एकाच वेळी सर्व्हरशी संवाद साधतात.

क्लायंट–सर्व्हर टेस्टिंग ची वैशिष्ट्ये (Characteristics)

1. **टू-टियर / मल्टी-टियर फोकस (Two-Tier / Multi-Tier)** – क्लायंट साइड (UI/Requests) आणि सर्व्हर साइड (प्रोसेसिंग, डेटाबेस) दोन्हीची चाचणी
2. **रिक्वेस्ट–रिस्पॉन्स पडताळणी (Request–Response)** – क्लायंट आणि सर्व्हरमधील योग्य डेटा एक्सचेंजची खात्री
3. **कन्करन्सी हँडलिंग (Concurrency Handling)** – अनेक क्लायंट्स असताना सर्व्हरची कार्यक्षमता तपासणे

4. **परफॉर्मन्स आणि लोड टेस्टिंग (Performance & Load Testing)** – प्रतिसाद वेळ (Response Time) आणि सर्व्हर स्थिरता मोजणे
5. **डेटा इंटेग्रिटी (Data Integrity)** – डेटा साठवण आणि पुनर्प्राप्ती अचूक व सुसंगत आहे का ते तपासणे
6. **सिक्युरिटी (Security)** – ऑथेंटिकेशन, ऑथरायझेशन आणि एन्क्रिप्शनची पडताळणी
7. **स्केलेबिलिटी (Scalability)** – वापरकर्त्यांच्या वाढत्या विनंत्या हाताळण्याची सर्व्हरची क्षमता
8. **एरर हँडलिंग (Error Handling)** – चुकीच्या इनपुट्स किंवा अपयशांवर योग्य प्रतिसाद मिळतो का ते तपासणे

उदाहरण: ऑनलाइन बँकिंग ॲप्लिकेशन

परिस्थिती (Scenario): वापरकर्ता ऑनलाइन बँकिंग सिस्टिममध्ये लॉगिन करतो, बॅलन्स तपासतो आणि फंड ट्रान्सफर करतो.

प्रक्रिया (Process)

1. क्लायंट (वेब/मोबाईल ॲप) लॉगिन क्रेडेन्शियल्स सर्व्हरकडे पाठवतो.
2. सर्व्हर क्रेडेन्शियल्स पडताळतो आणि डेटाबेसमधून माहिती घेतो.
3. क्लायंट फंड ट्रान्सफरची विनंती करतो → सर्व्हर प्रक्रिया करतो → डेटाबेसमध्ये व्यवहार अपडेट होतो.
4. सर्व्हर क्लायंटकडे कन्फर्मेशन पाठवतो.

निरीक्षण (Observation)

- योग्य क्रेडेन्शियल्सने यशस्वी लॉगिन होते.
- अनेक क्लायंट्स एकाच वेळी अकाउंट ॲक्सेस करू शकतात.
- व्यवहार डेटाबेसमध्ये अचूकरीत्या नोंदवले जातात.
- सर्व्हरवर जास्त लोड असल्यास प्रतिसादात विलंब दिसून येतो.

Test Cases:

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
CST-01	Verify login लॉगिन तपासणे	Enter valid credentials from client क्लायंटकडून वैध क्रेडेन्शियल्स टाकणे	Server validates & logs in सर्व्हर पडताळणी करून लॉगिन करतो	Login success लॉगिन यशस्वी	Pass (यशस्वी)
CST-02	Invalid login handling अवैध लॉगिन हँडलिंग	Enter wrong password चुकीचा पासवर्ड टाकणे	Server denies access सर्व्हर प्रवेश नाकारतो	Access denied प्रवेश नाकारला	Pass (यशस्वी)
CST-03	Multiple client access अनेक क्लायंट प्रवेश तपासणे	100 clients log in simultaneously 100 क्लायंट्स एकाच वेळी लॉगिन करणे	Server handles without crash सर्व्हर क्रॅश न होता हाताळतो	Concurrent access supported एकत्रित प्रवेश समर्थित	Pass (यशस्वी)
CST-04	Transaction update ट्रान्झॅक्शन अपडेट तपासणे	Transfer ₹1000 from A to B A कडून B कडे ₹1000 ट्रान्सफर	Server updates DB, confirms transfer	Transaction successful ट्रान्झॅक्शन यशस्वी	Pass (यशस्वी)

			सर्व्हर DB अपडेट करून ट्रान्सफर कन्फर्म करतो		
CST-05	Data integrity डेटा अखंडता तपासणे	Refresh account after transaction ट्रान्सफरनंतर खाते रिफ्रेश करणे	Updated balance shown correctly अपडेटेड बॅलन्स योग्य दाखवला जातो	Data consistent डेटा सुसंगत	Pass (यशस्वी)
CST-06	Load handling लोड हँडलिंग तपासणे	Simulate 5000 users at once 5000 युजर्स एकाच वेळी सिम्युलेट करणे	Server maintains stable response time सर्व्हर स्थिर प्रतिसाद वेळ राखतो	Minor delays not observed थोडा विलंब दिसून आला	Fail (अयशस्वी)
CST-07	Security check सिक्युरिटी तपासणी	Try SQL injection in login field लॉगिन फील्डमध्ये SQL Injection प्रयत्न	Server rejects malicious input सर्व्हर घातक इनपुट नाकारतो	Injection blocked Injection ब्लॉक झाले	Pass (यशस्वी)

2.5.5 GUI टेस्टिंग (GUI Testing)

GUI (ग्राफिकल युजर इंटरफेस) टेस्टिंग ही सॉफ्टवेअर टेस्टिंगची एक पद्धत आहे, जी ॲप्लिकेशनच्या युजर इंटरफेसची कार्यक्षमता, वापर सुलभता (Usability) आणि दृश्य घटक (Visual Elements) यांची पडताळणी करण्यासाठी वापरली जाते. यामध्ये बटणं, मेन्यू, टेक्स्ट बॉक्स, प्रतिमा, लिंक्स, फॉर्म्स इत्यादी GUI घटक योग्य प्रकारे काम करतात का आणि वापरकर्त्याला सहज व सुरळीत अनुभव देतात का हे तपासले जाते. GUI हा वापरकर्त्याचा ॲप्लिकेशनशी पहिला संपर्कबिंदू असल्यामुळे, त्यामधील कोणतीही त्रुटी थेट वापर सुलभता आणि ग्राहक समाधानावर परिणाम करू शकते. GUI टेस्टिंगद्वारे ॲप्लिकेशनचे फंक्शनल तसेच सौंदर्यात्मक (Aesthetic) पैलू आवश्यकता पूर्ण करतात का याची खात्री केली जाते. GUI टेस्टिंग मॅन्युअली केली जाऊ शकते, परंतु वेळ वाचवण्यासाठी आणि अचूकता वाढवण्यासाठी Selenium, QTP, TestComplete, Ranorex यांसारखी ऑटोमेशन टूल्स मोठ्या प्रमाणावर वापरली जातात. GUI टेस्टिंग सामान्यतः युनिट टेस्टिंग आणि इंटिग्रेशन टेस्टिंगनंतर केली जाते. यामध्ये वेगवेगळ्या पर्यावरणांमध्ये (Environments), स्क्रीन रिझोल्यूशन्स आणि ब्राउझर्समध्ये फ्रंट-एंड वर्तनाची तपासणी केली जाते. याचे मुख्य उद्दिष्ट वापरण्यास सोपे इंटरफेस, सुसंगत डिझाइन, ॲक्सेसिबिलिटी आणि योग्य नेव्हिगेशन फ्लो सुनिश्चित करणे हे आहे.

GUI टेस्टिंगची वैशिष्ट्ये (Characteristics of GUI Testing)

1. **युजर अनुभवाची पडताळणी (Validates User Experience):** ॲप्लिकेशनचा इंटरफेस वापरकर्त्यासाठी सोपा, समजण्यास सुलभ आणि युजेबिलिटी स्टँडर्ड्सनुसार आहे का हे तपासते.
2. **डिझाइन सुसंगततेची तपासणी (Checks Consistency of Design):** वेगवेगळ्या स्क्रीन आणि डिव्हाइसेसवर फॉन्ट्स, रंग, अलाईनमेंट आणि लेआउट्स एकसारखे आहेत का याची खात्री करते.
3. **कार्यात्मक अचूकता सुनिश्चित करते (Ensures Functional Correctness):** बटणं, इनपुट फील्ड्स, ड्रॉप-डाऊन मेन्यू यांसारखे इंटरॲक्टिव घटक अपेक्षेप्रमाणे काम करतात का ते तपासते.
4. **क्रॉस-प्लॅटफॉर्म सुसंगततेस समर्थन (Supports Cross-Platform Compatibility):** वेगवेगळ्या ऑपरेटिंग सिस्टिम्स, ब्राउझर्स आणि डिव्हाइसेसवर इंटरफेस योग्य प्रकारे वागतो का याची पडताळणी करते.

उदाहरण: ऑनलाइन शॉपिंग ॲप्लिकेशन

परिस्थिती (Scenario):डेव्हलपमेंट टीमने प्रॉडक्ट डिटेल पेजचा लेआउट पुन्हा डिझाइन केला आहे आणि नवीन "Wishlist" बटण जोडले आहे.

प्रक्रिया (Process):QA टीम GUI टेस्टिंग करते, ज्यामध्ये केवळ नवीन बटणच नाही तर खालील बाबींचीही पडताळणी केली जाते:

1. होमपेज, प्रॉडक्ट डिटेल पेज आणि चेकआउट पेजवरील लेआउटची सुसंगतता.
2. "Wishlist" बटणची कार्यक्षमता (योग्य प्रकारे काम करते का).
3. टेक्स्ट, प्रतिमा आणि किंमतींच्या तपशीलांची योग्य अलाईनमेंट.
4. वेगवेगळ्या ब्राउझर्ससोबत सुसंगतता (Chrome, Firefox, Edge).

निरीक्षण (Observation):पुन्हा डिझाइन केलेले पेज सुसंगत दिसते, "Wishlist" बटण योग्य प्रकारे कार्य करते, परंतु Firefox ब्राउझरमध्ये चेकआउट पेजवरील टेक्स्ट अलाईनमेंट बिघडलेली आढळते.

टेस्ट केसेस (Test Cases):

Test Case ID (टेस्ट केस आयडी)	**Objective (उद्दिष्ट)**	**Steps / Input (स्टेप्स / इनपुट)**	**Expected Output (अपेक्षित आउटपुट)**	**Actual Output (प्रत्यक्ष आउटपुट)**	**Status (स्थिती)**
GUI-01	Verify layout consistency लेआउट सुसंगतता तपासणे	Open homepage, product page, checkout page होमपेज, प्रॉडक्ट पेज व चेकआउट पेज उघडणे	Layout aligned properly on all pages सर्व पेजेसवर लेआउट योग्यरीत्या अलाइन्ड असावे	Layout aligned correctly लेआउट योग्य आहे	Pass (यशस्वी)
GUI-02	Verify Wishlist button विशलिस्ट बटण तपासणे	Click on "Add to Wishlist" "Add to Wishlist" वर क्लिक करणे	Product added to wishlist with confirmation प्रॉडक्ट विशलिस्टमध्ये जोडले जावे व कन्फर्मेशन मिळावे	Product added successfully प्रॉडक्ट यशस्वीपणे जोडले	Pass (यशस्वी)
GUI-03	Verify text alignment on checkout page चेकआउट पेजवरील टेक्स्ट अलाइन्मेंट तपासणे	Open checkout page in Firefox Firefox मध्ये चेकआउट पेज उघडणे	Text and fields aligned properly टेक्स्ट व फील्ड्स योग्यरीत्या अलाइन्ड असावेत	Text misaligned on Firefox Firefox मध्ये टेक्स्ट विस्कळीत	Fail (अयशस्वी)
GUI-04	Verify image rendering इमेज रेंडरिंग तपासणे	Load product images on detail page डिटेल पेजवरील प्रॉडक्ट इमेजेस लोड करणे	All images load without distortion सर्व इमेजेस विकृतीशिवाय लोड व्हाव्यात	Images loaded correctly इमेजेस योग्यरीत्या लोड झाल्या	Pass (यशस्वी)
GUI-05	Verify cross-browser compatibility	Test in Chrome, Firefox, Edge	Consistent appearance and functionality	Minor misalignment in Firefox	Fail (अयशस्वी)

	क्रॉस-ब्राउझर सुसंगतता तपासणे	Chrome, Firefox, Edge मध्ये टेस्ट करणे	सर्व ब्राउझरमध्ये समान दिसणे व कार्यक्षमता	Firefox मध्ये थोडी विस्कळीतता	

2.5.6 डेटाबेस टेस्टिंग (Database Testing):

डेटाबेस टेस्टिंग ही एक सॉफ्टवेअर टेस्टिंग तंत्र आहे जी डेटाबेस सिस्टिममधील स्कीमा (Schema), टेबल्स, ट्रिगर्स, स्टोर्ड प्रोसीजर्स, डेटा इंटेग्रिटी आणि ट्रान्झॅक्शन्स यांची पडताळणी करते. याचा मुख्य उद्देश ॲप्लिकेशनचा बॅक-एंड योग्य प्रकारे कार्य करतो का आणि डेटा योग्यरीत्या साठवला, मिळवला, अपडेट व डिलीट केला जातो का हे तपासणे हा आहे. GUI टेस्टिंग फ्रंट-एंड वर्तन तपासते, तर डेटाबेस टेस्टिंग बॅक-एंडमधील डेटाची विश्वासार्हता, सुसंगतता (Consistency) आणि सुरक्षा सुनिश्चित करते. यामध्ये बिझनेस नियम डेटाबेस स्तरावर योग्यरीत्या अंमलात आणले आहेत का, तसेच जड लोडमध्येही परफॉर्मन्स योग्य राहतो का हे तपासले जाते. डेटाबेस टेस्टिंग SQL क्वेरीज वापरून मॅन्युअली किंवा Selenium (JDBC/ODBC), QTP, DbUnit सारख्या टूल्सद्वारे ऑटोमेटेड पद्धतीने केली जाऊ शकते. बँकिंग, हेल्थकेअर आणि ई-कॉमर्स सारख्या ॲप्लिकेशन्समध्ये, जिथे डेटा अचूकता आणि सुरक्षा अत्यंत महत्त्वाची असते, तिथे डेटाबेस टेस्टिंग फारच आवश्यक असते. ही टेस्टिंग डेटा सुसंगतता, रेफरेंशियल इंटेग्रिटी आणि बिझनेस गरजांचे पालन सुनिश्चित करते.

डेटाबेस टेस्टिंगची वैशिष्ट्ये (Characteristics of Database Testing):

1. **डेटा इंटेग्रिटीची पडताळणी**: डेटा कोणत्याही प्रकारच्या करप्शन किंवा नुकसानाशिवाय योग्यरीत्या साठवला व मिळवला जातो याची खात्री करते.
2. **डेटाबेस स्तरावरील बिझनेस नियम तपासते:** ट्रिगर्स, स्टोर्ड प्रोसीजर्स आणि कन्स्ट्रेंट्स योग्य प्रकारे काम करतात का हे तपासते.
3. **ट्रान्झॅक्शनची विश्वासार्हता सुनिश्चित करते:** ट्रान्झॅक्शन्स ACID (Atomicity, Consistency, Isolation, Durability) गुणधर्मांचे पालन करतात का हे तपासते.
4. **परफॉर्मन्स आणि सुरक्षा समर्थन:** क्वेरी ऑप्टिमायझेशन, प्रतिसाद वेळ (Response Time) आणि अनधिकृत प्रवेश रोखला जातो का याचे मूल्यमापन करते.

उदाहरण: ऑनलाइन शॉपिंग ॲप्लिकेशन

परिस्थिती (Scenario): वापरकर्ता खरेदी पूर्ण केल्यानंतर ऑर्डर ट्रान्झॅक्शन्स डेटाबेसमध्ये योग्यरीत्या साठवल्या जात आहेत का हे QA टीमला तपासायचे आहे.

प्रक्रिया (Process): डेटाबेस टेस्टिंग करून खालील बाबींची पडताळणी केली जाते:

1. सर्व ग्राहक आणि ऑर्डर तपशील योग्यरीत्या इन्सर्ट झाले आहेत का.
2. फॉरेन की रिलेशनशिप (Customer ID – Order ID) योग्य प्रकारे जपली जाते का.
3. ऑर्डरमध्ये बदल केल्यानंतर डेटा योग्यरीत्या अपडेट होतो का.
4. पेमेंट फेल झाल्यास ट्रान्झॅक्शन योग्यरीत्या रोलबॅक होते का.

निरीक्षण (Observation):पूर्ण झालेल्या ऑर्डर्ससाठी डेटा योग्यरीत्या इन्सर्ट होतो, परंतु पेमेंट गेटवे एरर आल्यास रोलबॅक प्रक्रिया अयशस्वी ठरते.

टेस्ट केसेस (Test Cases):

Test Case ID (टेस्ट केस आयडी)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
DB-01	Verify order insertion ऑर्डर इन्सर्शन तपासणे	Place order with valid customer details वैध ग्राहक माहिती वापरून ऑर्डर देणे	Order details inserted into database ऑर्डरची माहिती डेटाबेसमध्ये इन्सर्ट झाली पाहिजे	Order inserted correctly ऑर्डर योग्यरीत्या इन्सर्ट झाली	Pass (यशस्वी)
DB-02	Verify foreign key constraint फॉरेन की कन्स्ट्रेंट तपासणे	Insert order without valid Customer ID अवैध Customer ID सह ऑर्डर इन्सर्ट करणे	Database rejects insertion डेटाबेसने इन्सर्शन नाकारले पाहिजे	Constraint violation error displayed कन्स्ट्रेंट व्हायोलेशन एरर दिसली	Pass (यशस्वी)
DB-03	Verify update operation अपडेट ऑपरेशन तपासणे	Update order status to "Shipped" ऑर्डर स्टेटस "Shipped" करणे	Status updated in Orders tableOrders टेबलमध्ये स्टेटस अपडेट झाले पाहिजे	Status updated correctly स्टेटस योग्यरीत्या अपडेट झाले	Pass (यशस्वी)
DB-04	Verify rollback on payment failure पेमेंट फेल झाल्यावर रोलबॅक तपासणे	Simulate payment failure पेमेंट फेल सिम्युलेट करणे	Transaction rolled back, no order entry ट्रान्झॅक्शन रोलबॅक होऊन कोणतीही ऑर्डर एन्ट्री नसावी	Rollback failed, partial data saved रोलबॅक फेल झाला, अर्धवट डेटा सेव्ह झाला	Fail (अयशस्वी)
DB-05	Verify stored procedure execution स्टोअर्ड प्रोसीजर एक्झिक्युशन तपासणे	Execute "GetOrderSummary" procedure "GetOrderSummary" प्रोसीजर रन करणे	Correct order details returned योग्य ऑर्डर तपशील मिळाला पाहिजे	Order details fetched correctly ऑर्डर तपशील योग्यरीत्या मिळाला	Pass (यशस्वी)

2.5.7 सॅनिटी टेस्टिंग (Sanity Testing):

सॅनिटी टेस्टिंग ही सॉफ्टवेअर टेस्टिंगची एक पद्धत आहे जी नवीन सॉफ्टवेअर बिल्ड मिळाल्यानंतर केली जाते, ज्यामध्ये एखादा विशिष्ट बग फिक्स किंवा नवीन फंक्शनॅलिटी योग्यरित्या काम करत आहे का हे तपासले जाते आणि त्या बदलांमुळे

संबंधित मॉड्यूल्समध्ये काही बिघाड झाला नाही याची खात्री केली जाते. ही नॅरो & डीप (Narrow & Deep) पद्धत आहे, म्हणजे संपूर्ण सिस्टीम तपासण्याऐवजी फक्त बदललेल्या किंवा प्रभावित भागांवर सखोल लक्ष दिले जाते. सॅनिटी टेस्टिंग बहुतेक वेळा अनस्क्रिप्टेड (Unscripted) आणि मॅन्युअली केली जाते, परंतु जर रिग्रेशन टेस्ट सूट उपलब्ध असेल तर ऑटोमेशनचा वापरही करता येतो. बग फिक्सनंतर लगेच सिस्टीम योग्य आहे का हे तपासण्यासाठी ही एक जलद चाचणी असते आणि त्यानंतरच पूर्ण रिग्रेशन किंवा सिस्टीम टेस्टिंग सुरू केली जाते.

सॅनिटी टेस्टिंगची वैशिष्ट्ये (Characteristics of Sanity Testing):

1. **फोकस्ड टेस्टिंग (Focused Testing):** बग फिक्स किंवा नवीन बदलांमुळे प्रभावित झालेल्या विशिष्ट भागांवर लक्ष केंद्रित करते.
2. **जलद आणि हलकी टेस्टिंग (Quick & Lightweight):** पूर्ण रिग्रेशन टेस्टिंगपेक्षा खूप जलद असते.
3. **अनस्क्रिप्टेड स्वरूप (Unscripted Nature):** बहुतेक वेळा अनौपचारिक पद्धतीने टेस्टर्सद्वारे केली जाते.
4. **गेटकीपरची भूमिका (Acts as a Gatekeeper):** बग फिक्स योग्य आहे का हे निश्चित करून पुढील टेस्टिंगसाठी परवानगी देते.

उदाहरण: ऑनलाइन शॉपिंग ॲप्लिकेशन

परिस्थिती (Scenario): "DISCOUNT10" हा कूपन कोड योग्यरित्या लागू होत नव्हता, हा बग डेव्हलपर्सनी दुरुस्त केला आहे.

प्रक्रिया (Process): QA टीम खालील गोष्टी तपासते:

1. बग फिक्सनंतर कूपन योग्यरीत्या लागू होत आहे का.
2. डिस्काउंटनंतर एकूण किंमत (Total Price) योग्यरीत्या कॅल्क्युलेट होते का.
3. संबंधित फंक्शनॅलिटीज जसे की चेकआउट आणि पेमेंट प्रक्रिया नीट चालते का.

निरीक्षण (Observation): कूपन योग्यरित्या लागू होते आणि डिस्काउंट नीट मिळतो, त्यामुळे बग फिक्स यशस्वी असल्याचे निश्चित होते.

टेस्ट केसेस (Test Cases):

Test Case ID (टेस्ट केस आयडी)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
SA-01	Verify coupon fix कूपन फिक्स तपासणे	Apply "DISCOUNT10" "डिस्काउंट10" कूपन लागू करणे	10% discount applied 10% सवलत लागू झाली पाहिजे	Discount applied सवलत लागू झाली	Pass (यशस्वी)
SA-02	Verify total price calculation एकूण किंमत कॅल्क्युलेशन तपासणे	Apply coupon on cart total ₹1000 ₹1000 कार्ट टोटलवर कूपन लागू करणे	Total = ₹900 एकूण रक्कम ₹900 झाली पाहिजे	Total shown correctly एकूण रक्कम योग्यरीत्या दाखवली	Pass (यशस्वी)
SA-03	Verify checkout flow after coupon कूपननंतर चेकआउट फ्लो तपासणे	Proceed to checkout चेकआउटकडे पुढे जाणे	Checkout continues चेकआउट सुरळीत सुरू राहिले पाहिजे	Checkout successful चेकआउट यशस्वी झाले	Pass (यशस्वी)

SA-04	Verify payment with discount सवलतीसह पेमेंट तपासणे	Make payment पेमेंट करणे	Payment with updated total succeeds अपडेटेड रकमेवर पेमेंट यशस्वी झाले पाहिजे	Payment successful पेमेंट यशस्वी झाले	Pass (यशस्वी)
SA-05	Verify order summary ऑर्डर समरी तपासणे	View final summary अंतिम समरी पाहणे	Shows discounted amount सवलतीची रक्कम दिसली पाहिजे	Displayed correctly योग्यरीत्या दाखवली	Pass (यशस्वी)

2.5.8 स्मोक टेस्टिंग (Smoke Testing):

स्मोक टेस्टिंग ही सॉफ्टवेअर टेस्टिंगची एक पद्धत आहे जी एखाद्या नवीन बिल्डमधील सर्वात महत्त्वाच्या (क्रिटिकल) फंक्शनॅलिटीज योग्यरित्या कार्य करत आहेत का हे तपासण्यासाठी केली जाते. ही टेस्टिंग सखोल चाचणीपूर्वी केली जाणारी प्राथमिक चाचणी असून तिला "बिल्ड व्हेरिफिकेशन टेस्टिंग (Build Verification Testing)" असेही म्हटले जाते. स्मोक टेस्टिंगमुळे सॉफ्टवेअर बिल्ड पुढील तपशीलवार टेस्टिंगसाठी पुरेसा स्थिर (Stable) आहे का हे निश्चित होते. सामान्यतः डेव्हलपर्सकडून नवीन बिल्ड रिलीज झाल्यानंतर आणि QA टीमकडे देण्यापूर्वी स्मोक टेस्टिंग केली जाते. ही शॅलो & वाइड (Shallow & Wide) पद्धत आहे, म्हणजे सर्व प्रमुख मॉड्यूल्स तपासले जातात, परंतु सखोल फंक्शनल तपशीलात जात नाही. कन्टिन्युअस इंटिग्रेशन आणि कन्टिन्युअस डिप्लॉयमेंट (CI/CD (Continuous Integration and Continuous Deployment)) पाइपलाईनमध्ये Selenium, QTP, Jenkins सारखी ऑटोमेशन टूल्स वापरून स्मोक टेस्ट्स जलदगतीने चालवल्या जातात.

स्मोक टेस्टिंगची वैशिष्ट्ये (Characteristics of Smoke Testing):

1. **बिल्ड व्हेरिफिकेशन (Build Verification):** सॉफ्टवेअर बिल्ड टेस्टिंगसाठी स्थिर आणि वापरण्यायोग्य आहे का हे सुनिश्चित करते.
2. **त्रुटींची लवकर ओळख (Early Detection of Errors):** सुरुवातीलाच गंभीर (Showstopper) दोष ओळखते.
3. **शॅलो & वाइड (Shallow & Wide) टेस्टिंग:** मुख्य फंक्शनॅलिटीजवर लक्ष केंद्रित करते, सखोल तपशील तपासत नाही.
4. **वेळ वाचवणारी (Time-Saving):** अस्थिर बिल्डवर वेळ वाया जाण्यापासून वाचवते.

उदाहरण: ऑनलाइन शॉपिंग ॲप्लिकेशन

परिस्थिती (Scenario): नवीन बिल्ड डिप्लॉय केल्यानंतर QA टीम तपशीलवार फंक्शनल टेस्टिंग सुरू करण्यापूर्वी स्मोक टेस्टिंग करते.

प्रक्रिया (Process):QA टीम खालील गोष्टी तपासते:

1. ॲप्लिकेशन यशस्वीपणे सुरू होते का.
2. यूजर लॉगिन / लॉगआउट कार्यरत आहे का.
3. प्रॉडक्ट्स सर्च करता येतात का.
4. कार्टमध्ये आयटम ॲड करता येतात का.
5. चेकआउट पेज योग्यरीत्या लोड होते का.

निरीक्षण (Observation): लॉगिन फेल होते, त्यामुळे डेव्हलपर्सनी बिल्ड दुरुस्त करेपर्यंत पुढील तपशीलवार टेस्टिंग थांबवली जाते.

टेस्ट केसेस (Test Cases):

Test Case ID (टेस्ट केस आयडी)	Objective (उद्दिष्ट)	Steps / Input (स्टेप्स / इनपुट)	Expected Output (अपेक्षित आउटपुट)	Actual Output (प्रत्यक्ष आउटपुट)	Status (स्थिती)
SM-01	Verify application launch ॲप्लिकेशन सुरू होते का ते तपासणे	Open URL यूआरएल उघडणे	Home page loads successfully होम पेज यशस्वीपणे लोड झाले पाहिजे	Loaded successfully यशस्वीपणे लोड झाले	Pass (यशस्वी)
SM-02	Verify login functionality लॉगिन फंक्शनॅलिटी तपासणे	Enter valid credentials वैध लॉगिन माहिती टाकणे	Login successful, dashboard shown लॉगिन यशस्वी होऊन डॅशबोर्ड दिसला पाहिजे	Login failed लॉगिन अयशस्वी झाले	Fail (अयशस्वी)
SM-03	Verify product search प्रॉडक्ट सर्च तपासणे	Search "Mobile" "मोबाईल" शोधणे	Product list displayed प्रॉडक्ट यादी दिसली पाहिजे	Products shown correctly प्रॉडक्ट्स योग्यरीत्या दिसले	Pass (यशस्वी)
SM-04	Verify add to cart कार्टमध्ये प्रॉडक्ट जोडणे तपासणे	Add one item to cart एक आयटम कार्टमध्ये जोडणे	Cart updated कार्ट अपडेट झाली पाहिजे	Cart updated correctly कार्ट योग्यरीत्या अपडेट झाली	Pass (यशस्वी)
SM-05	Verify checkout page चेकआउट पेज तपासणे	Navigate to checkout चेकआउट पेजवर जाणे	Checkout page loads चेकआउट पेज लोड झाले पाहिजे	Page loaded correctly पेज योग्यरीत्या लोड झाले	Pass (यशस्वी)

Table 2.6: सॅनिटी टेस्टिंग वि. स्मोक टेस्टिंग (Comparison between Sanity and Smoke Testing)

निकष (Criteria)	स्मोक टेस्टिंग (Smoke Testing)	सॅनिटी टेस्टिंग (Sanity Testing)
Definition (व्याख्या)	A preliminary test conducted to verify whether the critical functionalities of a software build are working, ensuring the build is stable for further testing. नवीन सॉफ्टवेअर बिल्डमधील महत्त्वाची फंक्शनॅलिटी कार्यरत आहे का हे तपासण्यासाठी केली जाणारी प्राथमिक चाचणी.	A focused test conducted to verify that specific bug fixes or new functionalities are working as expected. विशिष्ट बग फिक्स किंवा नवीन बदल योग्य प्रकारे काम करत आहेत का हे तपासण्यासाठी केली जाणारी चाचणी.
Objective (उद्दिष्ट)	To check the stability of the entire build before detailed testing begins. संपूर्ण बिल्ड पुढील टेस्टिंगसाठी स्थिर आहे का हे तपासणे.	To validate correctness of recent changes and ensure no side effects are introduced.

		नवीन बदल योग्य आहेत का आणि इतर भागांवर परिणाम झाला नाही ना हे तपासणे.
Testing Depth (टेस्टिंगची खोली)	Shallow and broad – covers all major functionalities without detailed verification. वरवरची पण व्यापक – सर्व मुख्य फिचर्स तपासली जातात.	Narrow and deep – focuses on specific modules or areas affected by changes. मर्यादित पण सखोल – फक्त बदललेल्या भागांवर लक्ष.
Performed When (केव्हा केली जाते)	After receiving a new build, before proceeding to functional/regression testing. नवीन बिल्ड मिळाल्यानंतर लगेच.	After bug fixes or minor code changes. बग फिक्स किंवा लहान बदलांनंतर.
Nature (स्वरूप)	Usually scripted and often automated. बहुतेक वेळा स्क्रिप्टेड आणि ऑटोमेटेड.	Often unscripted, ad-hoc, and exploratory. बहुतेक वेळा अनस्क्रिप्टेड व मॅन्युअल.
Time Required (लागणारा वेळ)	Very quick – usually a few hours. खूप कमी वेळ लागतो.	Quick, but slightly longer than smoke testing. स्मोकपेक्षा थोडा जास्त वेळ.
Responsibility (जबाबदारी)	Performed by QA engineers after build release. QA इंजिनिअरद्वारे केली जाते.	Performed by QA testers after fix confirmation. QA टेस्टर्सद्वारे केली जाते.
Automation (ऑटोमेशन)	Frequently automated (CI/CD pipelines). बहुतेक वेळा ऑटोमेटेड.	Usually manual, rarely automated. बहुतेक वेळा मॅन्युअल.
Example (उदाहरण)	Verifying login, product search, add to cart, checkout page load. लॉगिन, प्रॉडक्ट सर्च, कार्ट, चेकआउट तपासणे.	Verifying fixed “Apply Coupon” bug.“Apply Coupon ” बग फिक्स तपासणे.
Outcome (निकाल)	Confirms whether the build is stable for further testing. बिल्ड पुढील टेस्टिंगसाठी तयार आहे का ते ठरते.	Confirms whether bug fixes/changes are valid and stable. बग फिक्स योग्य आहे का ते ठरते.

References

1. Desikan, S., & Ramesh, G. (2016). *Software Testing: Principles and Practices*. Pearson India. ISBN: 9788177581218.
2. Limaye, M. G. (2012). *Software Testing: Principles, Techniques and Tools*. Tata McGraw Hill Education. ISBN: 9780070139909.
3. Chauhan, N. (2016). *Software Testing: Principles and Practices*. Oxford University Press. ISBN: 9780198061847.
4. Singh, Y. (2012). *Software Testing*. Cambridge University Press. ISBN: 9781107652781.
5. Infosys Springboard – *Software Testing Fundamentals*. Available at: https://infyspringboard.onwingspan.com
6. NPTEL – *Software Testing Course*. Available at: https://nptel.ac.in/courses/106101163

7. GeeksforGeeks – *Sanity vs Smoke Testing*. Available at: https://www.geeksforgeeks.org/difference-between-sanity-testing-and-smoke-testing/
8. Guru99 – *Smoke Testing vs Sanity Testing*. Available at: https://www.guru99.com/smoke-testing.html
9. W3Schools – *Software Testing Tutorials*. Available at: https://www.w3schools.in/software-testing/tutorials/

युनिट-3
टेस्ट मॅनेजमेंट
(Test Management)

विषय निष्पत्ती (Course Outcome):

CO3: विविध सॉफ्टवेअर टेस्टिंग पद्धती स्पष्ट करा.

घटक निष्पत्ती (Theory Learning Outcome):

1. दिलेल्या प्रोग्राममधील त्रुटी आणि बग्स ओळखा.
2. दिलेल्या टेस्ट अनुप्रयोगासाठी प्रवेश व निर्गमन निकष स्पष्ट करा.
3. सॉफ्टवेअर टेस्टिंगच्या विविध पद्धती स्पष्ट करा.

3.1 टेस्ट लाईफ सायकल (Software Test Life Cycle – STLC)

सॉफ्टवेअर टेस्ट लाईफ सायकल (STLC) म्हणजे सॉफ्टवेअरच्या टेस्टिंग प्रक्रियेदरम्यान पार पाडल्या जाणाऱ्या पद्धतशीर आणि क्रमबद्ध क्रियांची मालिका होय, ज्यामुळे सॉफ्टवेअर उत्पादन आवश्यक गुणवत्तेच्या मानकांना (Quality Standards) पूर्तता करते याची खात्री केली जाते. STLC मधील प्रत्येक टप्प्याला ठराविक उद्दिष्टे, एन्ट्री क्रायटेरिया, एक्झिट क्रायटेरिया (Entry Criteria, Exit Criteria) कार्ये (Tasks) आणि डिलिव्हरेबल्स (Deliverables) असतात, त्यामुळे संपूर्ण टेस्टिंग प्रक्रिया संरचित (Structured) आणि मोजण्यायोग्य (Measurable) बनते. STLC सॉफ्टवेअर टेस्टिंगसाठी स्टेप-बाय-स्टेप पद्धत प्रदान करते. ही प्रक्रिया Requirement Analysis पासून सुरू होऊन टेस्ट क्लोजर (Test Closure) पर्यंत पूर्ण होते. या जीवनचक्रामध्ये नियोजन (Planning), तयारी (Preparation), अंमलबजावणी (Execution) आणि मूल्यमापन (Evaluation) यावर विशेष भर दिला जातो, ज्यामुळे सॉफ्टवेअर सिस्टीममध्ये गुणवत्ता हमी (Quality Assurance) प्राप्त होते.

थोडक्यात, STLC मुळे:

- सॉफ्टवेअरची गुणवत्ता सुधारते
- दोष लवकर ओळखता येतात
- टेस्टिंग प्रक्रिया प्रभावी व नियंत्रीत होते
- अंतिम उत्पादन अधिक विश्वासार्ह आणि स्थिर बनते

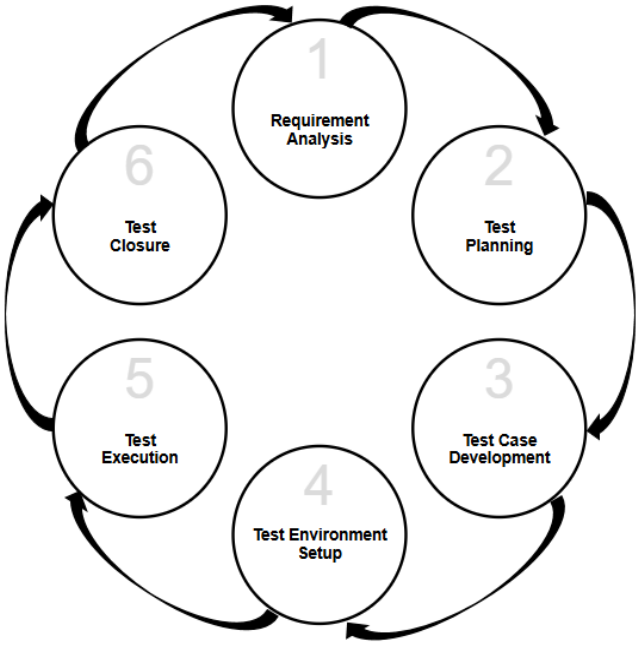

Fig 3.1: टेस्ट लाईफ सायकल (Test life cycle)

STLC चे टप्पे (Phases):

1. **आवश्यकता विश्लेषण (Requirement Analysis):** टेस्ट करता येणाऱ्या आवश्यकतांना समजून घेणे व त्यांचे विश्लेषण करणे.
2. **टेस्ट नियोजन (Test Planning):** टेस्टिंगसाठी धोरणे, व्याप्ती (Scope), संसाधने (Resources) आणि वेळापत्रक (Schedule) तयार करणे.
3. **टेस्ट केस विकास (Test Case Development):** टेस्ट केस डिझाइन करणे, टेस्ट स्क्रिप्ट्स आणि टेस्ट डेटा तयार करणे.
4. **टेस्ट वातावरण सेटअप (Environment Setup):** हार्डवेअर, सॉफ्टवेअर आणि टेस्टिंग टूल्स कॉन्फिगर करणे.
5. **टेस्ट अंमलबजावणी (Test Execution):** टेस्ट केस चालवणे आणि प्रत्यक्ष निकाल (Actual Result) अपेक्षित निकालांशी (Expected Result) तुलना करणे.
6. **टेस्ट केस क्लोजर (Test Case Closure):** टेस्ट सारांश अहवाल तयार करणे, शिकलेले धडे (Lessons Learned) नोंदवणे आणि टेस्टिंग प्रक्रिया औपचारिकरित्या बंद करणे.

उदाहरण (Example): परिस्थिती (Scenario): एक ऑनलाइन शॉपिंग ॲप्लिकेशन विकसित केले जात आहे. या सिस्टिममध्ये वापरकर्त्यांना लॉगिन करणे, प्रॉडक्ट शोधणे, कार्टमध्ये टाकणे आणि पेमेंट करणे शक्य असावे.

प्रक्रिया (Process):

1. रीक्वायरमेंट ॲनालिसिस (Requirement Analysis): लॉगिन, प्रॉडक्ट सर्च, कार्ट आणि पेमेंट ही मुख्य फंक्शन्स टेस्ट करायची आहेत हे ओळखले जाते.
2. टेस्ट प्लॅनिंग (Test Planning): मॅन्युअल टेस्टिंग आणि Selenium ऑटोमेशन वापरून फंक्शनल आणि सिक्युरिटी टेस्टिंग करण्याचा निर्णय घेतला जातो.
3. टेस्ट केस डेव्हलपमेंट (Test Case Development): उदा. "योग्य क्रेडेन्शियल्स वापरून नोंदणीकृत वापरकर्ता लॉगिन करू शकतो का ते तपासा." युजरनेम आणि पासवर्डसारखा इनपुट डेटा तयार केला जातो.
4. एन्व्हायरनमेंट सेटअप (Environment Setup): स्टेजिंग सर्व्हर कॉन्फिगर करणे, ॲप्लिकेशन डिप्लॉय करणे, टेस्ट डेटाबेस सेटअप करणे आणि Selenium WebDriver इन्स्टॉल करणे.
5. टेस्ट एक्झिक्युशन (Test Execution): लॉगिन टेस्ट केस चालवले जातात. योग्य इनपुट दिल्यास लॉगिन यशस्वी (Pass). चुकीचा इनपुट दिल्यास एरर मेसेज दिसतो (Pass).
6. टेस्ट केस क्लोजर (Test Case Closure): किती टेस्ट केस चालवले, किती पास झाले, किती फेल झाले आणि शिकलेले धडे यांचा समावेश असलेला टेस्ट सारांश अहवाल तयार केला जातो.

3.2 टेस्ट प्लॅनिंग (Test Planning)

टेस्ट प्लॅनिंग ही सॉफ्टवेअर टेस्टिंग प्रक्रियेतील एक महत्त्वाची पायरी आहे, ज्यामध्ये संपूर्ण टेस्टिंगसाठीची रणनीती, उद्दिष्टे, व्याप्ती (Scope), संसाधने, वेळापत्रक, भूमिका आणि जबाबदाऱ्या निश्चित केल्या जातात. या टप्प्याचा मुख्य परिणाम म्हणजे टेस्ट प्लॅन डॉक्युमेंट, जे संपूर्ण टेस्टिंग प्रक्रियेचा रोडमॅप म्हणून काम करते. टेस्ट प्लॅनिंगमुळे टेस्टिंग प्रक्रिया शिस्तबद्ध, कार्यक्षम आणि प्रोजेक्टच्या उद्दिष्टांशी सुसंगत राहते.

टेस्ट प्लॅनिंग म्हणजे काय? टेस्ट प्लॅनिंग ही सॉफ्टवेअर टेस्ट लाईफ सायकल (Software Test Life Cycle (STLC)) मधील एक अत्यंत महत्त्वाची अवस्था आहे. या टप्प्यात पुढील मूलभूत प्रश्नांची उत्तरे दिली जातात:

- काय टेस्ट करायचे आहे?
- टेस्टिंग कशा पद्धतीने करायचे आहे?
- टेस्टिंग कोण करणार आहे?
- टेस्टिंग कधी आणि कुठे केली जाणार आहे?

टेस्ट प्लॅनिंगमधील प्रमुख घटक (Components of Test Planning):

1. **टेस्ट प्लॅन तयार करणे (Preparing a Test Plan) :** टेस्ट प्लॅन हा अधिकृत डॉक्युमेंट असतो ज्यामध्ये संपूर्ण टेस्टिंग प्रक्रियेची माहिती असते. यात उद्दिष्टे, व्याप्ती, टेस्टिंग पद्धती, संसाधने, वेळापत्रक आणि जोखीम यांचा समावेश असतो.

2. **टेस्ट अप्रोच ठरवणे (Deciding the Test Approach):** या टप्प्यात टेस्टिंग कशा प्रकारे केली जाणार आहे हे ठरवले जाते:
   - मॅन्युअल टेस्टिंग
   - ऑटोमेशन टेस्टिंग
   - हायब्रिड अप्रोच (मॅन्युअल + ऑटोमेशन)
3. **एंट्री आणि एग्झिट निकष ठरवणे (Setting Up Criteria for Testing)**
   - एन्ट्री क्रायटेरिया (Entry Criteria): टेस्टिंग सुरू करण्यासाठी आवश्यक अटी
   - एक्झिट क्रायटेरिया (Exit Criteria): टेस्टिंग पूर्ण झाल्याचे निकष
   - यामुळे टेस्टिंग कधी सुरू करायची आणि कधी थांबवायची हे ठरते.
4. **जबाबदाऱ्या निश्चित करणे (Identifying Responsibilities):** प्रत्येक टीम मेंबरची भूमिका स्पष्ट केली जाते:
   - टेस्ट लीड
   - टेस्ट इंजिनियर
   - ऑटोमेशन इंजिनियरयामुळे गोंधळ टाळला जातो.
5. **मनुष्यबळ नियोजन (Staffing):** टेस्टिंगसाठी किती आणि कोणत्या कौशल्यांचे टेस्टर्स आवश्यक आहेत हे ठरवले जाते.
6. **संसाधनांची आवश्यकता (Resource Requirements):** यामध्ये पुढील गोष्टींचा समावेश होतो:
   - टेस्टिंग टूल्स
   - हार्डवेअर
   - सॉफ्टवेअर
   - टेस्टिंग एन्व्हायर्नमेंट
7. **टेस्ट डिलिव्हरेबल्स (Test Deliverables):** टेस्टिंग प्रक्रियेत तयार होणारे आउटपुट:
   - टेस्ट प्लॅन
   - टेस्ट केस
   - टेस्ट रिपोर्ट
   - डिफेक्ट रिपोर्ट
8. **टेस्टिंग टास्क्स (Testing Tasks):** टेस्टिंग दरम्यान करावयाच्या सर्व कामांची यादी:
   - टेस्ट केस तयार करणे
   - टेस्ट एक्झिक्युशन
   - डिफेक्ट लॉगिंग
   - रिपोर्टिंग

3.2.1 टेस्ट प्लॅन तयार करणे (Preparing a Test Plan)

टेस्ट प्लॅन तयार करणे ही संपूर्ण सॉफ्टवेअर टेस्टिंग प्रक्रियेचे दस्तऐवजीकरण (Documentation) करण्याची प्रक्रिया आहे. यामध्ये टेस्टिंगची उद्दिष्टे, व्याप्ती (Scope), टेस्ट स्ट्रॅटेजी, संसाधने, जबाबदाऱ्या, वेळापत्रक, जोखीम (Risks) आणि डिलिव्हरेबल्स यांचा समावेश असतो. यामुळे टेस्टिंग प्रक्रिया शिस्तबद्ध, कार्यक्षम आणि उद्दिष्ट-केंद्रित पद्धतीने पार पडते. टेस्ट प्लॅन तयार करणे हे सॉफ्टवेअर टेस्ट लाईफ सायकल (Software Test Life Cycle (STLC)) मधील Test Planning टप्प्यातील सर्वात महत्त्वाचे कार्य आहे. यामध्ये एक अधिकृत (Formal) डॉक्युमेंट तयार केले जाते, जे टेस्टिंग टीमसाठी मार्गदर्शक (Guideline) म्हणून काम करते.

टेस्ट प्लॅन तयार करण्याची प्रक्रिया:

1. **प्रोजेक्ट आवश्यकता समजून घेणे:** सर्वप्रथम सॉफ्टवेअरच्या आवश्यकता (Requirements) समजून घेतल्या जातात आणि त्यावर आधारित टेस्टिंगची उद्दिष्टे निश्चित केली जातात.
2. **टेस्टिंग उद्दिष्टे निश्चित करणे:** टेस्टिंगद्वारे काय साध्य करायचे आहे (उदा. गुणवत्ता तपासणी, डिफेक्ट शोधणे, विश्वसनीयता वाढवणे) हे ठरवले जाते.

3. **टेस्टिंगची व्याप्ती (Scope) निश्चित करणे:** कोणती फीचर्स/फंक्शनॅलिटी टेस्ट करायची आहेत (In-Scope) आणि कोणती नाहीत (Out-of-Scope) हे स्पष्ट केले जाते.
4. **टेस्ट स्ट्रॅटेजी निवडणे:** टेस्टिंग कशी केली जाणार आहे हे ठरवले जाते:
   - मॅन्युअल टेस्टिंग
   - ऑटोमेशन टेस्टिंग
   - मिक्स्ड (मॅन्युअल + ऑटोमेशन)
5. **संसाधनांची ओळख (Resource Identification):** आवश्यक संसाधने निश्चित केली जातात:
   - टेस्टर्स
   - टेस्ट एन्व्हायर्नमेंट
   - टेस्टिंग टूल्स
6. **भूमिका आणि जबाबदाऱ्या वाटप करणे:** प्रत्येक टीम मेंबरची भूमिका स्पष्ट केली जाते, जेणेकरून जबाबदाऱ्यांमध्ये गोंधळ होणार नाही.
7. **एंट्री आणि एग्झिट निकष ठरवणे**
   - एन्ट्री क्रायटेरिया (Entry Criteria): टेस्टिंग सुरू करण्याच्या अटी
   - एक्झिट क्रायटेरिया (Exit Criteria): टेस्टिंग पूर्ण झाल्याचे निकष
8. **टेस्टिंग वेळापत्रक (Schedule) तयार करणे:** टेस्टिंगसाठी वेळमर्यादा, टप्पे (Milestones) आणि डेडलाइन्स निश्चित केल्या जातात.
9. **जोखीम आणि प्रतिबंधक उपाय (Risks & Mitigation Plans):** संभाव्य अडचणी ओळखून त्यावर उपाययोजना आखल्या जातात.
10. **टेस्ट प्लॅन डॉक्युमेंटचे महत्त्व:** तयार केलेला टेस्ट प्लॅन डॉक्युमेंट हा:
    - टेस्टिंग टीमसाठी रोडमॅप म्हणून काम करतो
    - स्टेकहोल्डर्ससाठी प्रभावी संवाद साधन ठरतो
    - टेस्टिंग प्रक्रियेत स्पष्टता, सुसंगतता आणि गुणवत्ता सुनिश्चित करतो

टेस्ट प्लॅनचे सामान्य घटक (IEEE 829 स्टँडर्डनुसार): IEEE 829 सॉफ्टवेअर टेस्ट डॉक्युमेंटेशन स्टँडर्ड (IEEE 829 Software Test Documentation Standard) मध्ये टेस्ट प्लॅन तयार करण्यासाठी एक शिस्तबद्ध (Structured) स्वरूप दिलेले आहे. या स्टँडर्डमुळे टेस्टिंग प्रक्रिया सिस्टेमॅटिक, ट्रेसएबल आणि रिपीटेबल पद्धतीने पार पाडता येते. खाली टेस्ट प्लॅनचे प्रमुख घटक दिले आहेत:

1. **टेस्ट प्लॅन आयडेंटिफायर (Test Plan Identifier):** टेस्ट प्लॅन डॉक्युमेंटसाठी अद्वितीय ओळख क्रमांक. व्हर्जन कंट्रोल आणि संदर्भ क्रमांक
2. **परिचय (Introduction):** टेस्ट प्लॅनचा उद्देश आणि व्याप्ती, टेस्ट होणाऱ्या सिस्टीमचा आढावा, वापरलेली संज्ञा (Terminology) आणि डॉक्युमेंट कन्वेन्शन्स
3. **टेस्ट आयटम्स (Test Items):** टेस्ट करावयाचे सॉफ्टवेअर / हार्डवेअर घटक, व्हर्जन नंबर आणि बिल्ड माहिती, कॉन्फिगरेशन आयटम्स आणि डिलिव्हरेबल्स
4. **टेस्ट करावयाची वैशिष्ट्ये (Features to be tested):** तपासायच्या फंक्शनल आवश्यकता. पडताळायची परफॉर्मन्स वैशिष्ट्ये, तपासायची सिक्युरिटी फीचर्स, इंटिग्रेशन पॉइंट्सची पडताळणी.
5. **टेस्ट न करावयाची वैशिष्ट्ये (Features Not to be tested):** स्कोपबाहेरील फंक्शनॅलिटी, पुढे ढकललेली (Deferred) फीचर्स, टेस्टमध्ये न समाविष्ट केलेले थर्ड-पार्टी घटक, वगळण्यामागील कारणे.
6. **टेस्टिंग ॲप्रोच (Approach):** एकूण टेस्टिंग स्ट्रॅटेजी आणि पद्धती, टेस्ट लेव्हल्स युनिट, इंटिग्रेशन, सिस्टिम, ॲक्सेप्टन्स, टेस्ट प्रकार फंक्शनल, परफॉर्मन्स, सिक्युरिटी, युजेबिलिटी, वापरलेली टेस्ट तंत्रे आणि पद्धती
7. **आयटम पास/फेल निकष (Item Pass/Fail Criteria):** प्रत्येक टेस्ट आयटमसाठी ॲक्सेप्टन्स निकष, यश/अपयश मर्यादा, क्वालिटी गेट्स आणि निर्णय बिंदू, एग्झिट निकषांची व्याख्या.

8. **सस्पेन्शन निकष आणि पुन्हा सुरू करण्याच्या अटी (Suspension Criteria and Resumption Requirements):** टेस्टिंग थांबवण्याच्या अटी. टेस्ट सस्पेंड करण्यास कारणीभूत ठरणारे गंभीर दोष. टेस्टिंग पुन्हा सुरू करण्याच्या अटी, एस्कलेशन प्रक्रिया.
9. **टेस्ट डिलिव्हरेबल्स (Test Deliverables):** टेस्ट डिझाईन स्पेसिफिकेशन, टेस्ट केस स्पेसिफिकेशन, टेस्ट प्रोसीजर डॉक्युमेंट, टेस्ट एक्झिक्युशन लॉग्स आणि रिपोर्ट्स, डिफेक्ट रिपोर्ट्स आणि समरी रिपोर्ट्स.
10. **टेस्टिंग टास्क्स (Testing Tasks):** टेस्टिंग क्रियांची सविस्तर यादी, टास्क अवलंबित्व आणि क्रम, संसाधनांचे वाटप, प्रत्येक टास्कसाठी लागणारा वेळ व प्रयत्न.
11. **पर्यावरणीय गरजा (Environmental Needs):** हार्डवेअर आवश्यकता आणि कॉन्फिगरेशन, सॉफ्टवेअर आवश्यकता (OS, डेटाबेस, टूल्स), नेटवर्क आणि कनेक्टिव्हिटी गरजा, टेस्ट डेटा आवश्यकता, सिक्युरिटी आणि ॲक्सेस कंट्रोल.
12. **जबाबदाऱ्या (Responsibilities):** टेस्ट मॅनेजरच्या भूमिका, टेस्ट इंजिनियरच्या जबाबदाऱ्या, डेव्हलपर आणि बिझनेस ॲनालिस्ट सहभाग, स्टेकहोल्डर रिव्ह्यू आणि मंजुरी.
13. **स्टाफिंग आणि ट्रेनिंग गरजा (Staffing and Training Needs):** आवश्यक कौशल्ये आणि अनुभव, टीम मेंबर्ससाठी ट्रेनिंग आवश्यकता, संसाधन उपलब्धता, बॅकअप स्टाफची नेमणूक.
14. **वेळापत्रक (Schedule):** टेस्ट प्लॅनिंग माईलस्टोन्स (Milestones)m टेस्ट डिझाईन आणि डेव्हलपमेंट वेळापत्रक, टेस्ट एक्झिक्युशन टप्पे आणि कालावधी, रिव्ह्यू आणि मंजुरीचे वेळापत्रक, इतर प्रोजेक्ट ॲक्टिव्हिटीजवरील अवलंबित्व.
15. **जोखीम आणि पर्याय (Risks and Contingencies):** तांत्रिक जोखीम आणि त्यावरील उपाय, संसाधन जोखीम आणि बॅकअप प्लॅन, वेळापत्रकातील जोखीम आणि पर्यायी योजना, गुणवत्तेशी संबंधित जोखीम.
16. **मंजुरी (Approvals):** आवश्यक साइन-ऑफ आणि मंजुरी प्राधिकरण, रिव्ह्यू प्रक्रिया आणि निकष, स्टेकहोल्डर ॲक्सेप्टन्स प्रक्रिया, बदल नियंत्रण प्रक्रिया (Change Control).

उदाहरण (Example)

परिस्थिती (Scenario): ऑनलाइन बँकिंग ॲप्लिकेशन – टेस्ट प्लॅन प्रोसेस (Online Banking Application – Test Plan Process)

प्रक्रिया (Process): फंड ट्रान्सफर मॉड्यूल (Fund Transfer Module) साठी टेस्ट प्लॅन (Test Plan) तयार करणे

1. टेस्ट प्लॅन आयडेंटिफायर (Test Plan Identifier)
   - डॉक्युमेंट आयडी: OB-TP-FTM-001
   - व्हर्जन: 1.0 (Initial Draft)
2. परिचय (Introduction)
   - उद्देश (Purpose): ऑनलाइन बँकिंग ॲप्लिकेशन (Online Banking Application) मधील फंड ट्रान्सफर (Fund Transfer) फंक्शनॅलिटीची पडताळणी करणे.
   - व्याप्ती (Scope): लॉगिन, अकाउंट सिलेक्शन, रक्कम भरणे, ट्रान्सफर एक्झिक्युशन आणि कन्फर्मेशन यांचा समावेश.
   - सिस्टीम ओव्हरव्ह्यू: रिअल-टाइम व्यवहारांना समर्थन देणारे वेब-आधारित ॲप्लिकेशन.
3. टेस्ट आयटम्स (Test Items)
   - ॲप्लिकेशन बिल्ड: v5.2.3
   - मॉड्युल्स: फंड ट्रान्सफर, ट्रान्झॅक्शन हिस्ट्री (Fund Transfer, Transaction History)
   - कॉन्फिगरेशन्स: क्रोम v122, विंडोज 11, पोस्टग्रेसक्यूएल बॅकएंड (Chrome v122, Windows 11, PostgreSQL Backend)
4. टेस्ट करावयाची वैशिष्ट्ये (Features to be Tested)
   - स्वतःच्या खात्यांमधील ट्रान्सफर

- थर्ड-पार्टी अकाउंटला ट्रान्सफर
- ट्रान्झॅक्शन यश/अपयश संदेश
- रिअल-टाइम बॅलन्स अपडेट

5. टेस्ट न करावयाची वैशिष्ट्ये (Features Not to be Tested)
   - मोबाईल बँकिंग आवृत्ती (या रिलीजसाठी स्कोपबाहेर)
   - आंतरराष्ट्रीय फंड ट्रान्सफर (Deferred)
   - थर्ड-पार्टी पेमेंट गेटवे
6. ॲप्रोच **(**Approach**)**
   - टेस्ट लेव्हल्स: युनिट (Unit) डेव्हलपर्स (Developers) कडून, इंटिग्रेशन (Integration), सिस्टिम(System), यूएटी(UAT).
   - टेस्ट प्रकार: फंक्शनल, सिक्युरिटी, युजेबिलिटी, परफॉर्मन्स
   - तंत्रे: ब्लॅक-बॉक्स टेस्टिंग, बाउंडरी व्हॅल्यू ॲनालिसिस, इक्विव्हॅलन्स पार्टिशनिंग**.**
7. आयटम पास/फेल निकष (Item Pass/Fail Criteria)
   - पास(Pass): ट्रान्सफर यशस्वी होतो, बॅलन्स अपडेट होतो आणि कन्फर्मेशन संदेश दिसतो.
   - फेल (Fail): ट्रान्झॅक्शन फेल होते, चुकीचा बॅलन्स दिसतो किंवा सिस्टीम क्रॅश होते.
8. सस्पेन्शन आणि पुन्हा सुरू करण्याच्या अटी (Suspension Criteria and Resumption Requirements)
   - लॉगिन सर्व्हिस किंवा ट्रान्झॅक्शन डेटाबेस उपलब्ध नसल्यास टेस्टिंग थांबवणे.
   - सर्व्हिस स्थिर झाल्यानंतर आणि डिफेक्ट्स सोडवल्यानंतर टेस्टिंग पुन्हा सुरू करणे.
9. टेस्ट डिलिव्हरेबल्स (Test Deliverables): टेस्ट प्लॅन (हा डॉक्युमेंट)
   - टेस्ट केसेस (TC_FT_001 ते TC_FT_050)
   - टेस्ट एक्झिक्युशन रिपोर्ट
   - डिफेक्ट लॉग्स
   - फायनल टेस्ट समरी रिपोर्ट
10. टेस्टिंग टास्क्स (Testing Tasks): रीक्वायरमेंट रिव्ह्यू → टेस्ट केस डिझाईन → एन्व्हायरनमेंट सेटअप → टेस्ट एक्झिक्युशन → डिफेक्ट लॉगिंग → री-टेस्टिंग → फायनल रिपोर्टिंग (Requirement Review → Test Case Design → Environment Setup →Test Execution → Defect Logging → Re-testing → Final Reporting)
11. पर्यावरणीय गरजा (Environmental Needs)
    - हार्डवेअर: 2 टेस्ट सर्व्हर्स, 5 क्लायंट मशीन
    - सॉफ्टवेअर: सेलेनियम, वेबड्रायव्हर, जेमीटर, जीरा (डिफेक्ट ट्रॅकिंग साठी) (Selenium WebDriver, JMeter, Jira (for Defect Tracking))
    - टेस्ट डेटा: पुरेसा बॅलन्स असलेली सॅम्पल ग्राहक खाती
12. जबाबदाऱ्या (Responsibilities)
    - टेस्ट मॅनेजर (Test Manager): एकूण प्लॅनिंग आणि रिपोर्टिंग
    - टेस्ट लीड (Test Lead): टास्क वाटप आणि दररोजचे मॉनिटरिंग
    - टेस्ट इंजिनिअर्स (Test Engineers): मॅन्युअल आणि ऑटोमेटेड टेस्ट्सची अंमलबजावणी
    - डेव्हलपर्स (Developers): डिफेक्ट्स दुरुस्ती आणि नवीन बिल्ड्स देणे
13. स्टाफिंग आणि ट्रेनिंग गरजा (Staffing and Training Needs)
    - टीम: 1 टेस्ट लीड + 4 टेस्ट इंजिनिअर्स + 1 ऑटोमेशन इंजिनिअर (1 Test Lead + 4 Test Engineers + 1 Automation Engineer)
    - ट्रेनिंग: सिक्युरिटी टेस्टिंग वर्कशॉप, सेलेनियम रिफ्रेशर **(**Security Testing Workshop, Selenium Refresher)

14. वेळापत्रक (Schedule)
    - टेस्ट प्लॅनिंग: आठवडा 1
    - टेस्ट केस डिझाईन: आठवडा 2–3
    - एन्व्हायर्नमेंट सेटअप: आठवडा 3
    - टेस्ट एक्झिक्युशन: आठवडा 4–6
    - रिपोर्टिंग आणि क्लोजर: आठवडा 7
15. जोखीम आणि पर्याय (Risks and Contingencies)
    - तांत्रिक जोखीम: ट्रान्झॅक्शन एपीआय डाऊनटाइम (Transaction API downtime) → कॉन्टिन्जन्सी: मॉक एपीआय (Contingency: Mock API) वापरणे.
    - संसाधन जोखीम: टेस्ट इंजिनियर अनुपलब्ध → बॅकअप संसाधन नियुक्त
    - वेळापत्रक जोखीम: बिल्ड डिलिव्हरी उशीर → एक्झिक्युशन प्लॅनमध्ये बदल
16. मंजुरी (Approvals)
    - टेस्ट मॅनेजर: ॲप्रूव्हड (Test Manager: Approved)
    - QA हेड: ॲप्रूव्हड (QA Head: Approved)
    - प्रोजेक्ट मॅनेजर: ॲप्रूव्हड (Project Manager: Approved)
    - बिझनेस स्टेकहोल्डर: पेंडिंग (Business Stakeholder: Pending)

3.2.2 टेस्ट ॲप्रोच ठरवणे (Deciding the Test Approach)

टेस्ट ॲप्रोच (Test Approach) याला टेस्ट स्ट्रॅटेजी (Test Strategy) असेही म्हणतात. यामध्ये टेस्टिंग दरम्यान कोणती एकूण पद्धत (methodology) वापरली जाणार आहे हे स्पष्ट केले जाते. टेस्टिंग कसे केले जाईल, कोणकोणत्या टेस्ट लेव्हल्स वापरल्या जातील, कोणती तंत्रे (techniques) वापरली जातील आणि गुणवत्ता (quality) कशी सुनिश्चित केली जाईल याचे स्पष्टीकरण टेस्ट ॲप्रोचमध्ये दिलेले असते. योग्यरीत्या परिभाषित केलेला टेस्ट ॲप्रोच हा टेस्टिंग टीमसाठी मार्गदर्शक (guideline) म्हणून कार्य करतो आणि टेस्टिंगच्या सर्व टप्प्यांमध्ये सातत्य (consistency) राखण्यास मदत करतो.

टेस्ट ॲप्रोचवर परिणाम करणारे घटक (Factors Influencing the Test Approach)

- **प्रोजेक्टचा आकार आणि गुंतागुंत (Project Size and Complexity):** मोठ्या आणि क्लिष्ट सिस्टिम्ससाठी अनेक स्तरांवर टेस्टिंग करणे आवश्यक असते.
- **ॲप्लिकेशनची गंभीरता (Criticality of the Application):** सेफ्टी-क्रिटिकल किंवा फायनान्शियल सिस्टिम्ससाठी अधिक कठोर टेस्टिंग स्ट्रॅटेजी आवश्यक असते. (उदा. बाउंडरी व्हॅल्यू ॲनालिसिस, सिक्युरिटी टेस्टिंग) (Eg. Boundary Value Analysis, Security Testing)
- **वेळ आणि बजेट मर्यादा (Time and Budget Constraints):** मर्यादित संसाधनांमुळे सर्वसमावेशक टेस्टिंग शक्य नसते, त्यामुळे प्राधान्य (prioritization) ठरवावे लागते.
- **टूल्स आणि टेक्नॉलॉजी (Tools and Technology):** ऑटोमेशन टूल्स (उदा. सेलेनियम, जेयुनिट) उपलब्ध असल्यास ऑटोमेटेड रिग्रेशन टेस्टिंगवर अधिक भर दिला जातो.
- **टीमचे कौशल्य आणि अनुभव (Team Skills and Experience):** टेस्टर्सचा डोमेन नॉलेज आणि टूल्समधील अनुभव टेस्ट ॲप्रोच ठरवण्यावर प्रभाव टाकतो.

टेस्ट ॲप्रोचचे घटक

1. **टेस्ट लेव्हल्स (Levels of Testing)**
    - युनिट टेस्टिंग (Unit Testing)
    - इंटिग्रेशन टेस्टिंग (Integration Testing)
    - सिस्टिम टेस्टिंग (System Testing)
    - युजर ॲक्सेप्टन्स टेस्टिंग (User Acceptance Testing)

2. टेस्ट प्रकार (Types of Testing)
- फंक्शनल टेस्टिंग (Functional Testing)
- नॉन-फंक्शनल टेस्टिंग (Non-functional Testing)
  – परफॉर्मन्स (Performance)
  – सिक्युरिटी (Security)
  – युजेबिलिटी (Usability)
- रिग्रेशन टेस्टिंग (Regression Testing)

3. टेस्ट तंत्रे आणि पद्धती (Test Techniques & Methods)
- ब्लॅक-बॉक्स टेस्टिंग (Black-box Testing)
  – बाउंडरी व्हॅल्यू ॲनालिसिस (Boundary Value Analysis)
- व्हाइट-बॉक्स टेस्टिंग (White-box Testing)
  – पाथ कव्हरेज (Path Coverage)
- एक्सप्लोरेटरी टेस्टिंग (Exploratory Testing)

4. टेस्ट ऑटोमेशन विचार (Test Automation Considerations)
- टूल्सची निवड (उदा. Selenium, JUnit)
- ऑटोमेशनसाठी योग्य टेस्ट केसेस ओळखणे
- ऑटोमेशन फ्रेमवर्क डिझाइन करणे

5. रिस्क-आधारित टेस्टिंग (Risk-Based Testing)
- उच्च जोखीम असलेले, बिझनेस-क्रिटिकल किंवा क्लिष्ट मॉड्युल्स प्रथम टेस्ट करणे.

उदाहरण (Example):

परिस्थिती (Scenario): रुग्ण नोंदणी (Patient Registration), डॉक्टर शेड्युलिंग (Doctor Scheduling), बिलिंग (Billing) आणि वैद्यकीय अहवाल (Medical Reports) अशी मॉड्युल्स असलेली Healthcare Management System विकसित केली जात आहे.

प्रक्रिया (Process): हेल्थकेअर मॅनेजमेंट सिस्टीम साठी टेस्टिंग मेथडॉलॉजी निश्चित करणे.

1. टेस्टिंगचे स्तर (Levels of Testing)
   - युनिट टेस्टिंग (Unit Testing) →रुग्ण नोंदणी फॉर्ममधील इनपुट फील्ड्स, बिलिंग कॅल्क्युलेशन फंक्शन्सची पडताळणी.
   - इंटिग्रेशन टेस्टिंग (Integration Testing) →डॉक्टर शेड्युलिंग आणि बिलिंग मॉड्युल्समधील डेटा फ्लो सुरळीत आहे का हे तपासणे.
   - सिस्टिम टेस्टिंग (System Testing) →पूर्ण वर्कफ्लो तपासणे (रजिस्ट्रेशन → शेड्युलिंग → बिलिंग → रिपोर्ट जनरेशन).
   - युजर ॲक्सेप्टन्स टेस्टिंग (UAT) →हॉस्पिटल स्टाफकडून सिस्टिम त्यांच्या गरजा पूर्ण करते का याची खात्री करून घेणे.
2. टेस्टिंगचे प्रकार (Types of Testing)
   - फंक्शनल टेस्टिंग (Functional Testing) →योग्य रुग्ण नोंदणी, डॉक्टर उपलब्धता तपासणी आणि इनव्हॉइस जनरेशनची पडताळणी.
   - परफॉर्मन्स टेस्टिंग (Performance Testing) →पीक लोड हाताळण्याची क्षमता तपासणे (उदा. 500+ एकाच वेळी रुग्ण नोंदणी).
   - सिक्युरिटी टेस्टिंग (Security Testing) →संवेदनशील रुग्ण डेटा सुरक्षित ठेवणे (HIPAA पालन, Role-based Access Control).
   - युजेबिलिटी टेस्टिंग (Usability Testing) → हॉस्पिटल स्टाफसाठी सिस्टिम वापरण्यास सोपी आहे का हे तपासणे.

- रिग्रेशन टेस्टिंग (Regression Testing) →नवीन अपडेट्सनंतर (उदा. नवीन बिलिंग नियम) महत्त्वाच्या टेस्ट केसेस पुन्हा चालवणे.

3. टेस्टिंग तंत्रे आणि पद्धती (Techniques & Methods)
   - ब्लॅक-बॉक्स टेस्टिंग (Black-box Testing) →रजिस्ट्रेशन फॉर्म वैध / अवैध इनपुट्ससह तपासणे.
   - व्हाइट-बॉक्स टेस्टिंग (White-box Testing) →बिलिंग अल्गोरिदम्सची कव्हरेज आणि अचूकता तपासणे.
   - एक्सप्लोरेटरी टेस्टिंग (Exploratory Testing) →अपवादात्मक परिस्थिती तपासणे (उदा. डुप्लिकेट Patient ID, ओव्हरलॅप होणारे शेड्युल्स).
4. ऑटोमेशन बाबी (Automation Considerations)
   - बिलिंग कॅल्क्युलेशन्स आणि रिपोर्ट जनरेशनसाठी रिग्रेशन टेस्ट केसेस Selenium वापरून ऑटोमेट करणे.
   - मॉड्युल्समधील रुग्ण डेटा एक्सचेंजसाठी API टेस्टिंग टूल्स (Postman / REST-assured) वापरणे.
   - पीक वेळेत डॉक्टर शेड्युलिंगसाठी लोड टेस्टिंग करण्यासाठी JMeter वापरणे.
5. रिस्क-आधारित टेस्टिंग (Risk-Based Testing)
   - उच्च जोखीम (High Risk): रुग्ण डेटा गोपनीयता, बिलिंग अचूकता, वैद्यकीय अहवालांची अखंडता.
   - मध्यम जोखीम (Medium Risk): डॉक्टर शेड्युलिंग कॉन्फ्लिक्ट्स.
   - कमी जोखीम (Low Risk): UI प्रेफरन्सेस (थीम, भाषा पर्याय).

3.2.3 टेस्टिंगसाठी निकष निश्चित करणे (Setting Up Criteria for Testing)

टेस्टिंगसाठी निकष निश्चित करणे म्हणजे टेस्टिंग कधी सुरू करायचे, कधी थांबवायचे आणि कधी पूर्ण करायचे यासाठी स्पष्ट एंट्री क्रायटेरिया (Entry Criteria) आणि एग्झिट क्रायटेरिया (Exit Criteria) ठरवण्याची प्रक्रिया होय. हे निकष क्वालिटी गेट्स (Quality Gates) म्हणून काम करतात आणि टेस्टिंग प्रक्रिया सुसंगत, मोजता येण्यासारखी आणि प्रोजेक्टच्या उद्दिष्टांशी सुसंगत आहे याची खात्री करतात. IEEE 829 मानक (Standard) आणि प्रचलित टेस्टिंग पद्धतीनुसार, एंट्री आणि एग्झिट क्रायटेरिया हे टेस्टिंग ॲक्टिव्हिटीज सुरू आणि थांबवण्यासाठी मोजता येणाऱ्या अटी (Measurable Conditions) प्रदान करतात.

एंट्री क्रायटेरिया (Entry Criteria)

एंट्री क्रायटेरिया म्हणजे टेस्टिंग सुरू होण्यापूर्वी पूर्ण करणे आवश्यक असलेल्या अटी. या अटींमुळे टेस्टिंग टीमकडे प्रभावीपणे टेस्टिंग सुरू करण्यासाठी आवश्यक सर्व गोष्टी उपलब्ध आहेत याची खात्री होते.

सामान्यतः एंट्री क्रायटेरिया मध्ये खालील बाबींचा समावेश असतो:

- टेस्ट प्लॅन (Test Plan) उपलब्ध असणे
- स्थिर आणि टेस्ट करण्यायोग्य बिल्ड (Stable Build) उपलब्ध असणे
- टेस्ट डेटा (Test Data) तयार असणे
- टेस्ट एन्व्हायर्नमेंट (Test Environment) योग्यरित्या सेट केलेले असणे

एंट्री क्रायटेरिया मुळे टेस्टिंग तयार आणि नियंत्रित परिस्थितीत सुरू होते.

एग्झिट क्रायटेरिया (Exit Criteria)

एग्झिट क्रायटेरिया म्हणजे टेस्टिंग पूर्ण करण्यासाठी आवश्यक असलेल्या अटी. हे निकष टेस्टिंग ठराविक गुणवत्तेपर्यंत पूर्ण झाले आहे याची खात्री करतात.

सामान्यतः एग्झिट क्रायटेरिया मध्ये पुढील बाबी असतात:

- ठरवलेल्या टेस्ट केसेस यशस्वीपणे चालवलेल्या असणे
- गंभीर (Critical) आणि उच्च-प्राधान्य (High Priority) डिफेक्ट्स दुरुस्त झालेले असणे
- कोड कव्हरेज (Code Coverage) उद्दिष्टे पूर्ण झालेली असणे
- सर्व टेस्ट डिलिव्हरेबल्स (Test Reports, Logs, Summary) पूर्ण झालेली असणे

एग्झिट क्रायटेरिया हे दर्शवतात की टेस्टिंगने आपली उद्दिष्टे साध्य केली आहेत आणि सॉफ्टवेअर रिलीज किंवा पुढील फेजसाठी तयार आहे.

महत्त्व (Importance): एंट्री आणि एग्झिट क्रायटेरिया हे:

- प्रोजेक्ट मॅनेजर्स आणि QA टीमसाठी निर्णय घेण्याचे गेट्स म्हणून काम करतात
- प्रोजेक्ट उद्दिष्टांशी सुसंगती राखतात
- जोखीम (Risk) कमी करतात
- सॉफ्टवेअरची गुणवत्ता (Quality Assurance) सुनिश्चित करतात

म्हणूनच, टेस्टिंगसाठी स्पष्ट निकष निश्चित करणे हे प्रभावी आणि यशस्वी टेस्टिंग प्रक्रियेचा अत्यंत महत्त्वाचा भाग आहे.

उदाहरण (Example):

परिस्थिती (Scenario): हेल्थकेअर मॅनेजमेंट सिस्टिम (मॉड्यूल्स – Patient Registration, Doctor Scheduling, Billing, Medical Reports)

प्रक्रिया (Process):

एंट्री क्रायटेरिया (Entry Criteria):

- Patient Registration, Doctor Scheduling मॉड्यूल्सचे युनिट टेस्टिंग पूर्ण झालेले असून ती टेस्ट एन्व्हायर्नमेंटमध्ये इंटिग्रेट केलेली आहेत.
- टेस्ट डेटा (डमी रुग्ण, डॉक्टर, बिलिंग रेकॉर्ड्स) उपलब्ध आहे.
- टेस्ट प्लॅन आणि टेस्ट केसेसचे रिव्ह्यू झालेले असून त्यांना मंजुरी मिळालेली आहे.
- QA टीमला हॉस्पिटल डेटाबेस टेस्ट सर्व्हरचा ॲक्सेस उपलब्ध आहे.

एग्झिट क्रायटेरिया (Exit Criteria):

- Registration, Scheduling, Billing आणि Reports साठी सर्व Functional Test Cases ≥ 95% पास रेटसह यशस्वीपणे एक्झिक्यूट झालेले आहेत.
- Severity-1 (Critical) किंवा Severity-2 (Major) प्रकारचे कोणतेही ओपन डिफेक्ट्स शिल्लक नाहीत.
- परफॉर्मन्स टेस्टिंग द्वारे सिस्टिम 500 पेक्षा अधिक Concurrent Users हाताळू शकते हे सिद्ध झाले आहे.
- सिक्युरिटी टेस्टिंग द्वारे रुग्ण डेटा संरक्षण नियमांचे (Patient Data Protection Regulations) पालन होत असल्याची खात्री मिळालेली आहे.
- फायनल टेस्ट समरी रिपोर्ट ला QA Manager आणि हॉस्पिटल IT हेड यांची मंजुरी मिळालेली आहे.

निरीक्षण (Observation):

एंट्री क्रायटेरिया लागू केल्यामुळे QA टीमने टेस्टिंग फक्त मॉड्यूल्स, डेटा आणि एन्व्हायर्नमेंट पूर्णपणे तयार झाल्यानंतरच सुरू केले, ज्यामुळे वेळ आणि संसाधनांचा अपव्यय टाळला गेला. एग्झिट क्रायटेरिया मुळे सिस्टिमने कार्यात्मक (Functional), कार्यक्षमता (Performance) आणि सुरक्षा (Security) या सर्व आवश्यकता पूर्ण केल्याचा आत्मविश्वास मिळाला. त्यामुळे ही प्रणाली प्रत्यक्ष हॉस्पिटल वातावरणात सुरक्षितपणे डिप्लॉय करण्यास योग्य आहे हे सुनिश्चित झाले.

Table 3.1: एंट्री क्रायटेरिया वि. एग्झिट क्रायटेरिया (Comparison between Entry Criteria and Exit Criteria)

Criteria (निकष)	Entry Criteria (एंट्री क्रायटेरिया)	Exit Criteria (एग्झिट क्रायटेरिया)
Definition (व्याख्या)	Preconditions that must be satisfied before testing can begin. टेस्टिंग सुरू करण्यापूर्वी पूर्ण होणे आवश्यक असलेल्या अटी.	Conditions that must be met to formally conclude testing. टेस्टिंग औपचारिकरित्या समाप्त करण्यासाठी पूर्ण कराव्या लागणाऱ्या अटी.
Purpose (उद्देश)	Ensures that testing starts in a controlled and prepared environment. टेस्टिंग योग्य तयारीनिशी आणि नियंत्रित वातावरणात सुरू होते याची खात्री करते.	Ensures that testing ends only after achieving defined quality goals. निश्चित केलेली गुणवत्ता प्राप्त झाल्यानंतरच टेस्टिंग संपते याची खात्री करते.
Focus (लक्ष)	Readiness of build, environment, test data, and resources.	Completeness and quality of executed testing activities.

	बिल्ड, वातावरण, टेस्ट डेटा आणि संसाधनांची तयारी.	केलेल्या टेस्टिंगची पूर्णता आणि गुणवत्ता.
When Applied (केव्हा लागू होते)	At the start of testing. टेस्टिंगच्या सुरुवातीला.	At the end of testing. टेस्टिंगच्या शेवटी.
Examples (General) (सामान्य उदाहरणे)	Approved test plan and test cases- Stable software build- Test environment ready- Test data prepared. मंजूर टेस्ट प्लॅन आणि टेस्ट केसेस• स्थिर सॉफ्टवेअर बिल्ड• टेस्ट वातावरण तयार• टेस्ट डेटा उपलब्ध	All test cases executed- Critical/major defects fixed- Code coverage target achieved- Test summary report approved. सर्व टेस्ट केसेस एक्झिक्यूट झाल्या• गंभीर दोष दुरुस्त झाले• कोड कव्हरेज टार्गेट पूर्ण• टेस्ट समरी रिपोर्ट मंजूर
Healthcare Management System Example (हेल्थकेअर सिस्टीम उदाहरण)	Patient registration billing modules deployed- Dummy patient/doctor data available- QA team access granted• रुग्ण नोंदणी व बिलिंग मॉड्यूल्स उपलब्ध• डमी रुग्ण/डॉक्टर डेटा तयार• QA टीमला अॅक्सेस दिला	- ≥95% test cases passed- No open Severity-1 / Severity-2 defects- Performance data security validated- Final test report approved. • 95% पेक्षा जास्त टेस्ट केसेस पास• Severity-1/2 दोष शिल्लक नाहीत• परफॉर्मन्स व डेटा सिक्युरिटी तपासली• अंतिम टेस्ट रिपोर्ट मंजूर
Benefit (फायदा)	Prevents premature or unprepared testing. अपूर्ण तयारीत टेस्टिंग सुरू होणे टाळते.	Prevents incomplete or low-quality testing closure. अपूर्ण किंवा निकृष्ट टेस्टिंग थांबवते.

3.2.4 जबाबदाऱ्या ओळखणे (Identifying Responsibilities)

सॉफ्टवेअर टेस्टिंगमध्ये जबाबदाऱ्या ओळखणे म्हणजे टेस्टिंग लाइफ सायकलमध्ये सहभागी असलेल्या सर्व स्टेकहोल्डर्सच्या भूमिका, कर्तव्ये आणि कामांची मालकी स्पष्टपणे निश्चित करण्याची प्रक्रिया होय. यामुळे जबाबदारी निश्चित होते, समन्वय सुरळीत राहतो आणि टेस्ट प्लॅनची प्रभावी अंमलबजावणी होते. IEEE 829 मानकांनुसार, टेस्ट प्लॅनिंगमध्ये जबाबदाऱ्या निश्चित करणे ही एक अत्यंत महत्त्वाची क्रिया आहे. जर भूमिका स्पष्टपणे ठरवलेल्या नसतील, तर कामांची पुनरावृत्ती होणे, काही कामे राहून जाणे आणि जबाबदारीची कमतरता निर्माण होऊ शकते.

मुख्य जबाबदाऱ्या (Key Responsibilities):

1. टेस्ट मॅनेजर / QA लीड (Test Manager / QA Lead): टेस्ट प्लॅन तयार करणे, टेस्ट स्ट्रॅटेजी निश्चित करणे, संसाधनांचे वाटप करणे आणि टेस्टिंग प्रगतीवर देखरेख ठेवणे.
2. टेस्ट इंजिनिअर्स / QA ॲनालिस्ट्स (Test Engineers / QA Analysts): टेस्ट केसेस डिझाइन करणे, विकसित करणे व एक्झिक्यूट करणे, डिफेक्ट्स लॉग करणे आणि रिपोर्ट्स तयार करणे.
3. डेव्हलपर्स (Developers): डिफेक्ट्स दुरुस्त करणे, युनिट टेस्टिंग करणे आणि तांत्रिक स्पष्टीकरणे प्रदान करणे.
4. बिझनेस ॲनालिस्ट्स / डोमेन एक्स्पर्ट्स (Business Analysts / Domain Experts): आवश्यकतांनुसार टेस्ट केसेसची पडताळणी करणे आणि ॲक्सेप्टन्स टेस्टिंगमध्ये मदत करणे.
5. स्टेकहोल्डर्स / क्लायंट्स (Stakeholders / Clients): टेस्ट डिलिव्हरेबल्सना मंजुरी देणे, अभिप्राय देणे आणि उत्पादनाच्या गुणवत्तेचा स्वीकार किंवा नकार देणे.

उदाहरण (Example)

परिस्थिती (Scenario): हेल्थकेअर मॅनेजमेंट सिस्टीम (मॉड्यूल्स – रुग्ण नोंदणी, डॉक्टर शेड्युलिंग, बिलिंग, वैद्यकीय अहवाल)

प्रक्रिया (Process):

1. टेस्ट मॅनेजर (Test Manager): सर्व हेल्थकेअर मॉड्यूल्ससाठी एकूण टेस्ट स्ट्रॅटेजी निश्चित करतो, टेस्ट प्लॅनला मंजुरी

देतो आणि टेस्ट एक्झिक्युशनवर देखरेख ठेवतो.

2. टेस्ट इंजिनिअर्स (Test Engineers): रुग्ण नोंदणी आणि बिलिंग मॉड्यूलसाठी टेस्ट केसेस तयार करतात व चालवतात; डिफेक्ट मॅनेजमेंट सिस्टीममध्ये त्रुटी (Defects) नोंदवतात.
3. डेव्हलपर्स (Developers): डॉक्टर शेड्युलिंगमधील कॉन्फ्लिक्ट्स आणि बिलिंग कॅलक्युलेशनमधील त्रुटी दुरुस्त करतात.
4. बिझनेस ॲनालिस्ट्स (Business Analysts): वैद्यकीय अहवाल निर्मिती प्रक्रिया हॉस्पिटलच्या कम्प्लायन्स (Compliance) मानकांशी सुसंगत आहे की नाही याची पडताळणी करतात.
5. हॉस्पिटल IT स्टेकहोल्डर्स (Hospital IT Stakeholders): डिप्लॉयमेंटपूर्वी अंतिम टेस्ट समरी रिपोर्टचे पुनरावलोकन करून त्याला मंजुरी देतात.

निरीक्षण (Observation):

जबाबदाऱ्या स्पष्टपणे ओळखल्यामुळे QA Team मध्ये गोंधळ व कामातील आच्छादन (overlaps) टाळता आले. Test Engineers यांनी चाचणी अंमलबजावणीवर (test execution) लक्ष केंद्रित केले, तर Developers यांनी त्रुटी (issues/defects) अधिक जलदरीत्या निराकरण केल्या. यामुळे Stakeholders यांना चाचणी परिणामांबाबत विश्वास प्राप्त झाला. या भूमिकेतील स्पष्टतेमुळे Healthcare Management System ने प्रकाशनापूर्वी (before release) कार्यात्मक (functional) तसेच नियामक (regulatory) आवश्यकता पूर्ण केल्या.

3.2.5 स्टाफिंग (Staffing)

सॉफ्टवेअर टेस्टिंगमधील स्टाफिंग (Staffing in Software Testing) म्हणजे सॉफ्टवेअर डेव्हलपमेंट लाइफ सायकल (SDLC) आवश्यक असलेल्या चाचणी क्रियाकलापांसाठी मानवी संसाधनांची ओळख करणे, वाटप करणे आणि व्यवस्थापन करणे ही प्रक्रिया आहे. यामुळे योग्य कौशल्ये असलेले योग्य लोक योग्य वेळी उपलब्ध राहतात आणि टेस्ट प्लॅन (Test Plan) प्रभावीपणे अंमलात आणता येतो. स्टाफिंग हे टेस्ट प्लॅनिंग (Test Planning) मधील एक अत्यंत महत्त्वाचे पाऊल आहे, कारण चाचणीची गुणवत्ता मोठ्या प्रमाणावर टेस्टिंग टीम (Testing Team) च्या कौशल्यांवर, अनुभवावर आणि उपलब्धतेवर अवलंबून असते.

भूमिका वाटप (Role Distribution):

1. **टेस्ट मॅनेजर / QA लीड (Test Manager / QA Lead)** – चाचणी धोरण निश्चित करतो, संसाधने व्यवस्थापित करतो, प्रगतीचे निरीक्षण करतो आणि स्टेकहोल्डर्स / क्लायंट्स (Stakeholders / Clients) यांच्याशी संवाद साधतो.
2. **टेस्ट इंजिनिअर्स / मॅन्युअल टेस्टर्स (Test Engineers / Manual Testers)** – टेस्ट केसेस डिझाइन करतात, अंमलात आणतात आणि निकाल नोंदवतात.
3. **ऑटोमेशन टेस्ट इंजिनिअर्स (Automation Test Engineers)** – रिग्रेशन, परफॉर्मन्स आणि पुनरावृत्ती चाचण्यांसाठी स्वयंचलित स्क्रिप्ट्स विकसित करतात.
4. **परफॉर्मन्स टेस्टर्स (Performance Testers)** – लोड, स्ट्रेस आणि स्केलेबिलिटी टेस्टिंग करतात.
5. **सिक्युरिटी टेस्टर्स (Security Testers)** – पेनिट्रेशन टेस्टिंग आणि व्हल्नरेबिलिटी अस्सेसमेंट करतात.
6. **डोमेन एक्सपर्ट्स / बिझनेस ॲनालिस्ट्स (Domain Experts / Business Analysts)** – बिझनेस दृष्टिकोनातून गरजा आणि टेस्ट सिनेरिओजची पडताळणी करतात.
7. **कॉन्फिगरेशन मॅनेजर्स / DevOps इंजिनिअर्स (Configuration Managers / DevOps Engineers)** – टेस्ट एन्व्हायर्नमेंट्स सेटअप आणि मेंटेन करतात.

स्टाफिंगविषयक बाबी (Staffing Considerations):

1. **स्किल सेट आवश्यकता (Skill Set Requirements):** तांत्रिक कौशल्ये (ऑटोमेशन, टूल्स) आणि विश्लेषणात्मक कौशल्ये (डोमेन नॉलेज, डिफेक्ट ॲनालिसिस) यांचा योग्य समतोल आवश्यक असतो.
2. **टीम साइज अंदाज (Team Size Estimation):** प्रोजेक्टचा व्याप, क्लिष्टता, वेळापत्रक आणि टेस्टिंग लेव्हल्स यावर अवलंबून असतो.
3. **ट्रेनिंग गरजा (Training Needs):** जर टीमला विशिष्ट टूल्स किंवा पद्धतींचे ज्ञान नसेल, तर योग्य प्रशिक्षणाची व्यवस्था करावी लागते.

4. **बॅकअप प्लॅनिंग (Backup Planning):** महत्त्वाचे सदस्य अनुपलब्ध असल्यास पर्यायी संसाधने नियुक्त केली पाहिजेत. स्टाफिंग नेहमी टेस्ट ॲप्रोच / Test Approach (मॅन्युअल विरुद्ध ऑटोमेशन, फंक्शनल विरुद्ध नॉन-फंक्शनल) यांच्याशी सुसंगत असावे, जेणेकरून उत्पादकता वाढवता येईल.

उदाहरण(Example)

परिस्थिती(Scenario): Online Food Delivery Application

स्टाफिंग योजना (Staffing Plan):

- 1 टेस्ट मॅनेजर (Test Manager) – टेस्ट प्लॅनिंग, प्रगतीचे निरीक्षण आणि डेव्हलपर्स व बिझनेस टीमशी संवाद साधतो.
- 2 मॅन्युअल टेस्ट इंजिनिअर्स (Manual Test Engineers) – मुख्य कार्ये तपासतात (यूजर रजिस्ट्रेशन, मेनू ब्राउझिंग, कार्ट, पेमेंट्स).
- 1 ऑटोमेशन इंजिनिअर (Automation Engineer) – अनेक डिव्हाइसेसवर रिग्रेशन टेस्टिंगसाठी Selenium / Appium स्क्रिप्ट्स तयार करतो.
- 1 परफॉर्मन्स टेस्टर (Performance Tester) – पीक ऑर्डर वेळेत सर्व्हर लोड तपासण्यासाठी JMeter वापरतो.
- 1 सिक्युरिटी टेस्टर (Security Tester) – पेमेंट गेटवे आणि यूजर ऑथेंटिकेशनवर पेनिट्रेशन टेस्टिंग करतो.
- 1 बिझनेस ॲनालिस्ट / डोमेन एक्सपर्ट (Business Analyst / Domain Expert) – ऑर्डर फ्लो, ऑफर्स आणि डिलिव्हरी ट्रॅकिंग बिझनेस नियमांनुसार आहेत की नाही हे सुनिश्चित करतो.

निरीक्षण(Observation): संतुलित स्टाफिंग रचनेमुळे ऑनलाइन फूड डिलिव्हरी प्रोजेक्टमध्ये संपूर्ण कव्हरेज मिळाले:

- मॅन्युअल टेस्टर्सनी युजेबिलिटी आणि फंक्शनल समस्या शोधल्या.
- ऑटोमेशनमुळे रिग्रेशन सायकलमध्ये वेळ वाचला.
- परफॉर्मन्स आणि सिक्युरिटी टेस्टर्सनी सिस्टमची मजबुती आणि सुरक्षितता तपासली.
- बिझनेस ॲनालिस्टने कस्टमर अपेक्षांशी सुसंगतता सुनिश्चित केली.

योग्य स्टाफिंगमुळे हा प्रोजेक्ट इफिशियंट, रिलायबल आणि कस्टमर-फ्रेंडली (Efficient, Reliable, and Customer-Friendly) ॲप्लिकेशन म्हणून यशस्वीपणे रिलीज झाला.

3.2.8 टेस्टिंग टास्क्स (Testing Tasks)

टेस्टिंग टास्क्स (Testing Tasks) म्हणजे सॉफ्टवेअर टेस्टिंग जीवनचक्रामध्ये (Software Testing Life Cycle – STLC) करण्यात येणाऱ्या संरचित क्रियाकलापांचा (Structured Activities) संच आहे. या टास्क्स टेस्ट प्लॅन (Test Plan) मध्ये दस्तऐवजीकरण केलेले असतात आणि योग्य संसाधन वाटप, प्रगतीचे निरीक्षण, दोष ओळख (Defect Identification) आणि गुणवत्ता हमी (Quality Assurance) सुनिश्चित करण्यास मदत करतात.

सामान्य टेस्टिंग टास्क्स (Typical Testing Tasks):

1. **रिक्वायरमेंट ॲनालिसिस (Requirement Analysis)**
   - फंक्शनल आणि नॉन-फंक्शनल रिक्वायरमेंट्सचे पुनरावलोकन करणे.
   - त्रुटी, अस्पष्टता आणि टेस्ट करण्यायोग्य अटी (Testable Conditions) ओळखणे.
2. **टेस्ट प्लॅनिंग (Test Planning)**
   - टेस्ट प्लॅन तयार करणे.
   - उद्दिष्टे, व्याप्ती (Scope), टेस्ट ॲप्रोच आणि क्रायटेरिया निश्चित करणे.
   - रिस्क्स, संसाधने, जबाबदाऱ्या आणि वेळापत्रक ओळखणे.
3. **टेस्ट डिझाइन आणि डेव्हलपमेंट (Test Design and Development)**
   - टेस्ट केसेस, टेस्ट डेटा आणि टेस्ट स्क्रिप्ट्स डिझाइन करणे.
   - टेस्ट डिझाइन स्पेसिफिकेशन्स तयार करणे.
   - आवश्यक असल्यास ऑटोमेशन स्क्रिप्ट्स विकसित करणे.
4. **टेस्ट एन्व्हायर्नमेंट सेटअप (Test Environment Setup)**
   - हार्डवेअर, ऑपरेटिंग सिस्टिम्स, डेटाबेसेस आणि नेटवर्क्स कॉन्फिगर करणे.

- टेस्टिंग टूल्स इन्स्टॉल करणे आणि ॲक्सेस कंट्रोल्स सेट करणे.

5. **टेस्ट एक्झिक्युशन (Test Execution)**
   - मॅन्युअल आणि ऑटोमेटेड टेस्ट्स चालवणे.
   - टेस्ट एक्झिक्युशन रिझल्ट्स लॉग करणे.
   - प्रत्यक्ष निकाल (Actual Results) आणि अपेक्षित निकाल (Expected Results) यांची तुलना करणे.
6. **डिफेक्ट मॅनेजमेंट (Defect Management)**
   - डिफेक्ट्स नोंदवणे आणि ट्रॅक करणे.
   - सिव्हेरिटी आणि प्रायोरिटी वर्गीकृत करणे.
   - दुरुस्ती नंतर रिटेस्ट करणे आणि निराकरण झालेली प्रकरणे बंद करणे.
7. **टेस्ट रिपोर्टिंग (Test Reporting)**
   - दैनिक / साप्ताहिक टेस्ट स्टेटस रिपोर्ट्स तयार करणे.
   - डिफेक्ट ट्रेंड ॲनालिसिस प्रदान करणे.
   - मेट्रिक्स आणि डॅशबोर्ड्स स्टेकहोल्डर्स / क्लायंट्स (Stakeholders / Clients) सोबत शेअर करणे.
8. **टेस्ट क्लोजर (Test Closure)**
   - एकूण टेस्टिंग क्रियाकलापांचा सारांश देणे.
   - मिळालेल्या कव्हरेज आणि गुणवत्तेचे मूल्यमापन करणे.
   - शिकलेले धडे (Lessons Learned) नोंदवणे आणि टेस्ट आर्टिफॅक्ट्स संग्रहित करणे.
9. **साईज आणि एफर्ट अंदाज (Size and Effort Estimation)**
   टेस्टिंग टास्क्स ओळखण्यासोबतच, टेस्टिंग कामाचा साईज (Size) आणि ते पूर्ण करण्यासाठी लागणारा एफर्ट (Effort) यांचा अंदाज लावणे अत्यंत आवश्यक असते.
10. **साईज अंदाज (Size Estimation)**
    - टेस्टिंगचा व्याप मोजतो (उदा. टेस्ट केसेसची संख्या, टेस्ट स्क्रिप्ट्स, डेटा सेट्स किंवा एक्झिक्युशन सायकल्स).
    - फंक्शन पॉइंट्स, युज केस पॉइंट्स (Function Points, Use Case Points) किंवा रिक्वायरमेंट क्लिष्टता (Requirement Complexity) यांसारख्या मेट्रिक्सद्वारे ठरवता येतो.
11. **एफर्ट अंदाज (Effort Estimation)**
    - सर्व टेस्टिंग टास्क्स पूर्ण करण्यासाठी लागणारे एकूण पर्सन-आवर्स किंवा पर्सन-डेझ ठरवतो.
    - टीमचा अनुभव, वापरलेली टूल्स, ऑटोमेशन लेव्हल आणि ॲप्लिकेशन क्लिष्टता यांसारख्या घटकांवर अवलंबून असतो.
    - स्टाफिंग लेव्हल्स, शेड्युलिंग टाइमलाईन्स आणि बजेटिंग कॉस्ट्स यांचे नियोजन करण्यास मदत करतो.

उदाहरण (Example)

परिस्थिती (Scenario): Healthcare Management System मॉड्यूल्स (Modules): पेशंट रजिस्ट्रेशन (Patient Registration), डॉक्टर शेड्यूलिंग (Doctor Scheduling), बिलिंग (Billing), मेडिकल रिपोर्ट्स (Medical Reports) या सिस्टिममध्ये वरील सर्व टेस्टिंग टास्क्स लागू करून, प्रत्येक मॉड्यूलची फंक्शनलिटी, परफॉर्मन्स, सिक्युरिटी आणि विश्वसनीयता (Reliability) प्रभावीपणे पडताळता येते.

टेस्टिंग कार्य (Testing Task)	आकाराचा अंदाज (Size Estimate)	प्रयत्नांचा अंदाज (Effort Estimate)
Requirement Analysis (आवश्यकता विश्लेषण)	30 requirements across 4 modules 4 मॉड्यूल्समध्ये एकूण 30 आवश्यकता	12 hours 12 तास
Test Planning (टेस्ट प्लॅनिंग)	1 Test Plan (with approach, criteria, risks) 1 टेस्ट प्लॅन (पद्धत, क्रायटेरिया, जोखीम सहित)	8 hours 8 तास
Test Design & Development	250 test cases (functional + non-functional)	70 hours

(टेस्ट डिझाइन व विकास)	सुमारे 250 टेस्ट केसेस (फंक्शनल + नॉन-फंक्शनल)	70 तास
Test Environment Setup (टेस्ट वातावरण सेटअप)	HMS server + DB + role-based access setup HMS सर्व्हर + डीबी + रोल-बेस्ड अॅक्सेस सेटअप	15 hours 15 तास
Test Execution (टेस्ट एक्झिक्युशन)	Execute 250 test cases over 3 cycles 3 सायकल्समध्ये 250 टेस्ट केसेस चालवणे	90 hours 90 तास
Defect Management (डिफेक्ट मॅनेजमेंट)	50 defects logged, tracked, and retested सुमारे 50 डिफेक्ट्स लॉग, ट्रॅक व री-टेस्ट	25 hours 25 तास
Test Reporting (टेस्ट रिपोर्टिंग)	Weekly reports + Final Test Summary साप्ताहिक रिपोर्ट्स + अंतिम टेस्ट समरी	10 hours 10 तास
Test Closure (टेस्ट क्लोजर)	Coverage report + Lessons Learned कव्हरेज रिपोर्ट + लेसन्स लर्न्ड	8 hours 8 तास

एकूण साईज (Total Size): सुमारे 250 टेस्ट केसेस (Test Cases)

एकूण एफर्ट (Total Effort): सुमारे 238 तास (Hours) ≈ 30 पर्सन-डेझ (Person-Days)

निरीक्षण (Observation):

टास्क्सची स्पष्ट व्याख्या तसेच साईज आणि एफर्ट अंदाज (Size and Effort Estimation) केल्यामुळे टीमला Healthcare Management System चे टेस्टिंग सर्वसमावेशकपणे (Comprehensively), ठरलेल्या वेळापत्रकात (Within Schedule) आणि योग्य रिसोर्स अलोकेशन (Resource Allocation) सह करता येते. यामुळे विलंब टाळता येतो आणि डिप्लॉयमेंट(Deployment) पूर्वी सिस्टिमच्या गुणवत्तेवर (System Quality) अधिक विश्वास निर्माण होतो.

3.3 टेस्ट मॅनेजमेंट (Test Management)

टेस्ट इन्फ्रास्ट्रक्चर मॅनेजमेंट (Test Infrastructure Management) आणि टेस्ट पीपल मॅनेजमेंट (Test People Management) टेस्ट मॅनेजमेंट (Test Management) ही सॉफ्टवेअर टेस्टिंगशी संबंधित सर्व क्रियाकलापांचे नियोजन (Planning), नियंत्रण (Controlling) आणि निरीक्षण (Monitoring) करण्याची प्रक्रिया आहे, ज्यामुळे सॉफ्टवेअर टेस्टिंगची उद्दिष्टे पूर्ण होतात. यामध्ये टेस्टिंगसाठी आवश्यक असलेली इन्फ्रास्ट्रक्चर (Infrastructure) तसेच टेस्टिंग क्रियाकलाप करणाऱ्या लोकांचे व्यवस्थापन (People Management) या दोन्हींचा समावेश होतो. टेस्ट मॅनेजमेंटमुळे टेस्टिंग प्रक्रिया संरचित (Structured), खर्च-किफायतशीर (Cost-Effective) आणि कार्यक्षम (Efficient) पद्धतीने पार पडते.

टेस्ट मॅनेजमेंट (Test Management) मुख्यतः खालील दोन क्षेत्रांवर लक्ष केंद्रित करते.

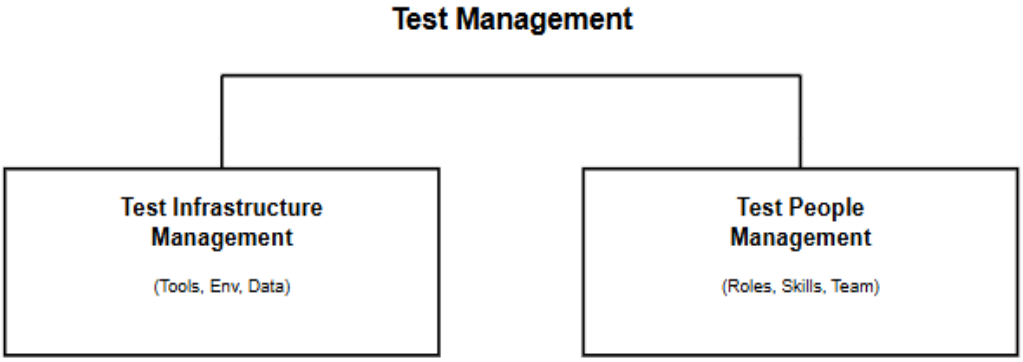

Fig 3.2: टेस्ट मॅनेजमेंट (Test Management)

1. **टेस्ट इन्फ्रास्ट्रक्चर मॅनेजमेंट (Test Infrastructure Management):** टेस्टिंग करण्यासाठी आवश्यक असलेली सर्व संसाधने हार्डवेअर, सॉफ्टवेअर, टेस्ट टूल्स, टेस्ट एन्व्हायर्नमेंट्स आणि टेस्ट डेटा यांचे व्यवस्थापन करते. योग्य इन्फ्रास्ट्रक्चरमुळे टेस्ट केसेसची अंमलबजावणी अडथळ्यांशिवाय (Without Bottlenecks) सुरळीतपणे होते.
2. **टेस्ट पीपल मॅनेजमेंट (Test People Management):** टेस्टिंग टीमचे व्यवस्थापन, भूमिका आणि जबाबदाऱ्या वाटप करणे, परफॉर्मन्स ट्रॅक करणे आणि आवश्यक प्रशिक्षण देणे यांचा समावेश होतो. यामुळे मानवी संसाधनांचा (Human Resources) प्रभावी वापर होऊन उच्च दर्जाचे (Quality) परिणाम साध्य होतात.

3.3.1 टेस्ट इन्फ्रास्ट्रक्चर मॅनेजमेंट (Test Infrastructure Management)

टेस्ट इन्फ्रास्ट्रक्चर मॅनेजमेंट (Test Infrastructure Management) म्हणजे प्रभावी सॉफ्टवेअर टेस्टिंगसाठी आवश्यक असलेल्या संसाधने, रिपॉझिटरीज आणि टूल्स यांचे नियोजन (Planning), संघटन (Organizing) आणि देखभाल (Maintaining) करण्याची प्रक्रिया होय. यामुळे टेस्ट केसेस (Test Cases), डिफेक्ट्स (Defects) आणि कॉन्फिगरेशन्स (Configurations) व्यवस्थितपणे साठवली (Stored), ट्रॅक केली (Tracked) आणि व्यवस्थापित (Managed) जातात, ज्यामुळे टेस्ट एक्झिक्युशन (Test Execution) सुरळीतपणे पार पडते.

टेस्ट इन्फ्रास्ट्रक्चर मॅनेजमेंटचे घटक (Components of Test Infrastructure Management):

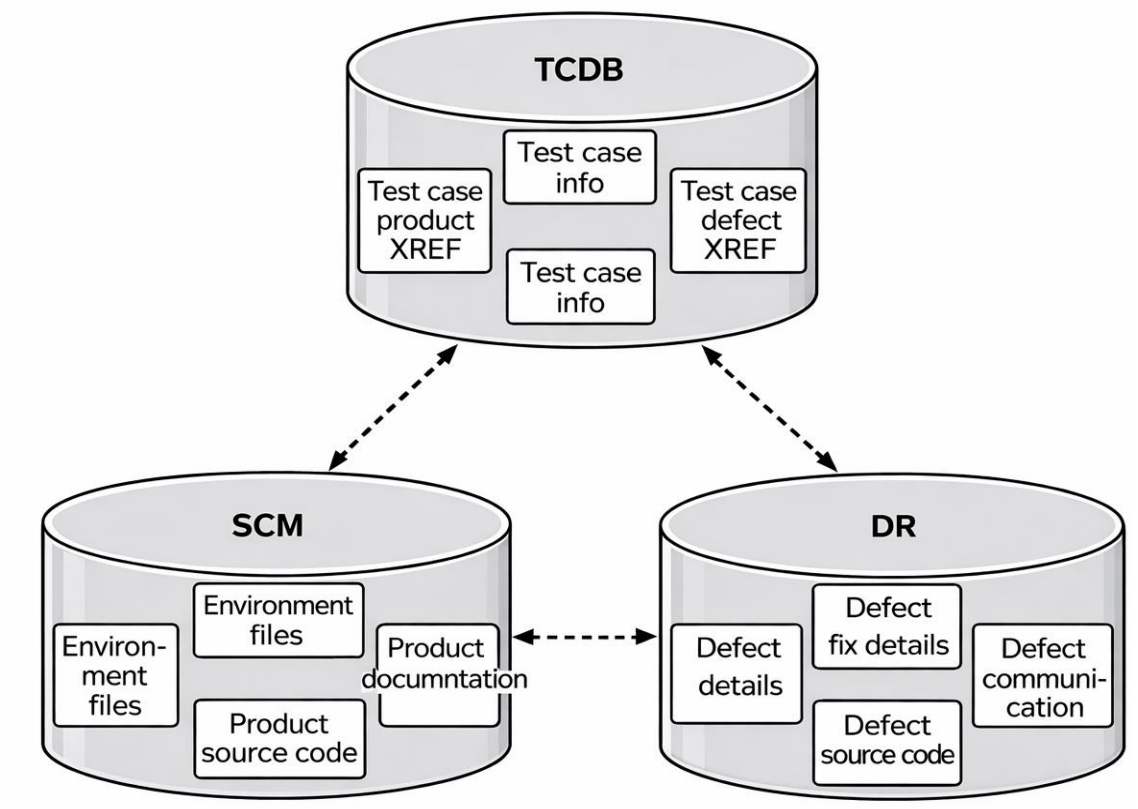

Fig 3.3: टेस्ट इन्फ्रास्ट्रक्चर मॅनेजमेंट (Test Infrastructure Management)

हा डायग्राम संरचित टेस्टिंग एन्व्हायर्नमेंटमध्ये (Structured Testing Environment) तीन रिपॉझिटरीज (Repositories) कशा प्रकारे परस्पर संवाद (Interact) साधतात हे दर्शवतो.

1. टेस्ट केस डेटाबेस (Test Case Database – TCDB)

टेस्ट केस डेटाबेस (Test Case Database (TCDB) ही एक केंद्रीकृत रिपॉझिटरी (Centralized Repository) आहे, ज्यामध्ये संस्थेतील सर्व टेस्ट केसेसशी संबंधित माहिती साठवली जाते. ही डेटाबेस टेस्ट केसेसचा इतिहास (History) तसेच प्रॉडक्ट्स आणि डिफेक्ट्ससोबतचे संबंध (Relationships) जतन करते.

एंटिटीज आणि ॲट्रिब्यूट्स (Entities and Attributes):

- **टेस्ट केस (Test Case):** आयडी (ID), वर्णन (Description), अपेक्षित निकाल (Expected Result), प्रायोरिटी (Priority).
- **टेस्ट केस–प्रॉडक्ट क्रॉस रेफरन्स (Test Case–Product Cross Reference):** टेस्ट केसेस आणि प्रॉडक्ट मॉड्यूल्स यांच्यातील मॅपिंग (Mapping)
- **टेस्ट केस रन हिस्टरी (Test Case Run History):** टेस्ट एक्झिक्युशनचे रेकॉर्ड तारीख (Date), स्थिती (Status), टेस्टर (Tester).
- **टेस्ट केस–डिफेक्ट क्रॉस रेफरन्स (Test Case–Defect Cross Reference):** फेल झालेल्या टेस्ट केसेस आणि नोंदवलेल्या डिफेक्ट्समधील लिंक्स (Links).

Sr. No. (क्रमांक)	Test Case Entity (टेस्ट केस घटक)	Purpose (उद्देश)	Attributes (घटक / वैशिष्ट्ये)
1	Test Case (टेस्ट केस)	Records all static information about tests टेस्टशी संबंधित सर्व स्थिर माहिती नोंदवते	Test Case ID, Name, Owner, Associated files टेस्ट केस आयडी, नाव, जबाबदार व्यक्ती, संबंधित फाईल्स
2	Test Case–Product Cross Reference (टेस्ट केस–प्रॉडक्ट क्रॉस रेफरन्स)	Maps test cases to product features, enables traceability टेस्ट केसेस आणि प्रॉडक्ट फिचर्स यांच्यात मॅपिंग करून ट्रेसेबिलिटी सुनिश्चित करते	Test Case ID, Module ID टेस्ट केस आयडी, मॉड्यूल आयडी
3	Test Case Run History (टेस्ट केस रन हिस्ट्री)	Captures execution history and outcomes (success/failure) टेस्ट केस एक्झिक्युशनचा इतिहास व निकाल (यश/अपयश) नोंदवते	Test Case ID, Run date, Time taken, Run status टेस्ट केस आयडी, रन तारीख, लागलेला वेळ, रन स्थिती
4	Test Case–Defect Cross Reference (टेस्ट केस–डिफेक्ट क्रॉस रेफरन्स)	Links defects with test cases for regression testing and root cause analysis रिग्रेशन टेस्टिंग व रूट कॉज ॲनालिसिससाठी डिफेक्ट्सना टेस्ट केसेसशी जोडते	Test Case ID, Defect reference टेस्ट केस आयडी, डिफेक्ट रेफरन्स

2. डिफेक्ट रिपॉझिटरी (Defect Repository – DR)

डिफेक्ट रिपॉझिटरी Defect Repository (DR) ही एक डेटाबेस आहे जी प्रॉडक्टशी संबंधित सर्व डिफेक्ट्स (Defects) ची माहिती साठवते. यामुळे बग स्टेटस, दुरुस्त्या (Fixes) आणि टेस्टर्स व डेव्हलपर्समधील संवाद (Communication) यांचे योग्य ट्रॅकिंग सुनिश्चित होते.

साठवली जाणारी माहिती (Information Stored):

- **डिफेक्ट तपशील (Defect Details):** आयडी(ID), सिव्हेरिटी (Severity), प्रायोरिटी (Priority), वर्णन (Description).
- **डिफेक्ट टेस्ट तपशील (Defect Test Details):** कोणत्या टेस्ट केस आणि कोणत्या एन्व्हायर्नमेंटमध्ये डिफेक्ट आढळला याची माहिती.
- **फिक्स तपशील (Fix Details):** रिझोल्यूशन, पॅच किंवा कोड बदल (Code Changes) यांची माहिती
- **संवाद (Communication):** टेस्टर्स आणि डेव्हलपर्स यांच्यातील चर्चेचे लॉग्स (Discussion Logs)

3. सॉफ्टवेअर कॉन्फिगरेशन मॅनेजमेंट रिपॉझिटरी (Software Configuration Management – SCM Repository)

सॉफ्टवेअर कॉन्फिगरेशन मॅनेजमेंट (SCM) ही टेस्ट आर्टिफॅक्ट्स (Test Artifacts) जसे की टेस्ट केसेस, टेस्ट स्क्रिप्ट्स आणि संबंधित फाइल्समधील सर्व बदल नियंत्रित (Control) आणि संघटित (Organize) करण्याची प्रक्रिया आहे. यामुळे सर्व टेस्ट-संबंधित दस्तऐवजांचे चेंज कंट्रोल (Change Control) आणि व्हर्जन कंट्रोल (Version Control) सुनिश्चित होते.

मुख्य कार्ये (Key Functions):

- टेस्ट फाइल्समधील बदलांसाठी मंजुरी (Approval) आवश्यक असते – नियंत्रित प्रक्रिया (Controlled Process).
- अनेक टेस्टर्स एकाच वेळी काम करत असताना फाइल्स ओव्हरराइट होणे किंवा हरवणे (Loss) टाळले जाते.
- फाइल्सच्या वेगवेगळ्या आवृत्त्या (Distinct Versions) जतन केल्या जातात, ज्या कोणत्याही वेळी पुन्हा तयार (Re-creatable) करता येतात.

- टेस्टर्सना फक्त सर्वात नवीन आवृत्ती (Most Recent Version) असलेल्या फाइल्सचा ॲक्सेस दिला जातो.

3.3.2 टेस्ट पीपल मॅनेजमेंट (Test People Management)

टेस्ट पीपल मॅनेजमेंट (Test People Management) म्हणजे सॉफ्टवेअर टेस्टिंगमध्ये सहभागी असलेल्या लोकांचे व्यवस्थापन (Managing), संघटन (Organizing) आणि प्रेरणा देणे (Motivating) ही एक प्रणालीबद्ध (Systematic) प्रक्रिया आहे. यामुळे योग्य वेळेत (Timeline) आणि अपेक्षित गुणवत्ता मानकांनुसार (Quality Standards) टेस्टिंगची उद्दिष्टे साध्य करण्यासाठी योग्य लोकांना योग्य भूमिका, जबाबदाऱ्या आणि संसाधने दिली जातात. सॉफ्टवेअर टेस्टिंग ही लोकांवर आधारित क्रिया (People-Driven Activity) असल्यामुळे टेस्ट प्रक्रियेची प्रभावीता मोठ्या प्रमाणावर टीम सदस्यांच्या कौशल्यांवर, समन्वयावर (Coordination) आणि सहकार्यावर (Collaboration) अवलंबून असते. टेस्ट पीपल मॅनेजमेंट मध्ये कौशल्यांनुसार (Competencies) टास्क्स वाटप करणे, स्पष्ट संवाद (Clear Communication) राखणे, संघर्ष निराकरण (Conflict Resolution) करणे आणि उच्च कार्यक्षमता (High Performance) साध्य करण्यासाठी टीमला प्रेरित करणे यावर भर दिला जातो. या प्रक्रियेत पुढील प्रमाणे विविध व्यवस्थापकीय जबाबदाऱ्या समाविष्ट असतात: भूमिका वाटप (Role Allocation), संवाद व्यवस्थापन (Communication Management), कौशल्य विकास (Skill Development), प्रेरणा (Motivation), परफॉर्मन्स मॉनिटरिंग (Performance Monitoring) आणि संघर्ष निराकरण (Conflict Resolution) टेस्ट मॅनेजर (Test Manager) टीममध्ये तांत्रिक कौशल्ये (Technical Expertise), डोमेन नॉलेज (Domain Knowledge) आणि आंतरवैयक्तिक कौशल्ये (Interpersonal Skills) यांचा योग्य समतोल राखण्यात महत्त्वाची भूमिका बजावतो. योग्य प्रकारे व्यवस्थापित केलेल्या टेस्टिंग टीम्समुळे डिफेक्ट लिकेज (Defect Leakage) कमी होते, वेळेत डिलिव्हरी (Timely Delivery) साध्य होते आणि अंतिम प्रॉडक्टची गुणवत्ता (Final Product Quality) सुधारते. आधुनिक टेस्ट मॅनेजमेंट ॲप्रोचेस (Test Management Approaches) मध्ये सतत शिक्षण (Continuous Learning), क्रॉस-फंक्शनल कोलॅबोरेशन (Cross-Functional Collaboration) आणि ॲजाइल टीम स्ट्रक्चर्स (Agile Team Structures) यावर भर दिला जातो, ज्यामुळे उत्पादकता (Productivity) आणि अनुकूलता (Adaptability) वाढते.

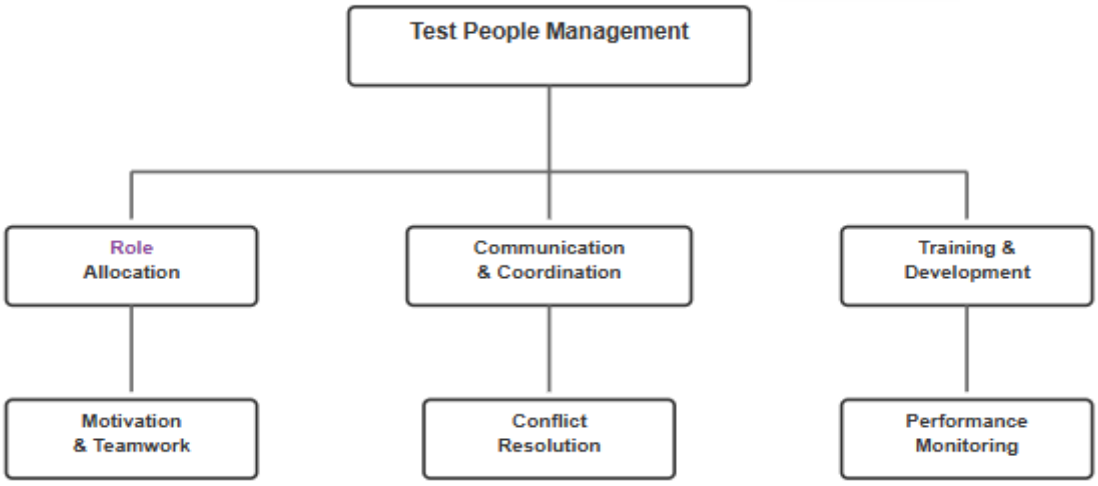

Fig 3.4: टेस्ट पीपल मॅनेजमेंट (Test People Management)

उदाहरण (Example)

परिस्थिती (Scenario): एक हेल्थकेअर मॅनेजमेंट सिस्टिम (Healthcare Management System) मध्ये अपॉइंटमेंट शेड्युलिंग (Appointment Scheduling) आणि बिलिंग (Billing) फीचर्सचे टेस्टिंग चालू आहे.

प्रक्रिया (Process):

- टेस्ट मॅनेजर (Test Manager) अपॉइंटमेंट वर्कफ्लोजसाठी फंक्शनल टेस्टर्स (Functional Testers), बिलिंग रिग्रेशन टेस्ट्ससाठी ऑटोमेशन टेस्टर्स (Automation Testers) आणि पेशंट डेटा व्हॅलिडेशनसाठी सिक्युरिटी टेस्टर्स (Security Testers) नियुक्त करतो.
- दररोजच्या डेली स्टँड-अप मिटिंग्स (Daily Stand-up Meetings) मुळे प्रभावी संवाद (Effective Communication) सुनिश्चित होतो.

- हेल्थकेअर नियमांचे (Healthcare Regulations) पालन (Compliance) मजबूत करण्यासाठी नवीन सिक्युरिटी टेस्टिंग टूल्स (Security Testing Tools) वर प्रशिक्षण दिले जाते.
- तीव्र रिलीज सायकल्स (Release Cycles) दरम्यान टेस्टर्सला प्रेरित ठेवण्यासाठी ओळख (Recognition) आणि लहान बक्षिसे (Small Rewards) दिली जातात.

निरीक्षण (Observation):

प्रभावी टेस्ट पीपल मॅनेजमेंट (Test People Management) मुळे टीमने अनेक सिक्युरिटी व्हल्नरेबिलिटीज (Security Vulnerabilities) लवकर ओळखल्या, रिलीज डेडलाईन्स (Release Deadlines) वेळेत पूर्ण केल्या आणि कामाच्या ताणातही (Workload Pressure) उच्च पातळीचे सहकार्य (High Collaboration) कायम ठेवले.

3.4 टेस्ट प्रोसेस (Test Process):

टेस्ट प्रोसेस (Test Process) म्हणजे सॉफ्टवेअर प्रॉडक्टची गुणवत्ता (Quality) सुनिश्चित करण्यासाठी करण्यात येणाऱ्या संरचित क्रियाकलापांचा (Structured Activities) संच होय. हा प्रोसेस टेस्टिंगसाठी आवश्यक असलेल्या टास्क्सचा क्रम (Sequence), पद्धती (Methods) आणि दस्तऐवजीकरण (Documentation) स्पष्टपणे परिभाषित करतो. प्रभावी सॉफ्टवेअर टेस्टिंगसाठी टेस्ट प्रोसेस(Test Process) अत्यंत महत्त्वाचा असतो कारण तो टेस्टिंग क्रियाकलापांवर स्पष्टता (Clarity), सुसंगतता (Consistency) आणि नियंत्रण (Control) प्रदान करतो.

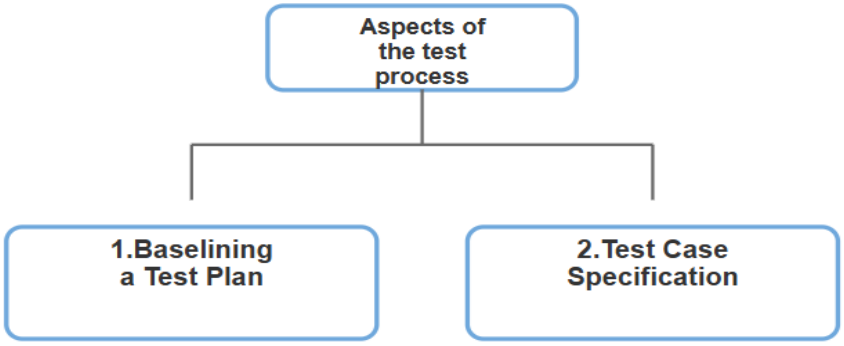

Fig 3.5: टेस्ट प्रोसेसचे पैलू (Aspects of Test Process)

3.4.1. टेस्ट प्लॅन बेसलाईन करणे (Baselining a Test Plan)

टेस्ट प्लॅन बेसलाईन करणे (Baselining a Test Plan) म्हणजे पुनरावलोकन (Reviews) आणि मंजुरी (Approvals) झाल्यानंतर टेस्ट प्लॅन दस्तऐवज अंतिम (Finalize) करून "फ्रीज" (Freeze) करणे, जेणेकरून तो संपूर्ण टेस्टिंग प्रक्रियेदरम्यान औपचारिक संदर्भ (Formal Reference) म्हणून वापरता येईल. बेसलाईन केलेला टेस्ट प्लॅन हा एक कंट्रोल्ड व्हर्जन (Controlled Version) असतो, ज्यामध्ये टेस्टिंग उद्दिष्टे, व्याप्ती (Scope), ॲप्रोच (Approach), संसाधने (Resources), वेळापत्रक (Schedule) आणि रिस्क्स (Risks) स्पष्टपणे नमूद केलेले असतात. एकदा टेस्ट प्लॅन बेसलाईन झाल्यानंतर, तो अधिकृत संदर्भ बिंदू (Official Reference Point) बनतो आणि त्यामधील कोणताही बदल फॉर्मल चेंज मॅनेजमेंट अप्रुव्हल (Formal Change Management Approval) शिवाय करता येत नाही. यामुळे स्कोप क्रीप (Scope Creep) टाळता येतो, सुसंगत टेस्ट एक्झिक्युशन (Consistent Execution) सुनिश्चित होते आणि टेस्टिंग क्रियाकलापांची ट्रेसबिलिटी (Traceability) साध्य होते.

उदाहरण (Example):

परिस्थिती (Scenario): UPI ट्रान्झॅक्शन्ससाठी (UPI Transactions) एका बँकिंग ॲप्लिकेशनचा टेस्ट प्लॅन तयार करण्यात येतो.

प्रक्रिया (Process):

QA टीम टेस्ट प्लॅन ड्राफ्ट करते → डेव्हलपर्स, मॅनेजर्स आणि स्टेकहोल्डर्स / क्लायंट्स (Stakeholders / Clients) सोबत रिव्ह्यू करते → आवश्यक बदल समाविष्ट करते → मंजुरी घेते → आणि त्याला बेसलाईन प्लॅन म्हणून फ्रीज करते.

निरीक्षण (Observation):

टेस्ट एक्झिक्युशन दरम्यान सर्व टीम मेंबर्स स्पष्टतेसाठी (Clarity) बेसलाईन केलेल्या प्लॅनचे पालन करतात. कोणतीही नवीन गरज (New Requirement) कंट्रोल्ड चेंज रिक्वेस्ट्स (Controlled Change Requests) द्वारे हाताळली जाते.

3.4.2 टेस्ट केस स्पेसिफिकेशन (Test Case Specification)

टेस्ट केस स्पेसिफिकेशन (Test Case Specification) म्हणजे एखाद्या विशिष्ट सॉफ्टवेअर फीचरच्या टेस्टिंगसाठी लागणाऱ्या टेस्ट केसेसचे सविस्तर वर्णन होय. यामध्ये इनपुट्स (Inputs), एक्झिक्युशन कंडीशन्स (Execution Conditions), प्रक्रिया (Procedures) आणि अपेक्षित निकाल (Expected Results) यांचा समावेश असतो. टेस्ट केस स्पेसिफिकेशनमुळे प्रत्येक फंक्शनॅलिटीचे टेस्टिंग प्रणालीबद्ध (Systematic) पद्धतीने होते.

योग्य प्रकारे लिहिलेल्या टेस्ट केसमध्ये खालील घटक स्पष्टपणे नमूद केलेले असतात:

- टेस्ट केस आयडी (Test Case ID)
- टेस्ट डिस्क्रिप्शन (Test Description)
- प्रीकंडिशन्स (Preconditions)
- इनपुट डेटा (Input Data)
- स्टेप्स (Steps)
- अपेक्षित आउटपुट (Expected Output)
- पास / फेल निकष (Pass / Fail Criteria)

या स्टँडर्डायझेशनमुळे (Standardization) टेस्ट कव्हरेज (Coverage) सुधारते, अस्पष्टता (Ambiguity) कमी होते आणि मोजता येण्याजोगे व्हेरिफिकेशन पॉइंट्स (Measurable Verification Points) मिळतात. साधारणपणे टेस्ट केस डेटाबेस (Test Case Database / TCDB) मध्ये टेस्ट केसेस साठविल्या जातात आणि रिक्वायरमेंट्स (Requirements) सोबत लिंक केल्या जातात, ज्यामुळे ट्रेसबिलिटी (Traceability) साध्य होते. ऑटोमेटेड टेस्ट केस स्पेसिफिकेशन (Automated Test Case Specification) मुळे रीयुजेबिलिटी (Reusability) आणि रिग्रेशन टेस्टिंग (Regression Testing) ची कार्यक्षमता वाढते.

उदाहरण (Example):

परिस्थिती (Scenario): एका E-commerce Website मध्ये "Add to Cart" फंक्शनसाठी टेस्ट केस स्पेसिफाय करण्यात येतो.

प्रक्रिया (Process):

- टेस्ट केस आयडी (Test Case ID): TC001
- प्रीकंडिशन (Precondition): युजर लॉग-इन असणे आवश्यक
- इनपुट(Input): आयटम सिलेक्ट करणे → "Add to Cart" क्लिक करणे
- अपेक्षित आउटपुट (Expected Output): आयटम योग्य क्वांटिटी आणि प्राइससह कार्टमध्ये दिसणे

निरीक्षण (Observation):

टेस्ट एक्झिक्युशनद्वारे ही फंक्शन योग्य प्रकारे कार्य करते का हे पडताळले जाते; कोणताही फरक (Deviation) आढळल्यास तो डिफेक्ट (Defect) म्हणून लॉग केला जातो.

3.5 टेस्ट रिपोर्टिंग (Test Reporting)

टेस्ट रिपोर्टिंग (Test Reporting) म्हणजे टेस्टिंग क्रियाकलापांचे निकाल दस्तऐवजीकरण (Documenting) करणे आणि ते संबंधित पक्षांपर्यंत (Communicating) पोहोचवणे ही प्रक्रिया आहे. यामध्ये टेस्ट केसेस एक्झिक्यूट करणे आणि रिपोर्ट्स तयार करणे यांचा समावेश होतो, ज्याद्वारे टेस्टिंगची प्रगती (Progress), आढळलेले डिफेक्ट्स (Defects), मिळालेले कव्हरेज (Coverage) आणि एकूण प्रॉडक्ट गुणवत्ता (Overall Product Quality) दर्शवली जाते.

3.5.1 टेस्ट केसेस एक्झिक्यूट करणे (Executing Test Cases)

टेस्ट केस एक्झिक्युशन (Test Case Execution) म्हणजे विकसित सॉफ्टवेअरवर ठरवलेल्या टेस्ट एन्व्हायर्नमेंट (Test Environment) मध्ये टेस्ट केसेस चालवण्याची प्रक्रिया होय, ज्याद्वारे ॲप्लिकेशन अपेक्षेप्रमाणे वागते का हे पडताळले जाते. टेस्ट केसेस डिझाइन आणि मंजूर (Approved) झाल्यानंतर त्या मॅन्युअली (Manually) किंवा ऑटोमेटेड टूल्स (Automated Tools) वापरून एक्झिक्यूट केल्या जातात. प्रत्येक एक्झिक्युशनचा निकाल एक्स्पेक्टेड आउटकम (Expected Outcome) सोबत तुलना केला जातो, ज्यावरून टेस्ट पास / फेल (Pass / Fail) ठरते. कोणताही फरक (Deviation) आढळल्यास तो डिफेक्ट रिपॉझिटरी (Defect Repository) मध्ये नोंदवला जातो. योग्य एक्झिक्युशनमुळे

रिक्वायरमेंट कव्हरेज (Requirement Coverage), लवकर डिफेक्ट शोध (Early Defect Detection) आणि गुणवत्ता-संबंधित निर्णय (Quality Decisions) घेण्यास मदत होते.

उदाहरण (Example):

परिस्थिती (Scenario): एका Mobile Banking App मध्ये "Funds Transfer" फीचरचे टेस्टिंग केले जाते.

प्रक्रिया (Process):

QA वैध अकाउंट नंबर आणि रक्कम वापरून टेस्ट केस एक्झिक्यूट करतो → सिस्टिम ट्रान्सफर प्रोसेस करते → कन्फर्मेशन मेसेज तपासला जातो.

निरीक्षण (Observation):

मेसेज योग्यरित्या दिसल्यास टेस्ट पास होते; अन्यथा ती समस्या डिफेक्ट (Defect) म्हणून लॉग केली जाते, जेणेकरून डेव्हलपर्स दुरुस्ती करू शकतील.

3.5.2 टेस्ट समरी रिपोर्ट तयार करणे (Preparing Test Summary Report)

टेस्ट समरी रिपोर्ट तयार करणे (Test Summary Report (TSR)) हा टेस्ट सायकलच्या शेवटी तयार होणारा अंतिम दस्तऐवज आहे, जो टेस्टिंगचे एकूण निकाल सादर करतो. यात टेस्टिंग क्रियाकलापांचा सारांश, विचलने (Deviations), निकाल (Results), डिफेक्ट्स आणि प्रॉडक्ट रिलीजसाठी शिफारसी (Recommendations) समाविष्ट असतात. टेस्ट लाईफ सायकलमधील (Test Life Cycle) हा शेवटचा टप्पा असून, प्रॉडक्ट रिलीजसाठी योग्य आहे का याबाबत शिफारस केली जाते. टेस्ट समरी रिपोर्ट मुळे स्टेकहोल्डर्स / क्लायंट्स (Stakeholders / Clients), प्रोजेक्ट मॅनेजर्स (Project Managers) आणि ग्राहकांना सॉफ्टवेअरची एकूण गुणवत्ता समजते. यात किती प्रमाणात टेस्टिंग झाले, कोणते डिफेक्ट्स आढळले व निराकरण झाले, तसेच एक्झिट क्रायटेरिया (Exit Criteria) पूर्ण झाले आहेत का हे स्पष्ट केले जाते.

टेस्ट समरी रिपोर्टचे प्रकार (Types of Test Summary Reports):

1. फेज-वाइज टेस्ट समरी रिपोर्ट (Phase-wise Test Summary Report): प्रत्येक टेस्टिंग फेजच्या शेवटी तयार होतो (उदा. युनिट टेस्टिंग, इंटिग्रेशन टेस्टिंग, सिस्टिम टेस्टिंग)
2. अंतिम टेस्ट समरी रिपोर्ट (Final Test Summary Report): सर्व टेस्टिंग फेजेसचे एकत्रित (Consolidated) निकाल आणि प्रॉडक्ट रिलीजसाठी अंतिम शिफारसी समाविष्ट करणारा रिपोर्ट.

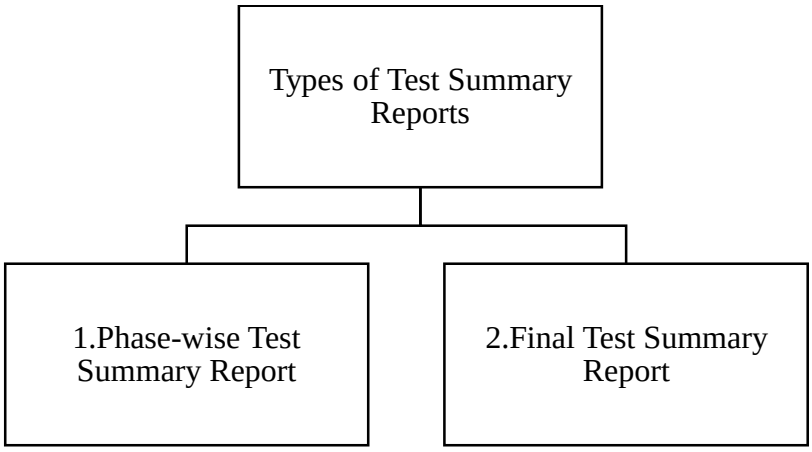

Fig 3.6: टेस्ट समरी रिपोर्टचे प्रकार (Types of TSR)

1. फेज-वाइज टेस्ट समरी रिपोर्ट (Phase-wise Test Summary Report)

फेज-वाइज टेस्ट समरी रिपोर्ट (Phase-wise Test Summary Report) हा प्रत्येक टेस्टिंग फेजच्या शेवटी (उदा. युनिट टेस्टिंग, इंटिग्रेशन टेस्टिंग, सिस्टिम टेस्टिंग किंवा ॲक्सेप्टन्स टेस्टिंग) तयार केला जाणारा दस्तऐवज आहे. हा रिपोर्ट त्या विशिष्ट फेजमधील टेस्टिंग क्रियाकलाप, निकाल आणि आढळलेले डिफेक्ट्स यांचा सारांश सादर करतो. सॉफ्टवेअर टेस्टिंग सामान्यतः अनेक फेजेसमध्ये केली जाते, जसे की युनिट टेस्टिंग (Unit Testing), इंटिग्रेशन टेस्टिंग (Integration Testing), सिस्टम टेस्टिंग (System Testing) आणि यूजर ॲक्सेप्टन्स टेस्टिंग (User Acceptance Testing (UAT). प्रत्येक फेज पूर्ण झाल्यानंतर, त्या फेजची उद्दिष्टे साध्य झाली आहेत का हे तपासण्यासाठी फेज-वाइज टेस्ट समरी रिपोर्ट तयार केला जातो.

या रिपोर्टमध्ये खालील बाबी ठळकपणे नमूद केल्या जातात (This Report Highlights):

- त्या फेजमध्ये एक्झिक्यूट केलेल्या टेस्ट केसेस
- पास / फेल आकडेवारी (Pass/Fail Statistics)
- आढळलेले डिफेक्ट्स आणि त्यांची सिव्हेरिटी (Severity)
- नियोजित प्रक्रियेपासून झालेले बदल / फरक (Variances)
- त्या फेजचे एक्झिट क्रायटेरिया (Exit Criteria) पूर्ण झाले आहेत का याचे मूल्यमापन

हा रिपोर्ट प्रोजेक्ट मॅनेजर्स (Project Managers) आणि स्टेकहोल्डर्स / क्लायंट्स (Stakeholders / Clients) यांना प्रगती ट्रॅक करण्यास, लवकर रिस्क्स ओळखण्यास आणि पुढील टेस्टिंग फेजमध्ये जाण्याचा निर्णय घेण्यास मदत करतो.

फेज-वाइज टेस्ट समरी रिपोर्टची रचना (Phase-wise Test Summary Report Structure):

1. रिपोर्ट आयडेंटिफायर (Report Identifier)
2. फेज तपशील (Phase Details) – युनिट / इंटिग्रेशन / सिस्टिम / यूएटी (Unit / Integration / System / UAT)
3. टेस्टिंगचा सारांश (Summary of Testing)
    - नियोजित विरुद्ध एक्झिक्यूट केलेल्या टेस्ट केसेस
    - पास / फेल / ब्लॉक्ड (Pass / Fail / Blocked) केसेस
4. नोंदवलेले डिफेक्ट्स (Defects Reported)
    - डिफेक्ट्सची संख्या (Critical, Major, Minor)
    - फिक्स स्टेटस (Fix Status)
5. व्हेरिअन्सेस (Variances) – प्लॅनपासून झालेले बदल (असल्यास)
6. मूल्यमापन (Assessment) – फेजच्या निकालाची गुणवत्ता, मिळालेले कव्हरेज
7. शिफारस (Recommendation) – पुढील फेजला जाणे / दुरुस्तीसाठी थांबवणे

उदाहरण – फेज-वाइज रिपोर्ट (Example – Phase-wise Report):

रिपोर्ट आयडेंटिफायर (Report Identifier): TSR_System_2025_02

फेज (Phase): सिस्टिम टेस्टिंग (System Testing)

टेस्टिंगचा सारांश (Summary of Testing):

- Planned (नियोजित): 120 टेस्ट केसेस
- Executed (एक्झिक्यूट): 118 टेस्ट केसेस
- Passed (पास): 100, Failed (फेल): 12, Blocked (ब्लॉक्ड): 6

नोंदवलेले डिफेक्ट्स (Defects Reported):

- 3 क्रिटिकल, 5 मेजर, 7 मायनर (3 Critical, 5 Major, 7 Minor)
- फिक्सेस चालू आहेत, क्रिटिकल डिफेक्ट्स रिटेस्ट अंतर्गत आहेत.

व्हेरिअन्सेस (Variances): टेस्ट एन्व्हायर्नमेंट क्रॅशमुळे टेस्ट एक्झिक्युशनमध्ये विलंब

मूल्यमापन (Assessment):

- 85% पास रेट, एक्झिट क्रायटेरिया अंशतः पूर्ण
- कव्हरेज (Coverage): 92% रिक्वायरमेंट्स टेस्ट झाल्या

शिफारस (Recommendation): क्रिटिकल डिफेक्ट्स क्लोज झाल्यानंतरच यूएटी (UAT) कडे पुढे जावे.

2. अंतिम टेस्ट समरी रिपोर्ट (Final Test Summary Report)

अंतिम टेस्ट समरी रिपोर्ट (Final Test Summary Report) हा सर्व टेस्टिंग फेजेस पूर्ण झाल्यानंतर तयार केला जाणारा सविस्तर (Comprehensive) दस्तऐवज आहे. यामध्ये युनिट टेस्टिंग, इंटिग्रेशन टेस्टिंग, सिस्टिम टेस्टिंग आणि ॲक्सेप्टन्स टेस्टिंग (Unit Testing, Integration Testing, System Testing and Acceptance Testing) या सर्व टप्प्यांतील निकाल एकत्रित (Consolidated) करून प्रॉडक्टच्या एकूण गुणवत्तेचे (Overall Product Quality) मूल्यमापन आणि रिलीजसाठीची तयारी (Release Readiness) याबाबत शिफारस केली जाते. फेज-वाइज रिपोर्ट्स जिथे प्रत्येक टेस्टिंग टप्प्याची स्वतंत्र माहिती देतात, तिथे अंतिम टेस्ट समरी रिपोर्ट सर्व टेस्टिंग क्रियाकलापांचे एकत्रित चित्र (Holistic View)

सादर करतो. यात टेस्टिंग उद्दिष्टे, व्याप्ती (Scope), एक्झिक्यूट केलेल्या टेस्ट केसेस, पास/फेल आकडेवारी, डिफेक्ट स्टेटस, रिस्क ॲनालिसिस (Risk Analysis) आणि रिलीज शिफारसी (Release Recommendations) समाविष्ट असतात. हा रिपोर्ट सहसा टेस्ट लीड (Test Lead) किंवा टेस्ट मॅनेजर (Test Manager) तयार करतो आणि स्टेकहोल्डर्स / क्लायंट्स (Stakeholders / Clients), प्रोजेक्ट मॅनेजर्स, डेव्हलपर्स आणि क्लायंट्ससोबत शेअर केला जातो. यामुळे प्रॉडक्ट रिलीज करण्याचा निर्णय पूर्ण व्हिजिबिलिटी सह घेतला जातो गुणवत्ता, कव्हरेज आणि रिस्क्स यांचा विचार करून. हा रिपोर्ट अत्यंत महत्त्वाचा असतो कारण तो एक मुख्य प्रश्न सोडवतो: "सॉफ्टवेअर प्रॉडक्ट रिलीजसाठी योग्य आहे का?"

IEEE फॉरमॅट – टेस्ट समरी रिपोर्ट (IEEE Format – Test Summary Report)

1. **टेस्ट समरी रिपोर्ट आयडेंटिफायर (Test Summary Report Identifier):** रिपोर्टसाठी दिलेला एक अद्वितीय ओळख क्रमांक (Unique Identifier).
   उदाहरण (Example): TSR_BankingApp_2025_01
2. **सारांश (Summary):** केलेल्या टेस्टिंग क्रियाकलापांचा एकूण आढावा.
   - नियोजित आणि एक्झिक्यूट केलेल्या टेस्ट केसेसची संख्या
   - पास्ड, फेल्ड, ब्लॉक्ड आणि स्किप्ड केसेसची संख्या
   - एकूण टेस्ट कव्हरेज (Overall Test Coverage)
3. **व्हेरिअन्सेस (Variances):** व्हेरिअन्सेस मूळ टेस्ट प्लॅन (Test Plan) किंवा टेस्ट प्रोसीजर्सपासून झालेले कोणतेही बदल. शेड्युल व्हेरिअन्सेस (Schedule Variances) रिसोर्स / एन्व्हायर्नमेंट व्हेरिअन्सेस (Resource / Environment Variances) प्रोसेस डिव्हिएशन्स (Process Deviations)
4. **सर्वसमावेशक मूल्यमापन (Comprehensive Assessment):** टेस्टिंगच्या आधारे सॉफ्टवेअर गुणवत्तेचे मूल्यमापन.
   - आढळलेले डिफेक्ट्स – क्रिटिकल, मेजर, मायनर (Critical, Major, Minor)
   - सिव्हेरिटी आणि सिस्टिम फंक्शनॅलिटीवरील प्रभाव (Impact)
   - प्रॉडक्ट स्टॅबिलिटी (Product Stability) आणि परफॉर्मन्स
5. **निकालांचा सारांश (Summary of Results):** टेस्टिंग क्रियाकलापांचे सविस्तर निकाल.
   - एक्झिक्यूट केलेल्या टेस्ट केसेस: Total, Passed, Failed, Blocked
   - डिफेक्ट मेट्रिक्स: ओपन, क्लोज्ड, डिफर्ड (Open, Closed, Deferred)
   - प्रमुख अपयशांचे (Major Failures) रूट कॉज ॲनालिसिस (Root Cause Analysis)
6. **मूल्यमापन (Evaluation):** टेस्ट उद्दिष्टे आणि एक्झिट क्रायटेरिया (Exit Criteria) पूर्ण झाली आहेत का याचे मूल्यमापन.
   - रिक्वायरमेंट्सच्या तुलनेत कव्हरेज
   - सॉफ्टवेअरवरील विश्वास पातळी (Confidence Level)
   - सुधारणा आवश्यक असलेले भाग (Areas Needing Improvement)
7. **क्रियाकलापांचा सारांश (Summary of Activities):** टेस्टिंग दरम्यान केलेल्या प्रमुख क्रियाकलापांची थोडक्यात माहिती.
   - टेस्ट केस डिझाइन
   - टेस्ट एन्व्हायर्नमेंट सेटअप
   - टेस्ट एक्झिक्युशन
   - डिफेक्ट लॉगिंग आणि रिटेस्टिंग
8. **मंजुरी (Approval):** टेस्ट लीड / QA मॅनेजर / (Test Lead / QA Manager) आणि प्रोजेक्ट स्टेकहोल्डर्स (Project Stakeholders) यांची साइन-ऑफ (Sign-off)
   - अंतिम शिफारस (Final Recommendation):
     - रिलीजसाठी योग्य (Fit for Release)
     - रिलीजसाठी अयोग्य (Not Fit for Release)

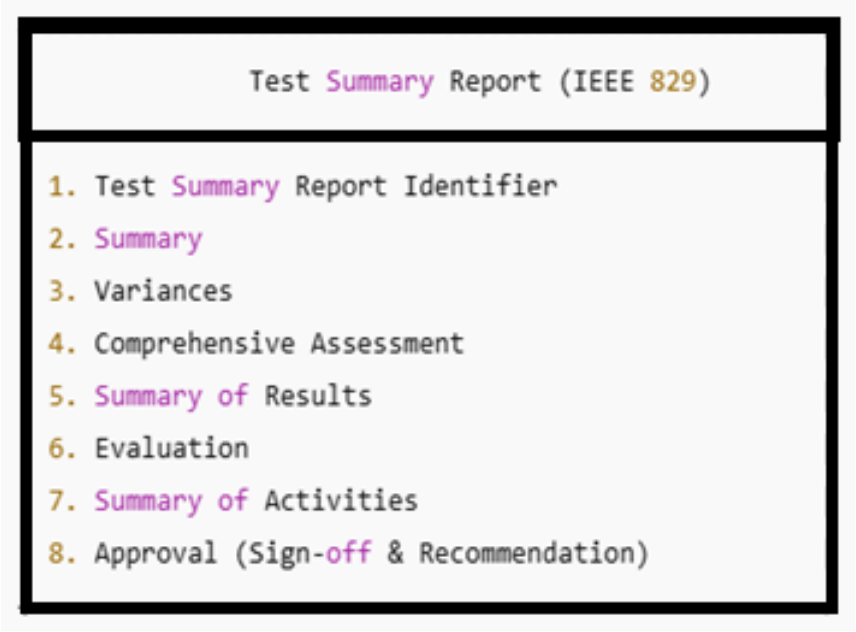

Fig 3.7: टेस्ट समरी रिपोर्ट (Test Summary Report (IEEE 829))

उदाहरण – IEEE 829 टेस्ट समरी रिपोर्ट (Example – IEEE 829 TSR)

1. ओळख क्रमांक (Identifier): TSR_BankApp_2025_01
2. सारांश (Summary):
   - 200 टेस्ट केसेस एक्झिक्यूट
   - Passed (पास): 180
   - Failed (फेल): 15
   - Blocked (ब्लॉक्ड): 5
3. व्हेरिअन्सेस (Variances): सिक्युरिटी टेस्टिंग (Security Testing) टूल लायसन्स (Tool License) समस्येमुळे उशिरा झाले
4. मूल्यमापन (Assessment):
   - एकूण 15 डिफेक्ट्स आढळले – 2 क्रिटिकल, 5 मेजर, 8 मायनर (2 Critical, 5 Major, 8 Minor)
   - क्रिटिकल डिफेक्ट्स दुरुस्त (Fixed) करण्यात आले
5. निकाल (Results):
   - कव्हरेज (Coverage) प्राप्त: 95%
   - सर्व डिफेक्ट्स रिटेस्ट करून क्लोज (Closed) करण्यात आले
6. मूल्यमापन (Evaluation): एग्झिट क्रायटेरिया पूर्ण झाले. रिस्क लेव्हल स्वीकार्य आहे.
7. क्रियाकलाप (Activities):
   - टेस्ट केस डिझाइन
   - टेस्ट एक्झिक्युशन
   - डिफेक्ट लॉगिंग
   - रिग्रेशन टेस्टिंग (Regression Testing)
8. मंजुरी (Approval): QA लीड (QA Lead) आणि प्रोजेक्ट मॅनेजर (Project Manager) यांनी मंजुरी दिली शिफारस (Recommendation): रिलीजसाठी योग्य (Fit for Release)

References

1. Desikan, S., & Ramesh, G. (2016). *Software Testing: Principles and Practices*. Pearson India. ISBN: 9788177581218.
2. Limaye, M. G. (2012). *Software Testing: Principles, Techniques and Tools*. Tata McGraw Hill Education. ISBN: 9780070139909.
3. Chauhan, N. (2016). *Software Testing: Principles and Practices*. Oxford University Press. ISBN: 9780198061847.
4. Singh, Y. (2012). *Software Testing*. Cambridge University Press. ISBN: 9781107652781.
5. Infosys Springboard – *Software Testing Fundamentals*. Available at: https://infyspringboard.onwingspan.com
6. NPTEL – *Software Testing Course*. Available at: https://nptel.ac.in/courses/106101163
7. GeeksforGeeks – *Sanity vs Smoke Testing*. Available at: https://www.geeksforgeeks.org/difference-between-sanity-testing-and-smoke-testing/
8. Guru99 – *Smoke Testing vs Sanity Testing*. Available at: https://www.guru99.com/smoke-testing.html
9. W3Schools – *Software Testing Tutorials*. Available at: https://www.w3schools.in/software-testing/tutorials/

युनिट-4
डिफेक्ट मॅनेजमेंट
(Defect Management)

विषय निष्पत्ती (Course Outcome):

CO4: दिलेल्या ॲप्लिकेशन साठी डिफेक्ट रिपोर्ट तयार करा.

घटक निष्पत्ती (Theory Learning Outcome):

1. अंदाजित परिणामाच्या (estimated impact) आधारे डिफेक्ट्स चे वर्गीकरण करा.
2. दिलेल्या ॲप्लिकेशन साठी डिफेक्ट टेम्पलेट तयार करा.
3. दिलेल्या ॲप्लिकेशन वर डिफेक्ट मॅनेजमेंट प्रोसेस स्पष्ट करा.

4.1. डिफेक्ट क्लासिफिकेशन, डिफेक्ट मॅनेजमेंट प्रोसेस (Defect Classification, Defect Management Process)

4.1.1 डिफेक्ट क्लासिफिकेशन (Defect Classification)

डिफेक्ट (किंवा बग) म्हणजे सॉफ्टवेअरचे अपेक्षित वर्तन किंवा आवश्यकता (requirements) यांपासून झालेला कोणताही फरक (deviation) होय. सोप्या शब्दांत सांगायचे झाल्यास, जेव्हा एखादी सॉफ्टवेअर सिस्टीम अपेक्षेप्रमाणे कार्य करत नाही किंवा चुकीचे परिणाम देते, तेव्हा त्या सिस्टीममध्ये डिफेक्ट असल्याचे मानले जाते. सॉफ्टवेअर डेव्हलपमेंट लाईफ सायकल (SDLC) च्या विविध टप्प्यांमध्ये डिफेक्ट्स निर्माण होऊ शकतात आणि त्यांचे स्वरूप वेगवेगळे असू शकते. त्यामुळे डिफेक्ट्स चे व्यवस्थित वर्गीकरण करणे आवश्यक ठरते. डिफेक्ट क्लासिफिकेशन ही अशी प्रक्रिया आहे ज्यामध्ये डिफेक्ट्स ना त्यांच्या स्वरूप (nature), सिव्हेरिटी (severity), प्रायोरिटी (priority) किंवा उगम (origin) यांच्या आधारे विविध श्रेणींमध्ये विभागले जाते. डिफेक्ट्स चे वर्गीकरण केल्यामुळे सॉफ्टवेअर टीमला हे स्पष्टपणे समजते की डिफेक्ट्स कुठून निर्माण होत आहेत. रिक्वायरमेंट्स, डिझाइन, कोडिंग किंवा टेस्टिंग, ते सिस्टीमसाठी किती गंभीर (critical) आहेत आणि त्यांचे निराकरण किती लवकर करणे आवश्यक आहे. उदाहरणार्थ, एखादा डिफेक्ट ज्यामुळे सिस्टीम क्रॅश होते, तो हाय सिव्हेरिटी डिफेक्ट म्हणून वर्गीकृत केला जातो, तर युजर इंटरफेसवरील (user interface) किरकोळ स्पेलिंग चूक ही लो सिव्हेरिटी डिफेक्ट मानली जाते. डिफेक्ट क्लासिफिकेशन चा मुख्य उद्देश म्हणजे टेस्टर्स, डेव्हलपर्स आणि प्रोजेक्ट मॅनेजर्स यांना डिफेक्ट्स ची कारणे विश्लेषित (analyze) करण्यास मदत करणे, त्यांना योग्य प्रायोरिटी देऊन दुरुस्ती करणे आणि भविष्यात अशा प्रकारच्या समस्या टाळणे होय. यामुळे टीममधील संवाद (communication) सुधारतो, कारण सर्व सदस्यांना डिफेक्ट्स च्या प्रकारांची समान समज प्राप्त होते. तसेच, अनेक प्रोजेक्ट्स मध्ये डिफेक्ट्स चे ट्रॅकिंग आणि कॅटेगरायझेशन केल्यामुळे संस्था त्यांच्या डेव्हलपमेंट प्रोसेस मधील कमकुवत भाग (weak points) ओळखू शकतात (उदा. बहुतेक डिफेक्ट्स रीक्वायरमेंट्स गॅदरिंग (requirements gathering) दरम्यान निर्माण होतात). परिणामी, भविष्यातील सॉफ्टवेअर प्रॉडक्ट्स ची एकूण गुणवत्ता सुधारण्यास मदत होते.

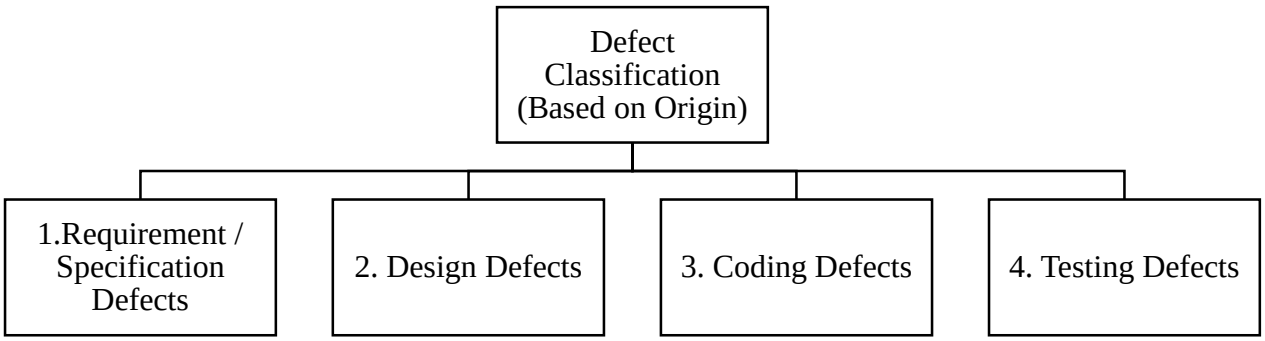

Fig 4.1: डिफेक्ट वर्गीकरण (Defect Classification)

उत्पत्तीवर आधारित डिफेक्ट वर्गीकरण (Defect Classification Based on Origin)

1. रिक्वायरमेंट / स्पेसिफिकेशन डिफेक्ट्स (Requirement / Specification Defects)

हे डिफेक्ट्स तेव्हा निर्माण होतात जेव्हा ग्राहकांच्या गरजा (Requirements) चुकीच्या प्रकारे समजल्या जातात, अस्पष्ट (Unclear) असतात, अपूर्ण (Incomplete) असतात किंवा योग्य प्रकारे अंमलात आणल्या जात नाहीत.

उदाहरण (Example)

परिस्थिती (Scenario):एका बँकिंग ॲप्लिकेशनच्या रिक्वायरमेंटमध्ये नमूद आहे:"System must support INR transactions."

पण प्रत्यक्षात ग्राहकाला मल्टी-करन्सी सपोर्ट (Multi-Currency Support)(INR, USD, EUR) हवा होता.

निरीक्षण (Observation):

• सिस्टिम फक्त INR स्वीकारते.

• आंतरराष्ट्रीय ग्राहक (International Customers) व्यवहार पूर्ण करू शकत नाहीत.

प्रक्रिया (Process):

• कारण (Cause): रिक्वायरमेंट गॅदरिंगमध्ये अंतर (Customer Gap / Producer Gap).

• उपाय (Fix):

- कस्टमर वर्कशॉप्स (Customer Workshops) आयोजित करणे
- रिक्वायरमेंट व्हॅलिडेशन (Requirement Validation)
- प्रोटोटायपिंग (Prototyping) वापरणे

2. डिझाइन डिफेक्ट्स (Design Defects)

हे डिफेक्ट्स चुकीच्या आर्किटेक्चरमुळे (Faulty Architecture), कॉम्पोनंट्समधील अयोग्य इंटरॅक्शन किंवा खराब सिस्टिम इंटिग्रेशन डिझाइनमुळे निर्माण होतात.

उदाहरण (Example)

रिस्थिती (Scenario):

एका ई-कॉमर्स वेबसाईट (E-commerce Website) मध्ये सर्व प्रॉडक्ट डिटेल्स (इमेजेस, डिस्क्रिप्शन्स, स्टॉक) एकाच टेबलमध्ये साठवले जातात.

निरीक्षण (Observation):

- सर्च / फिल्टर ऑपरेशनसाठी 15–20 सेकंद लागतात.
- डेटा वाढल्यावर सिस्टिम स्लो होते.

प्रक्रिया (Process):

कारण (Cause): खराब सिस्टिम डिझाइन (डेटाबेस स्कीमा ऑप्टिमाईज्ड नाही).

उपाय (Fix):

- डेटाबेस नॉर्मलायझेशन (Database Normalization)
- इंडेक्सिंग (Indexing)
- कॅशिंग (Caching)
- कोडिंगपूर्वी डिझाइन रिव्ह्यूज (Design Reviews) आणि वॉकथ्रूज (Walkthroughs) करणे.

3. कोडिंग डिफेक्ट्स (Coding Defects)

हे डिफेक्ट्स अंमलबजावणीतील चुका (Implementation Errors) जसे की सिंटॅक्स, लॉजिक, डेटा हँडलिंग किंवा एरर चेक्स न केल्यामुळे निर्माण होतात.

उदाहरण (Example)

परिस्थिती (Scenario): लॉगिन मॉड्यूल मधील कोड:

```
if (password = "12345") {
  login_success = true;
}
```

डेव्हलपरने == ऐवजी = वापरले आहे.

निरीक्षण (Observation):

- कोणतेही इनपुट दिल्यास लॉगिन यशस्वी होते.
- ॲप्लिकेशनची सिक्युरिटी (Application Security) धोक्यात येते.

प्रक्रिया (Process):

कारण (Cause): कोडिंग मिस्टेक आणि रिव्ह्यूचा अभाव.

उपाय (Fix):

- कोड रिव्ह्यूज (Code Reviews)
- युनिट टेस्टिंग (Unit Testing)
- स्टॅटिक कोड ॲनालिसिस टूल्स (Static Code Analysis Tools) उदा. सोनारक्यूब(SonarQube)
- कोडिंग स्टँडर्ड्स (Coding Standards) लागू करणे

4. टेस्टिंग डिफेक्ट्स (Testing Defects)

हे डिफेक्ट्स चुकीच्या, अपूर्ण किंवा गहाळ (Missing) टेस्ट केसेस किंवा टेस्ट प्रोसीजर्समुळे निर्माण होतात.

उदाहरण (Example)

परिस्थिती (Scenario): पासवर्ड नियम (Password Rule): किमान 8 कॅरेक्टर्स आवश्यक. टेस्टर 6, 7 आणि 10 कॅरेक्टर्ससाठी टेस्ट केसेस लिहितो, पण नेमके 8 कॅरेक्टर्ससाठी टेस्ट केस चुकतो.

निरीक्षण (Observation):

- बाउंडरी केस (Boundary Case) प्रॉडक्शनमध्ये निसटतो.
- काही वैध युजर्स (Valid Users) लॉग-इन करू शकत नाहीत.

प्रक्रिया (Process):

• कारण (Cause): योग्य टेस्ट डिझाइन टेक्निक वापरली नाही.

• उपाय (Fix):

- बाउंडरी व्हॅल्यू ॲनालिसिस (Boundary Value Analysis (BVA))
- इक्विव्हॅलन्स पार्टिशनिंग (Equivalence Partitioning)
- टेस्ट केसेसचे पीअर रिव्ह्यूज (Peer Reviews)

Table 4.1: डिफेक्ट क्लासिफिकेशन (Defect Classification)

Defect Type (डिफेक्ट प्रकार)	Definition (व्याख्या)	Scenario (परिस्थिती)	Observation (निरीक्षण)	Process (Cause & Fix) (कारण व उपाय)
Requirement Defect (आवश्यकता दोष)	Errors due to misunderstood or missing requirements चुकीच्या समजुतीमुळे किंवा अपूर्ण आवश्यकतांमुळे होणारे दोष	Banking app supports only INR instead of multiple currencies बँकिंग ॲप फक्त INR सपोर्ट करते, मल्टी-करन्सी नाही	International users can't transact आंतरराष्ट्रीय युजर्स व्यवहार करू शकत नाहीत	Improve requirement gathering & validation योग्य आवश्यकता संकलन व व्हॅलिडेशन करणे
Design Defect (डिझाईन दोष)	Errors in architecture or system design सिस्टीम आर्किटेक्चर किंवा डिझाईनमधील त्रुटी	Single DB table for products प्रॉडक्टसाठी एकच डेटाबेस टेबल वापरणे	Slow searches as data grows डेटा वाढल्यावर शोध प्रक्रिया मंद होते	Use normalization, design reviews नॉर्मलायझेशन वापरणे व डिझाईन रिव्ह्यू करणे

Coding Defect (कोडिंग दोष)	Errors in implementation (logic/syntax) कोड लिहिताना लॉजिक किंवा सिंटॅक्समध्ये चूक	Wrong operator (= instead of ==) in login लॉगिनमध्ये == ऐवजी = वापरणे	Invalid logins accepted अवैध लॉगिन्स स्वीकारली जातात	Code reviews, unit testing कोड रिव्ह्यू आणि युनिट टेस्टिंग
Testing Defect (टेस्टिंग दोष)	Errors in test cases or test data टेस्ट केसेस किंवा टेस्ट डेटामधील त्रुटी	Missing boundary case for password = 8 पासवर्ड 8 कॅरेक्टरचा बाउंडरी केस नाही	Valid users rejected वैध युजर्स नाकारले जातात	Use BVA, peer review of test cases BVA वापरणे व टेस्ट केसेसचे पीअर रिव्ह्यू

4.1.2 डिफेक्ट मॅनेजमेंट प्रोसेस (Defect Management Process)

डिफेक्ट मॅनेजमेंट प्रोसेस (Defect Management Process) म्हणजे सॉफ्टवेअर सिस्टिममधील डिफेक्ट्स ओळखणे (Identify), दस्तऐवजीकरण करणे (Document), विश्लेषण करणे (Analyze), प्राधान्य देणे (Prioritize), दुरुस्त करणे (Fix) आणि भविष्यात पुन्हा होऊ नयेत यासाठी प्रतिबंध (Prevent) करणे यासाठी वापरली जाणारी एक प्रणालीबद्ध (Systematic) पद्धत आहे. ही प्रक्रिया डिफेक्ट्स नियंत्रित (Controlled) आणि कार्यक्षम (Efficient) पद्धतीने हाताळली जात आहेत याची खात्री करते, ज्यामुळे रिस्क्स (Risks) कमी होतात आणि सॉफ्टवेअरची एकूण गुणवत्ता (Overall Software Quality) सुधारते. सॉफ्टवेअर टेस्टिंगमध्ये केवळ डिफेक्ट्स शोधणे पुरेसे नसते—त्यांचे योग्य प्रकारे व्यवस्थापन करणे तितकेच महत्त्वाचे असते. डिफेक्ट मॅनेजमेंट प्रोसेस मुळे डिफेक्ट्सचा प्रवास (Lifecycle) शोध लागल्यापासून (Discovery) ते क्लोजर (Closure) पर्यंत प्रभावीपणे ट्रॅक करता येतो. या प्रक्रियेमध्ये योग्य स्टेकहोल्डर्स / क्लायंट्स (Stakeholders / Clients) जसे की टेस्टर्स (Testers), डेव्हलपर्स (Developers) आणि प्रोजेक्ट मॅनेजर्स (Project Managers) यांना जबाबदाऱ्या (Responsibilities) दिल्या जातात आणि डिफेक्ट्सना त्यांच्या सिव्हेरिटी (Severity) आणि बिझनेस इम्पॅक्ट (Business Impact) नुसार प्राधान्य (Priority) दिले जाते. याशिवाय, ही प्रक्रिया मेट्रिक्स (Metrics) गोळा करण्यास मदत करते, ज्यांचा वापर सतत प्रक्रिया सुधारणा (Continuous Process Improvement) साठी केला जातो. त्यामुळे कोणताही डिफेक्ट अपूर्ण (Unresolved) राहत नाही किंवा भविष्यातील प्रोजेक्ट्समध्ये पुन्हा उद्भवत नाही याची खात्री केली जाते. सुस्पष्ट (Well-defined डिफेक्ट मॅनेजमेंट प्रोसेस, हा क्वालिटी ॲश्युरन्स (QA) चा अविभाज्य भाग असून सॉफ्टवेअर डेव्हलपमेंट लाईफ सायकल (SDLC) मधील सतत सुधारणा (Continuous Improvement) याचा कणा (Backbone) मानला जातो.

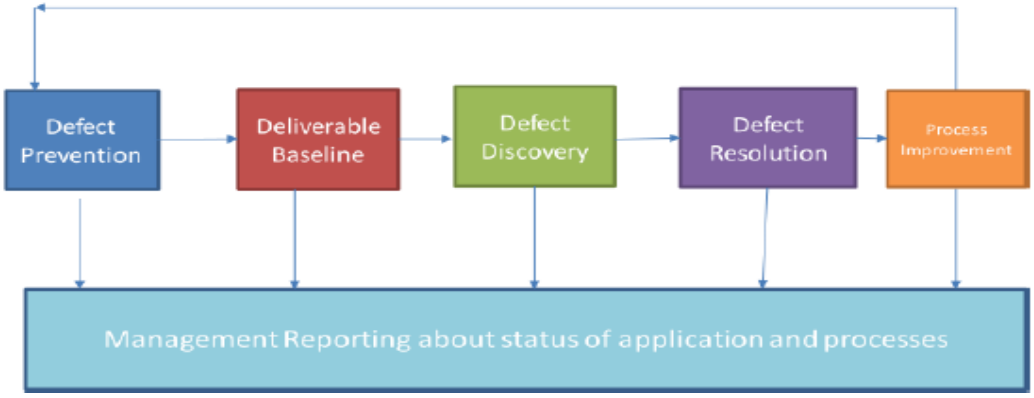

Fig 4.2: डिफेक्ट मॅनेजमेंट प्रोसेस (Defect Management Process)

1. **डिफेक्ट प्रिव्हेंशन (Defect Prevention)**

   डिफेक्ट प्रिव्हेंशन म्हणजे दोष(Defect) होण्यापूर्वीच ते टाळून गुणवत्ता (Quality) आणि उत्पादकता (Productivity) सुधारण्याची प्रक्रिया आहे. दोष पूर्णपणे काढून टाकणे अशक्य असले तरी, प्रतिबंधात्मक धोरणे (Prevention Strategies) सिस्टिममध्ये (System) डिफेक्ट येण्याचा धोका कमी करतात. यामध्ये प्रमाणित पद्धती (Standard Methodologies), रिव्ह्यू, इन्स्पेक्शन, कोडिंग स्टँडर्ड, टेस्ट-ड्रिव्हन डेव्हलपमेंट आणि पीअर रिव्ह्यू

यांचा समावेश होतो. रिक्वायरमेंट्स आणि डिझाइन टप्प्यांत संभाव्य समस्या लवकर ओळखल्यास, डेव्हलपमेंट सायकल) च्या नंतरच्या टप्प्यात डिफेक्ट दुरुस्त करण्याचा खर्च मोठ्या प्रमाणात कमी होतो. डिफेक्ट प्रिव्हेंशन चे मुख्य उद्दिष्ट प्रॉडक्ट मध्ये त्रुटी येण्याची शक्यता कमी करणे किंवा दूर करणे हे आहे.

2. डिलिव्हरेबल बेसलाईन (Deliverable Baseline)

डिलिव्हरेबल बेसलाईन म्हणजे रिक्वायरमेंट्स, डिझाइन किंवा कोड यांच्या स्थिर (Stable) आवृत्तीचा संदर्भ बिंदू होय, ज्याच्या आधारे डिफेक्ट्स मोजले आणि व्यवस्थापित केले जातात. बेसलाईन तयार केल्यामुळे सर्व स्टेकहोल्डर्स सिस्टिमचे अपेक्षित वर्तन (Expected Behavior) काय आहे यावर सहमत होतात. अचूकता (Accuracy) आणि सुसंगतता (Consistency) राखण्यासाठी डिफेक्ट्स फक्त परिभाषित बेसलाईनविरुद्धच लॉग केले जातात. बेसलाईन्स प्रोजेक्ट प्रगती ट्रॅक करण्यासाठी चेकपॉइंट्स म्हणून काम करतात आणि चेंज मॅनेजमेंट साठी अँकर (Anchor) प्रदान करतात.

उदाहरणे(Examples):

- रिक्वायरमेंट डॉक्युमेंट (Requirements Document)
- डिझाइन स्पेसिफिकेशन्स (Design Specifications)
- सोर्स कोड (Source Code)
- टेस्ट केसेस (Test Cases)

3. डिफेक्ट डिस्कव्हरी (Defect Discovery)

डिफेक्ट डिस्कव्हरी म्हणजे रिव्ह्यू, इन्स्पेक्शन, टेस्टिंग) किंवा युजर फीडबॅक यांसारख्या प्रणालीबद्ध क्रियाकलापांद्वारे डिफेक्ट्स ओळखणे आणि नोंदवणे ही प्रक्रिया आहे.

डिफेक्ट्स वेगवेगळ्या स्वरूपाचे असू शकतात—

- फंक्शनल त्रुटी (Functional Errors)
- परफॉर्मन्स समस्या (Performance Issues) जसे की स्लो रिस्पॉन्स टाइम
- युजेबिलिटी समस्या (Usability Problems) जसे की गोंधळात टाकणारा यूजर इंटरफेस

प्रत्येक डिफेक्ट डिफेक्ट ट्रॅकिंग सिस्टिम जसे की JIRA किंवा Bugzilla मध्ये लॉग केला जातो. यामध्ये Defect ID, सिव्हेरिटी(Severity), प्रायोरिटी(Priority), रिप्रोडक्शन स्टेप्स आणि ओनर यांचा समावेश असतो. डिफेक्ट दुरुस्तीचा खर्च वेळेसोबत वाढत असल्यामुळे, रिक्वायरमेंट रिव्ह्यूज, डिझाइन रिव्ह्यूज, युनिट टेस्टिंग, इंटिग्रेशन टेस्टिंग, सिस्टिम टेस्टिंग किंवा यूजर ॲक्सेप्टन्स टेस्टिंग (UAT) दरम्यान लवकर डिफेक्ट शोधणे अत्यंत महत्त्वाचे आहे.

4. डिफेक्ट रिझोल्यूशन (Defect Resolution)

डिफेक्ट रिझोल्यूशन म्हणजे ओळखलेले डिफेक्ट्स विश्लेषण(Analyze) करणे, दुरुस्त करणे(Fix), चाचणी करणे(Test) आणि औपचारिकरीत्या बंद करणे(Close) ही प्रक्रिया आहे. डिफेक्ट रिपोर्ट झाल्यानंतर डेव्हलपर मूळ कारण शोधतो, आवश्यक कोड करेक्शन करतो आणि युनिट टेस्टिंग (Unit Testing) करतो. त्यानंतर टेस्टर्स व्हेरिफिकेशन आणि रिग्रेशन टेस्टिंग द्वारे फिक्स तपासतात.

डिफेक्ट विविध स्टेटसमधून जाऊ शकतो:

ओपन → असाइनड → फिक्स्ड → व्हेरिफाइड → क्लोज्ड

(Open → Assigned → Fixed → Verified → Closed)

समस्या पूर्णपणे सुटली नाही तर डिफेक्ट री-ओपन केला जाऊ शकतो. प्रभावी डिफेक्ट रिझोल्यूशनसाठी टेस्टर्स, डेव्हलपर्स (Testers, Developers) आणि प्रोजेक्ट मॅनेजर्स (Project Managers) यांच्यात स्पष्ट संवाद आणि सहकार्य आवश्यक आहे.

5. प्रोसेस इम्प्रूव्हमेंट (Process Improvement)

प्रोसेस इम्प्रूव्हमेंट मध्ये डिफेक्ट डेटाचे विश्लेषण करून सॉफ्टवेअर डेव्हलपमेंट आणि टेस्टिंग पद्धती सतत सुधारल्या जातात.

मुख्य मेट्रिक्स:

- डिफेक्ट डेन्सिटी (Defect Density)

- मीन टाइम टू रिझोल्यूशन (एमटीटीआर) (Mean Time to Resolution (MTTR)
- डिफेक्ट री-ओपन रेट (Defect Re-Open Rate)

या विश्लेषणातून मिळालेल्या शिकलेले धडे (Lessons Learned) वापरून कोडिंग स्टँडर्ड्स, टेस्टिंग चेकलिस्ट्स (Coding Standards, Testing Checklists) सुधारल्या जातात आणि उत्तम पद्धती (Best Practices) स्वीकारल्या जातात. याचे अंतिम उद्दिष्ट भविष्यात समान डिफेक्ट्सची पुनरावृत्ती टाळणे, गुणवत्ता वाढवणे आणि रीवर्क कॉस्ट कमी करणे हे आहे.

6. मॅनेजमेंट रिपोर्टिंग (Management Reporting)

मॅनेजमेंट रिपोर्टिं ही डिफेक्ट मॅनेजमेंटमधील सहाय्यक क्रिया आहे, ज्यामध्ये डिफेक्ट स्थिती, प्रगती (Progress) आणि ट्रेंड्स (Trends) यांची माहिती नियमितपणे स्टेकहोल्डर्स(Stakeholders) पर्यंत पोहोचवली जाते.

हे रिपोर्ट्स पुढील बाबी स्पष्ट करतात:

- ओपन वि. क्लोज्ड डिफेक्ट्स (Open vs Closed Defects)
- सिव्हेरिटी वितरण (Severity Distribution)
- प्रोजेक्ट स्टॅबिलिटी (Project Stability) वर होणारा एकूण परिणाम

उदाहरण (Example)

परिस्थिती (Scenario): एक ऑनलाईन लर्निंग प्लॅटफॉर्म (LMS – लर्निंग मॅनेजमेंट सिस्टिम) (Online Learning Platform (LMS – Learning Management System)) चे टेस्टिंग चालू आहे.

प्रक्रिया (Process):

1. डिफेक्ट प्रिव्हेंशन (Defect Prevention): डिझाइन रिव्ह्यूज (Design Reviews) दरम्यान टीम युजर ऑथेंटिकेशन (User Authentication) स्पष्टपणे परिभाषित आहे याची खात्री करते. यामध्ये पासवर्ड मजबुती (Password Strength), टू-फॅक्टर ऑथेंटिकेशन (Two-Factor Authentication) आणि सेशन टाइमआउट्स (Session Timeouts) यांसंबंधी रिक्वायरमेंट्स समाविष्ट केल्या जातात, ज्यामुळे सिक्युरिटी लूपहोल्स (Security Loopholes) टाळता येतात.
2. डिलिव्हरेबल बेसलाईन (Deliverable Baseline): ऑथेंटिकेशन मॉड्यूलच्या डिझाइन आणि सोर्स कोड ची बेसलाईन आवृत्ती मंजूर केली जाते आणि संदर्भासाठी कॉन्फिगरेशन मॅनेजमेंट सिस्टिम (Configuration Management System) मध्ये साठवली जाते.
3. डिफेक्ट डिस्कव्हरी (Defect Discovery): सिस्टम टेस्टिंग (System Testing) दरम्यान टेस्टर्सना आढळते की 15 मिनिटे निष्क्रियता (Inactivity) झाल्यानंतरही सिस्टिम युजरला ऑटोमॅटिक लॉगआउट करत नाही, जे सिक्युरिटी रिक्वायरमेंट नुसार अपेक्षित होते. हा डिफेक्ट बगझिला (Bugzilla) मध्ये सिव्हेरिटी = क्रिटिकल (Severity = Critical) म्हणून लॉग केला जातो.
4. डिफेक्ट रिझोल्यूशन (Defect Resolution): डेव्हलपर (Developer) समस्येचे विश्लेषण करतो आणि लक्षात येते की सेशन टाइमआउट व्हॅल्यू (Session Timeout Value) कॉन्फिगरेशन फाइलमध्ये चुकून "Unlimited" असे सेट केले होते. दुरुस्ती (Fix) लागू केली जाते, युनिट टेस्टिंग (Unit Testing) केली जाते आणि नंतर टेस्टर्स द्वारे व्हेरिफाय (Verify) केली जाते. यशस्वी रिटेस्टनंतर डिफेक्ट क्लोज्ड म्हणून मार्क केला जातो.
5. प्रोसेस इम्प्रूव्हमेंट (Process Improvement): डिफेक्ट लॉग्सच्या (Defect Logs) विश्लेषणातून असे आढळते की स्टँडर्ड चेकलिस्ट (Standard Checklist) नसल्यामुळे काही सिक्युरिटी-संबंधित रिक्वायरमेंट्स राहून गेल्या. त्यामुळे टीम भविष्यातील प्रोजेक्ट्ससाठी रिक्वायरमेंट रिव्ह्यू चेकलिस्ट (Requirement Review Checklist) अपडेट करते, ज्यामध्ये सर्व सिक्युरिटी आणि कॉम्प्लायन्स निकष (Compliance Criteria) समाविष्ट केले जातात.
6. मॅनेजमेंट रिपोर्टिंग (Management Reporting): QA लीड (QA Lead) साप्ताहिक डिफेक्ट स्टेटस रिपोर्ट (Defect Status Report) तयार करतो, ज्यामध्ये 95% सिक्युरिटी-संबंधित डिफेक्ट्स दुरुस्त झाले असल्याचे आणि सरासरी टर्नअराउंड टाइम (Turnaround Time) 1.5 दिवस असल्याचे नमूद केले जाते. या रिपोर्टमुळे मॅनेजमेंटला निर्णय घेण्यास मदत होते की लर्निंग मॅनेजमेंट सिस्टिम (LMS) पायलट रिलीजसाठी (Pilot Release) पुरेशी स्थिर (Stable) आहे.

4.2 डिफेक्ट लाईफ सायकल, डिफेक्ट टेम्पलेट (Defect Life Cycle, Defect Template)

4.1.1. डिफेक्ट लाईफ सायकल (Defect Life Cycle)

डिफेक्ट लाईफ सायकल किंवा बग लाईफ सायकल (Bug Life Cycle) म्हणजे एखादा दोष (Defect) त्याच्या संपूर्ण आयुष्यात (Lifetime) ज्या टप्प्यांतून जातो ती प्रक्रिया होय—डिफेक्ट सापडल्यापासून (Discovery) ते तो पूर्णपणे दुरुस्त होऊन बंद (Closure) होईपर्यंत.

हा लाईफ सायकल डिफेक्टच्या विविध स्टेट्स / अवस्था(States) आणि त्यांच्यातील ट्रान्झिशन्स (Transitions) स्पष्टपणे परिभाषित केल्या जातात, जोपर्यंत डिफेक्ट रिझॉल्व्हड (Resolved) आणि क्लोज्ड (Closed) होत नाही.

डिफेक्ट लाईफ सायकल (Defect Life Cycle) मुळे सॉफ्टवेअर टेस्टिंगदरम्यान आढळलेला प्रत्येक डिफेक्ट खालीलप्रमाणे हाताळला जातो:

- डिफेक्ट ट्रॅकिंग टूल (Defect Tracking Tool) जसे की जिरा (JIRA) किंवा बगझिला (Bugzilla) वापरून योग्य प्रकारे ट्रॅक केला जातो.
- योग्य व्यक्तीकडे असाइन (Assign) केला जातो डेव्हलपर (Developer), टेस्टर (Tester) किंवा प्रोजेक्ट मॅनेजर (Project Manager).
- सिव्हेरिटी (Severity) आणि बिझनेस इम्पॅक्ट (Business Impact) नुसार प्राधान्य (Prioritize) केला जातो.
- सॉफ्टवेअर रिलीज होण्यापूर्वी दुरुस्त (Resolve) आणि तपासणी (Verify) केला जातो.ही संरचित प्रक्रिया (Structured Process) डिफेक्ट्स दुर्लक्षित होण्यापासून प्रतिबंध करते, स्पष्ट जबाबदारी सुनिश्चित करते आणि मॅनेजमेंट ला प्रोजेक्टची स्थिती (Project Health) ट्रॅक करण्यास मदत करते.

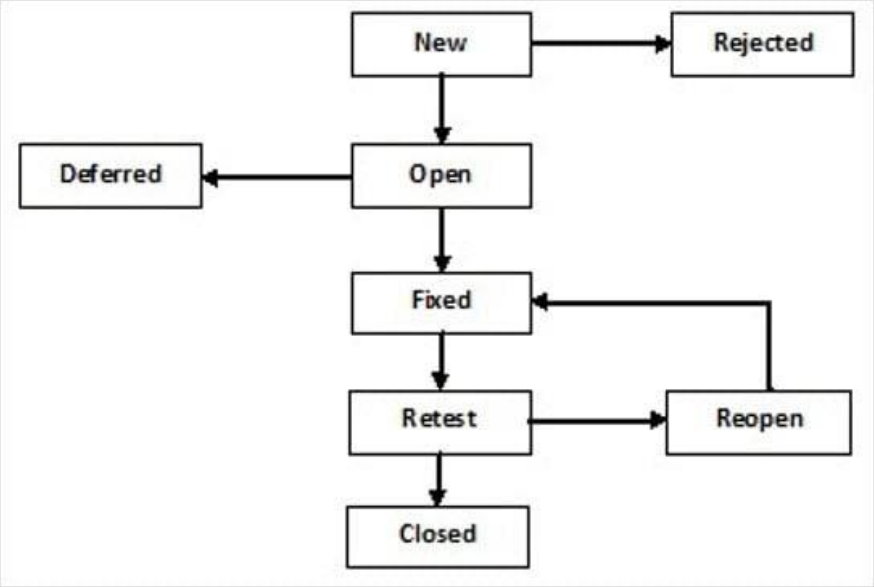

Fig 4.3: डिफेक्ट लाईफ सायकल (Defect Life Cycle)

1. **न्यू (New)**
   जेव्हा टेस्टर (Tester) एखादा डिफेक्ट (Defect) शोधतो, तेव्हा तो प्रथम डिफेक्ट ट्रॅकिंग टूल (Defect Tracking Tool) जसे की JIRA किंवा Bugzilla मध्ये लॉग केला जातो. या टप्प्यावर डिफेक्टचा स्टेटस न्यू (New) असा सेट केला जातो.
2. **रिजेक्टेड (Rejected)**
   जर डेव्हलपर किंवा प्रोजेक्ट मॅनेजर ला असे वाटले की डिफेक्ट वैध नाही—
   उदा.
   - रिक्वायरमेंटचा गैरसमज (Misunderstanding of Requirement)
   - डुप्लिकेट बग (Duplicate Bug)
   - अपेक्षित वर्तन (Expected Behavior)

   तर डिफेक्ट रिजेक्टेड म्हणून मार्क केला जातो.

3. **ओपन (Open)**
   जे डिफेक्ट्स वैध (Valid) असतात ते ओपन स्टेटसमध्ये हलवले जातात. याचा अर्थ डिफेक्ट मान्य (Acknowledged) केला आहे आणि तो दुरुस्तीसाठी (Fixing) तयार आहे.
4. **डिफर्ड (Deferred)**
   कधी कधी डिफेक्ट वैध असतो पण तातडीचा (Urgent) नसतो.
   उदा.:
   - कमी प्राधान्याचा दोष (Low Priority Defect)
   - सुधारणा विनंती (Enhancement Request)

   अशा डिफेक्ट्सना पुढील रिलीजसाठी (Future Release) पुढे ढकलले जाते आणि त्यांचा स्टेटस डिफर्ड असा सेट केला जातो.
5. **फिक्स्ड (Fixed)**
   डेव्हलपर कोडमधील डिफेक्ट दुरुस्त करतो आणि स्टेटस फिक्स्ड असा बदलतो. याचा अर्थ डिफेक्ट दुरुस्त केला आहे, पण अजून व्हेरिफिकेशन झालेले नाही.
6. **रिटेस्ट (Retest)**
   टेस्टर पुन्हा ॲप्लिकेशनची चाचणी करतो, डिफेक्ट खरोखरच दुरुस्त झाला आहे का हे तपासण्यासाठी. जर डिफेक्ट दुरुस्त झाला असेल, तर तो पुढील टप्प्याकडे जातो; अन्यथा तो री-ओपन केला जातो.
7. **री-ओपन (Reopen)**
   जर डेव्हलपरने केलेल्या फिक्सनंतरही डिफेक्ट अस्तित्वात असेल, तर टेस्टर तो री-ओपन करतो. ही सायकल डिफेक्ट पूर्णपणे दुरुस्त होईपर्यंत चालू राहते.
8. **क्लोज्ड (Closed)**
   जेव्हा टेस्टर खात्री करतो की डिफेक्ट पूर्णपणे दुरुस्त झाला आहे आणि पुन्हा निर्माण होत नाही (Not Reproducible), तेव्हा त्याचा स्टेटस क्लोज्ड केला जातो.
   याचा अर्थ डिफेक्ट यशस्वीपणे प्रॉडक्ट मधून काढून टाकण्यात आला आहे.

उदाहरण (Example)

परिस्थिती (Scenario): ऑनलाईन लायब्ररी मॅनेजमेंट सिस्टिममधील डिफेक्ट लाईफ सायकल (Defect Life Cycle in LMS – Online Library Management System)

प्रक्रिया (Process):

1. न्यू (New)
   - टेस्टिंगदरम्यान QA टेस्टर PDF फॉरमॅटमध्ये असाइनमेंट अपलोड करण्याचा प्रयत्न करतो.
   - PDF सपोर्ट असतानाही सिस्टिम फाइल रिजेक्ट करते.
   - टेस्टर हा दोष (Defect) डिफेक्ट ट्रॅकिंग टूल (Defect Tracking Tool) जसे की JIRA किंवा Bugzilla मध्ये लॉग करतो.

डिफेक्ट तपशील (Defect Details):

- शीर्षक (Title): असाइनमेंट मॉड्यूलमध्ये PDF अपलोड काम करत नाही" ("PDF upload not working in Assignment Module")
- सिव्हेरिटी (Severity): High (विद्यार्थी असाइनमेंट सबमिट करू शकत नाहीत)
- रिप्रोडक्शन स्टेप्स(Steps to Reproduce), अपेक्षित निकाल(Expected Result), प्रत्यक्ष निकाल(Actual Result), स्क्रीनशॉट्स(Screenshots)

डिफेक्ट स्टेटस (Defect Status) = New

2. असाइनड (Assigned)
   - प्रोजेक्ट मॅनेजर किंवा टेस्ट लीड डिफेक्टचे रिव्ह्यू करतो.
   - डिफेक्ट वैध (Valid) असल्याची खात्री करून तो असाइनमेंट अपलोड मॉड्यूल (Assignment Upload

Module) साठी जबाबदार असलेल्या डेव्हलपर कडे असाइन केला जातो.

स्टेटस(Status) = Assigned

3. ओपन (Open)
   - डेव्हलपर डिफेक्ट स्वीकारतो.
   - लर्निंग मॅनेजमेंट सिस्टिम मधील फाइल अपलोड लॉजिकचे विश्लेषण (Analysis) सुरू करतो.
   - लक्षात येते की व्हॅलिडेशन स्क्रिप्ट फक्त .docx आणि .txt फॉरमॅट्सना परवानगी देते, पण .pdf ला नाही.

   स्टेटस(Status) = Open
4. फिक्स्ड (Fixed)
   - डेव्हलपर व्हॅलिडेशन लॉजिकमध्ये बदल करून .pdf फाइल्सना परवानगी देतो.
   - योग्य कार्यक्षमता सुनिश्चित करण्यासाठी युनिट टेस्टिंग (Unit Testing) करतो.

   डिफेक्ट स्टेटस फिक्स्ड(Fixed) असा अपडेट केला जातो.
5. रिटेस्ट (Retest)
   - टेस्टर फिक्स केलेला बिल्ड प्राप्त करतो आणि पुन्हा टेस्टिंग करतो.
   - PDF फाइल पुन्हा अपलोड करून तपासतो.
   - जर अपलोड यशस्वी झाला, तर पुढील स्टेपकडे जातो.

   स्टेटस (Status) = Retest
6. व्हेरिफाइड (Verified)
   - टेस्टर खात्री करतो की समस्या पूर्णपणे निराकरण झाली आहे.
   - इतर फाइल टाइप्स (.docx, .txt) वर कोणतेही साइड इफेक्ट्स (Side Effects) दिसत नाहीत.

   स्टेटस (Status) = Verified
7. क्लोज्ड (Closed)
   - डिफेक्ट फिक्स झाल्याची खात्री झाल्यावर टेस्टर स्टेटस क्लोज्ड करतो.
   - याचा अर्थ डिफेक्ट ऑनलाईन लायब्ररी मॅनेजमेंट सिस्टिम (Online Library Management System) मधून यशस्वीपणे काढून टाकण्यात आला आहे.
8. री-ओपन (Reopen) – If Issue Persists
   - जर रिटेस्टिंगदरम्यान डिफेक्ट पुन्हा आढळला (उदा. मोठ्या फाइल साइजसाठी PDF अपलोड फेल होणे),
   - तर टेस्टर डिफेक्ट री-ओपन करतो.
   - त्यानंतर लाईफ सायकल पुन्हा ओपन → फिक्स्ड → रीटेस्ट → क्लोज्ड (Open → Fixed → Retest → Closed) या क्रमाने पुढे चालू राहते.

4.2.2 डिफेक्ट टेम्पलेट (Defect Template)

डिफेक्ट रिपोर्ट (Defect Report) हा टेस्टर्सकडून ॲप्लिकेशनमध्ये आढळलेल्या समस्या डेव्हलपमेंट टीम आणि प्रोजेक्ट टीम पर्यंत पोहोचवण्यासाठी वापरला जाणारा एक औपचारिक (Formal) दस्तऐवज आहे. यामुळे डिफेक्ट्स सिस्टेमॅटिकली ट्रॅक, प्राधान्यक्रम (Prioritize) आणि निराकरण (Resolve) केले जातात.

डिफेक्ट टेम्पलेटमधील मानक घटक (Standard Fields of a Defect Template)

1. **डिफेक्ट आयडी (Defect ID)**
   - काय आहे (What it is): प्रत्येक डिफेक्टसाठी दिला जाणारा युनिक ओळख क्रमांक (उदा. DEF_001, BUG_120, ATM_05).
   - महत्त्व (Why it matters): डिफेक्ट लाईफ सायकलमध्ये ट्रॅकिंग सोपे होते आणि अनेक डिफेक्ट्स असताना गोंधळ टाळला जातो. हा आयडी सहसा JIRA, Bugzilla, QC सारख्या टूल्सद्वारे ऑटो-जनरेट केला जातो.
2. **प्रोजेक्ट (Project)**
   - काय आहे (What it is): ज्या प्रोजेक्टमध्ये डिफेक्ट सापडला त्याचे नाव (उदा. ATM Simulator, Library Management System).

- महत्त्व (Why it matters): मोठ्या संस्थांमध्ये अनेक प्रोजेक्ट्स असतात; त्यामुळे वेगवेगळ्या सिस्टिम्समधील डिफेक्ट्स मिसळले जात नाहीत.

3. प्रॉडक्ट (Product)

- काय आहे (What it is): ज्या विशिष्ट ॲप्लिकेशन किंवा प्रॉडक्टमध्ये डिफेक्ट आढळला (उदा. Online Library Portal).
- महत्त्व (Why it matters): एका प्रोजेक्टमधून अनेक प्रॉडक्ट्स असू शकतात; हा फील्ड नेमका डिफेक्ट कुठे आहे ते स्पष्ट करतो.

4. रिलीज व्हर्जन (Release Version)

- काय आहे (What it is): ज्या अधिकृत व्हर्जनमध्ये फंक्शन अपेक्षितपणे काम करायला हवे होते (उदा. v1.0).
- महत्त्व (Why it matters): डिफेक्ट प्रॉडक्शनमध्ये आहे की फक्त टेस्ट बिल्डमध्ये आहे हे समजते.
- हॉटफिक्स (Hotfix) आवश्यक आहे का हे ठरवता येते.

5. डिटेक्टेड बिल्ड व्हर्जन (Detected Build Version)

- काय आहे (What it is): ज्या बिल्डमध्ये प्रत्यक्ष डिफेक्ट आढळला (उदा. Build v1.1).
- महत्त्व (Why it matters): कोणत्या बिल्डमध्ये डिफेक्ट आला हे डेव्हलपर्सना अचूकपणे समजते.

6. मॉड्यूल (Module)

- काय आहे (What it is): सिस्टिममधील नेमका भाग जिथे डिफेक्ट आहे (उदा. Login Module, Payment Gateway).
- महत्त्व (Why it matters): डेव्हलपरचा वेळ वाचतो आणि डिफेक्टचे लोकेशन लगेच समजते.

7. सारांश (Summary)

- काय आहे (What it is): एका ओळीत डिफेक्टचे वर्णन (उदा. "Withdrawal limited to ₹3000 only").
- महत्त्व (Why it matters): बहुतेक स्टेकहोल्डर्स (Stakeholders) फक्त सारांश वाचतात; म्हणून तो स्पष्ट आणि नेमका असावा.

8. सविस्तर वर्णन(Description)

- काय आहे (What it is): डिफेक्टचे सविस्तर स्पष्टीकरण—समस्या, परिणाम, तांत्रिक माहिती.
- महत्त्व (Why it matters): टेस्टर आणि डेव्हलपर यांच्यात गैरसमज टाळतो.

9. पुन्हा तयार करण्याच्या पायऱ्या (Steps to Reproduce)

- काय आहे (What it is): डिफेक्ट कसा सापडला याचे क्रमवार स्टेप्स.
- महत्त्व (Why it matters): डेव्हलपरला डिफेक्ट रिप्रोड्यूस करता आला नाही तर तो दुरुस्त होऊ शकत नाही.

10. प्रत्यक्ष निकाल (Actual Result)

- काय आहे (What it is): सिस्टिम सध्या चुकीचे काय वर्तन दाखवत आहे.
- महत्त्व (Why it matters): नेमकी चूक काय आहे हे स्पष्ट होते.

11. अपेक्षित निकाल (Expected Result)

- काय आहे (What it is): डिफेक्ट नसता तर सिस्टिम कसे वागले असते.
- महत्त्व (Why it matters): Actual vs Expected मधील फरक स्पष्ट होतो—हाच डिफेक्टचा गाभा आहे.

12. ॲटॅचमेंट्स (Attachments)

- काय आहे (What it is): स्क्रीनशॉट्स, लॉग्स, व्हिडिओज (Screenshots, Logs, Videos) इत्यादी पुरावे.
- महत्त्व (Why it matters): दृश्य पुराव्यामुळे वाद टाळले जातात आणि वेळ वाचतो.

13. टीप (Remarks)

- काय आहे (What it is): अतिरिक्त माहिती—बिझनेस इम्पॅक्ट किंवा तात्पुरता वर्कअराउंड.
- महत्त्व (Why it matters): डिफेक्ट प्रायोरिटायझेशनमध्ये मॅनेजमेंटला मदत होते.

14. सिव्हेरिटी (Severity)

- काय आहे (What it is): तांत्रिक परिणामाची तीव्रता क्रिटिकल, हाय, मीडियम, लो (Critical, High, Medium, Low).
- महत्त्व (Why it matters): सिस्टिम किती प्रमाणात बिघडते ते दर्शवते.
  उदा. System Crash = Critical.

15. प्रायोरिटी (Priority)

- काय आहे (What it is): बिझनेसदृष्ट्या दुरुस्तीची तातडी.
- महत्त्व (Why it matters): प्रोजेक्ट मॅनेजर ठरवतो.
  उदा. Spelling mistake = Low Severity पण Brand Name मध्ये असल्यास High Priority.

16. रिपोर्ट करणारा (Reported By)

- काय आहे (What it is): डिफेक्ट लॉग करणाऱ्या टेस्टरचे नाव.
- महत्त्व (Why it matters): पुढील स्पष्टीकरणासाठी संपर्क सोपा होतो.

17. असाइन केलेले (Assigned To)

- काय आहे (What it is): डिफेक्ट दुरुस्त करणाऱ्या व्यक्तीचे नाव.
- महत्त्व (Why it matters): जबाबदारी (Accountability) स्पष्ट होते.

18. स्टेटस (Status)

- काय आहे (What it is): डिफेक्टची सध्याची अवस्था (न्यू → असाइनड → ओपन → फिक्स्ड → रीटेस्ट → व्हेरिफाइड → क्लोज्ड → रिओपनड → डिफर्ड → रिजेक्टेड.)
- महत्त्व (Why it matters): डिफेक्ट लाईफ सायकल आणि प्रगती स्पष्टपणे दिसते.

उदाहरण (Example)

परिस्थिती (Scenario): एका युजरने एटीएम सिम्युलेटर ॲप्लिकेशन (ATM Simulator Application) मधून ₹5000 रक्कम काढण्याचा (Withdraw) प्रयत्न केला.

प्रक्रिया (Process): डिफेक्ट टेम्पलेट वापरून डिफेक्ट रिपोर्ट (Defect Report Using Defect Template)

Field	Details (English)	तपशील (मराठी)
ID	Def_02	Def_02
Project	ATM Simulator	एटीएम सिम्युलेटर
Product	Cash Simulator ATM	कॅश सिम्युलेटर एटीएम
Release Version	v1.0	आवृत्ती v1.0
Module	Cash Withdrawal → Denomination Selection	रोख रक्कम काढणे → डिनॉमिनेशन निवड
Detected Build Version	v1.1	आढळलेली बिल्ड आवृत्ती v1.1
Summary	ATM does not allow withdrawal above ₹3000 due to limited denomination options.	मर्यादित डिनॉमिनेशन पर्यायांमुळे एटीएम ₹3000 पेक्षा जास्त रक्कम काढू देत नाही.
Description	The withdrawal module only provides fixed denomination options up to ₹3000. Users cannot withdraw higher amounts (e.g., ₹5000), which is inconsistent with real ATM functionality.	विथड्रॉवल मॉड्यूलमध्ये फक्त ₹3000 पर्यंतचे ठराविक डिनॉमिनेशन पर्याय उपलब्ध आहेत. त्यामुळे वापरकर्ते ₹5000 सारखी जास्त रक्कम काढू शकत नाहीत, जे प्रत्यक्ष एटीएम कार्यपद्धतीशी सुसंगत नाही.
Steps to	1. Open ATM Simulator.	1. एटीएम सिम्युलेटर उघडा.

Reproduce	2. Select Our Programs. 3. Go to Digital Inclusion Tools → Cash Machine Simulator (ATM). 4. Choose language and enter simulator. 5. Insert card and select account type. 6. Go to Other Functions → Cash Withdrawal. 7. Attempt withdrawal of ₹5000.	2. Our Programs निवडा. 3. Digital Inclusion Tools → Cash Machine Simulator (ATM) येथे जा. 4. भाषा निवडा व सिम्युलेटरमध्ये प्रवेश करा. 5. कार्ड घाला व खाते प्रकार निवडा. 6. Other Functions → Cash Withdrawal निवडा. 7. ₹5000 रक्कम काढण्याचा प्रयत्न करा.
Actual Result	System restricts withdrawal to fixed denominations (max ₹3000).	सिस्टीम फक्त ठराविक डिनॉमिनेशनपर्यंत (कमाल ₹3000) विथड्रॉवल मर्यादित करते.
Expected Result	System should allow withdrawal above ₹3000 if balance permits, by either adding more denominations or allowing user-input amounts.	खात्यात शिल्लक असल्यास ₹3000 पेक्षा जास्त रक्कम काढण्याची परवानगी द्यावी, त्यासाठी अधिक डिनॉमिनेशन किंवा युजर-इनपुट रक्कम पर्याय द्यावेत.
Attachments		
Remarks	Restricts simulator usability and causes inconvenience. Not aligned with actual ATM use.	सिम्युलेटरचा वापर मर्यादित होतो व वापरकर्त्यांना अडचण येते. प्रत्यक्ष एटीएम वापराशी सुसंगत नाही.
Defect Severity	High	उच्च
Defect Priority	High	उच्च
Reported By	abc (QA Tester)	abc (QA टेस्टर)
Assigned To	xyz (Developer)	xyz (डेव्हलपर)
Status	Assigned	नेमणूक केलेली

4.3 डिफेक्टचा अपेक्षित परिणाम, डिफेक्ट शोधण्याच्या तंत्रे, डिफेक्ट रिपोर्टिंग (Estimate Expected Impact of a Defect, Techniques for Finding Defects, Reporting a Defect)

4.3.1 डिफेक्टचा अपेक्षित परिणाम मोजणे (Estimate Expected Impact of a Defect)

फेक्टचा अपेक्षित परिणाम (Expected Impact of a Defect) म्हणजे एखादा डिफेक्ट (Defect) जर दुरुस्त केला गेला नाही, तर तो सॉफ्टवेअर सिस्टिम, युजर्स, बिझनेस प्रक्रिया आणि टेस्टिंग क्रियाकलाप यांच्यावर कोणते संभाव्य परिणाम (Consequences) होऊ शकतात याचा अंदाज होय. हा परिणाम फंक्शनॅलिटी, रिलायबिलिटी, युजेबिलिटी आणि प्रॉडक्टच्या एकूण गुणवत्ता (Overall Quality) या दृष्टीने डिफेक्ट किती गंभीर आहे हे अधोरेखित करतो.

साधारणपणे अपेक्षित परिणाम (Expected Impact) खालील घटकांच्या आधारे विश्लेषित केला जातो:

- बिझनेस इम्पॅक्ट (Business Impact): महसूल तोटा (Loss of Revenue), ग्राहक असमाधान (Customer Dissatisfaction)
- युजर इम्पॅक्ट (User Impact): युजेबिलिटी समस्या (Usability Issues), गैरसोय (Inconvenience), चुकीचे निकाल (Incorrect Results)
- सिस्टम / टेक्निकल इम्पॅक्ट (System / Technical Impact): मुख्य फंक्शनॅलिटी बिघडणे (Breakdown of Core Functionality), न तपासलेल्या परिस्थिती (Untested Scenarios)
- सिव्हेरिटी आणि प्रायोरिटी (Severity & Priority): डिफेक्ट किती गंभीर आहे (How Critical) आणि तो किती लवकर दुरुस्त करणे आवश्यक आहे (Urgency of Fix)

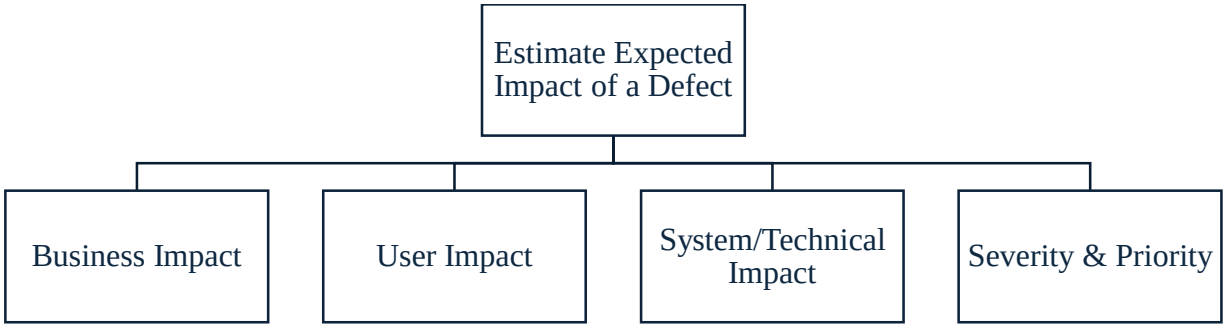

Fig 4.4: डिफेक्टचा अपेक्षित परिणाम मोजणे (Estimate Expected Impact of a Defect)

1. बिझनेस दृष्टीकोन (Business Perspective)

कोअर फंक्शन्स (Core Functions) मधील डिफेक्ट्स—उदा. एटीएम कॅश विथड्रॉवल, बँकिंग ट्रान्झॅक्शन्स, ऑनलाईन पेमेंट्स (ATM Cash Withdrawal, Banking Transactions, Online Payments)—थेट ग्राहक विश्वास (Customer Trust) आणि कंपनीची प्रतिमा (Company Reputation) यावर परिणाम करतात. ग्राहकांसमोर वापरल्या जाणाऱ्या मॉड्यूल्समध्ये (Customer-facing Modules) अगदी लहान डिफेक्ट्सही विश्वसनीयता(Credibility) कमी करू शकतात.

2. युजर दृष्टीकोन (User Perspective)

महत्त्वाची फीचर्स ब्लॉक झाल्यास युजर्सना गैरसोय किंवा निराशा अनुभवावी लागते. सिम्युलेशन सिस्टिम्स किंवा प्रशिक्षण सिस्टिम्स मध्ये डिफेक्ट असल्यास चुकीचे शिकण्याचे परिणाम (Incorrect Learning Outcomes) किंवा वास्तवाशी विसंगत अपेक्षा निर्माण होऊ शकतात.

3. सिस्टम / टेक्निकल दृष्टीकोन (System / Technical Perspective)

काही टेस्ट सिनेरिओज—उदा. जास्त रकमेचे विथड्रॉवल, एरर मेसेजेस—टेस्ट करता येत नाहीत. डिफेक्ट दुरुस्त न केल्यास सिस्टम अस्थिरता (System Instability) किंवा डेटा अखंडतेच्या समस्या (Data Integrity Issues) निर्माण होऊ शकतात.

4. सिव्हेरिटी आणि प्रायोरिटी (Severity and Priority)

सिव्हेरिटी: डिफेक्टमुळे सिस्टिम किंवा बिझनेसला होणाऱ्या नुकसानीची तीव्रता मोजते. उदा: जर कॅश विथड्रॉवल पूर्णपणे होत नसेल, तर तो क्रिटिकल सिव्हेरिटी डिफेक्ट आहे.

प्रायोरिटी: डिफेक्ट किती लवकर दुरुस्त करणे आवश्यक आहे हे ठरवते.

उदाहरण (Example)

परिस्थिती (Scenario): प्रोजेक्ट (Project): एटीएम सिम्युलेटर (ATM Simulator) आणि डिफेक्ट (Defect): विथड्रॉवल पर्यायातून जास्तीत जास्त ₹3000 इतकीच रक्कम काढता येते; त्यापेक्षा जास्त रक्कम एंटर करता येत नाही.

प्रक्रिया: अपेक्षित परिणाम विश्लेषण (Expected Impact Analysis)

1. बिझनेस इम्पॅक्ट (Business Impact): खरे ATM युजर्स कॅश विथड्रॉवल (Cash Withdrawal) मध्ये लवचिकता अपेक्षित करतात. ₹3000 मर्यादा असल्यामुळे असमाधान (Dissatisfaction) आणि सिम्युलेटरच्या अचूकतेवर (Accuracy) विश्वास कमी होतो. प्रत्यक्ष ATM मध्ये असा डिफेक्ट असल्यास ग्राहक तक्रारी (Customer Complaints) आणि बँकेच्या प्रतिमेला धक्का बसू शकतो.

2. युजर इम्पॅक्ट (User Impact): ATM Simulator वापरणारे विद्यार्थी / प्रशिक्षणार्थी वास्तववादी विथड्रॉवल सिनेरिओजचा सराव करू शकत नाहीत. त्यामुळे शिकण्याचे परिणाम मध्ये अंतर निर्माण होते, कारण प्रत्यक्ष ATM मध्ये खात्याच्या मर्यादेनुसार जास्त रक्कम काढता येते.
3. सिस्टम / टेक्निकल इम्पॅक्ट (System / Technical Impact): उच्च रकमेचे व्यवहार (High-value Transactions), दैनिक मर्यादा एरर हँडलिंग (Daily Limit Error Handling),कॅश उपलब्धता तपासणी (Cash Availability Checks) अशा सिनेरिओज टेस्ट करता येत नाहीत. त्यामुळे एंड-टू-एंड टेस्टिंग कव्हरेज (End-to-End Testing Coverage) मर्यादित राहते.
4. सिव्हेरिटी (Severity): हाय सिव्हेरिटी कारण हा डिफेक्ट कॅश विथड्रॉवल ही कोअर फंक्शनॅलिटी थेट ब्लॉक करतो.

4.3.2 डिफेक्ट शोधण्याच्या तंत्रे (Techniques for Finding Defects)

सॉफ्टवेअरमधील डिफेक्ट्स (Defects)शोधण्यासाठी मुख्यतः तीन प्रकारच्या तंत्रांचा वापर केला जातो: स्टॅटिक (Static), डायनॅमिक (Dynamic) आणि ऑपरेशनल (Operational). प्रत्येक तंत्राची स्वतःची पद्धत (Approach), सॉफ्टवेअर डेव्हलपमेंट लाईफ सायकल – SDLC मधील वापराचा टप्पा आणि वेगवेगळ्या प्रकारचे डिफेक्ट्स शोधण्याची प्रभावीता (Effectiveness) असते.

1. **स्टॅटिक तंत्रे (Static Techniques):** ही तंत्रे सॉफ्टवेअर चालविल्याशिवाय कोड एक्झिक्युट न करता (Without Executing the Code) डिफेक्ट्स शोधण्यासाठी वापरली जातात. यामध्ये रिव्ह्यूज (Reviews), इन्स्पेक्शन्स (Inspections) आणि वॉकथ्रूज (Walkthroughs) यांचा समावेश होतो. ही तंत्रे प्रामुख्याने रिक्वायरमेंट्स (Requirements) आणि डिझाइन (Design) टप्प्यांत उपयुक्त ठरतात.
2. **डायनॅमिक तंत्रे (Dynamic Techniques):** ही तंत्रे सॉफ्टवेअर प्रत्यक्ष चालवून (Executing the Software) डिफेक्ट्स शोधतात. यामध्ये टेस्टिंग (Testing)—जसे की युनिट टेस्टिंग (Unit Testing), इंटिग्रेशन टेस्टिंग (Integration Testing), सिस्टम टेस्टिंग (System Testing) आणि ॲक्सेप्टन्स टेस्टिंग (Acceptance Testing)—यांचा समावेश होतो. रनटाइम एरर्स, लॉजिक चुका आणि परफॉर्मन्स समस्या (Performance Issues) शोधण्यासाठी ही तंत्रे प्रभावी असतात.
3. **ऑपरेशनल तंत्रे (Operational Techniques):** ही तंत्रे सॉफ्टवेअर प्रत्यक्ष वापरात (Real-world Usage) असताना डिफेक्ट्स ओळखण्यासाठी वापरली जातात. यामध्ये युजर फीडबॅक (User Feedback), प्रॉडक्शन मॉनिटरिंग (Production Monitoring) आणि एरर लॉग्स (Error Logs) यांचा समावेश होतो. प्रत्यक्ष युजर बिहेवियर (User Behavior) मधून येणाऱ्या समस्यांचा शोध घेण्यासाठी ही तंत्रे उपयुक्त ठरतात.

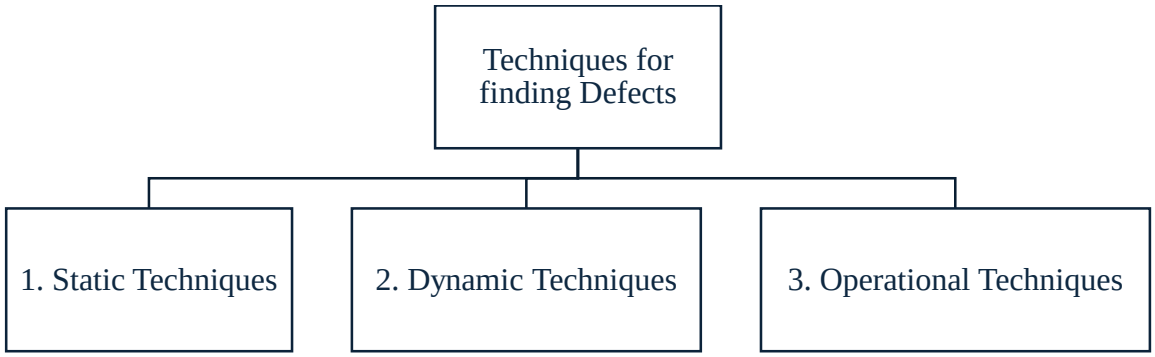

Fig 4.5: डिफेक्ट शोधण्याच्या तंत्रे (Techniques for Finding Defects)

1. स्टॅटिक टेक्निक्स (Static Techniques)

स्टॅटिक टेक्निक्स (Static Techniques) या प्रोग्राम एक्झिक्यूट न करता डिफेक्ट्स (दोष) शोधण्यासाठी वापरल्या जातात. या तंत्रांचा भर रिक्वायरमेंट्स (Requirements), डिझाइन (Design) आणि कोड (Code) यांचे विश्लेषण करण्यावर असतो. ही तंत्रे डेव्हलपमेंटच्या (Development) सुरुवातीच्या टप्प्यात वापरली जात असल्यामुळे डिफेक्ट दुरुस्तीचा खर्च कमी होतो.

स्टॅटिक टेक्निक्सची उदाहरणे (Examples of Static Techniques): रिव्ह्यूज आणि इन्स्पेक्शन्स (Reviews and Inspections)

- इन्फॉर्मल रिव्ह्यू (Informal Review): ऑथर (Author) ला पीअर्स (Peers) कडून अनौपचारिक फीडबॅक मिळतो.
- वॉकथ्रू (Walkthrough): ऑथर (Author) टीमसमोर आर्टिफॅक्ट (Artifact) सादर करून चर्चा करतो.
- पीअर रिव्ह्यू / इन्स्पेक्शन (Peer Review / Inspection): चेकलिस्ट्स (Checklists) आणि स्टँडर्ड्स (Standards) वापरून केली जाणारी औपचारिक प्रक्रिया.
- स्टॅटिक कोड ॲनालिसिस (Static Code Analysis): ऑटोमेटेड टूल्स (Automated Tools) कोड स्कॅन करून खालील त्रुटी शोधतात:
  - सिन्टॅक्स एरर्स (Syntax Errors) आणि कोडिंग स्टँडर्ड व्हायोलेशन्स (Coding Standard Violations)
  - अनयूज्ड व्हेरिएबल्स (Unused Variables), डेड कोड (Dead Code), इन्फिनिट लूप्स (Infinite Loops)
  - सिक्युरिटी व्हल्नरेबिलिटीज (Security Vulnerabilities)
- आर्किटेक्चरल ॲनालिसिस (Architectural Analysis): सायक्लोमॅटिक कॉम्प्लेक्सिटी (Cyclomatic Complexity) सारख्या मेट्रिक्स वापरून कोड कॉम्प्लेक्सिटी तपासली जाते आणि डिफेक्ट-प्रोन मॉड्यूल्स ओळखले जातात.

इम्पॅक्ट (Impact): डिफेक्ट्स लवकर सापडतात, रीवर्क कॉस्ट (Rework Cost) कमी होते आणि मेंटेनबिलिटी (Maintainability) सुधारते.

2. डायनॅमिक टेक्निक्स (Dynamic Techniques)

डायनॅमिक टेक्निक्स (Dynamic Techniques) मध्ये सॉफ्टवेअर प्रत्यक्ष एक्झिक्यूट करून त्याचे रनटाइम बिहेवियर (Runtime Behavior) निरीक्षण केले जाते. प्रोग्राम एक्झिक्यूशनदरम्यान होणारे डिफेक्ट्स शोधण्यासाठी ही तंत्रे अत्यंत महत्त्वाची आहेत.

डायनॅमिक टेक्निक्सची उदाहरणे (Examples of Dynamic Techniques)

- फंक्शनल टेस्टिंग (Functional Testing): फीचर्स (Features) रिक्वायरमेंट्सनुसार काम करतात का हे तपासते.
  - बाउंडरी व्हॅल्यू ॲनालिसिस (Boundary Value Analysis)
  - इक्विव्हॅलन्स पार्टीशनिंग (Equivalence Partitioning)
  - डिसिजन टेबल टेस्टिंग (Decision Table Testing)
- नॉन-फंक्शनल टेस्टिंग (Non-functional Testing):
  - परफॉर्मन्स आणि लोड टेस्टिंग (Performance and Load Testing) – रिस्पॉन्स टाइम्स (Response Times), स्केलेबिलिटी (Scalability)
  - युजेबिलिटी टेस्टिंग (Usability Testing) – ईज ऑफ यूज (Ease of Use), यूजर इंटरफेस इश्यूज (User Interface Issues)
  - सिक्युरिटी टेस्टिंग / पेनिट्रेशन टेस्टिंग (Security Testing / Penetration Testing) – व्हल्नरेबिलिटीज (Vulnerabilities)
- एक्सप्लोरेटरी टेस्टिंग (Exploratory Testing): टेस्टर (Tester) सर्जनशीलतेचा वापर करून अनपेक्षित बग्स (Bugs) शोधतो.
- ऑटोमेटेड टेस्टिंग (Automated Testing): रिग्रेशन (Regression) आणि रिलायबिलिटी (Reliability) साठी टेस्ट स्क्रिप्ट्स (Test Scripts) वारंवार चालवल्या जातात.
- डिबगिंग (Debugging): प्रोग्राम कोड स्टेप-बाय-स्टेप चालवून एरर्स (Errors) चे रूट कॉज (Root Cause) शोधले जाते.

इम्पॅक्ट (Impact): क्रॅशेस (Crashes), इनकररेक्ट आउटपुट्स (Incorrect Outputs) आणि परफॉर्मन्स बॉटलनेक्स (Performance Bottlenecks) यांसारख्या रनटाइम समस्या उघड होतात.

3. ऑपरेशनल टेक्निक्स (Operational Techniques)

ऑपरेशनल टेक्निक्स (Operational Techniques) या सॉफ्टवेअर प्रत्यक्ष वापरात (रिअल-वर्ल्ड एन्व्हायर्नमेंट) आल्यानंतर डिफेक्ट्स शोधण्यासाठी वापरल्या जातात. या तंत्रांमध्ये एंड-युजर्स (End Users), मॉनिटरिंग (Monitoring) आणि फीडबॅक लूप्स (Feedback Loops) यांचा वापर होतो.

ऑपरेशनल टेक्निक्सची उदाहरणे (Examples of Operational Techniques)

- यूजर ॲक्सेप्टन्स टेस्टिंग – यूएटी (User Acceptance Testing – UAT): अंतिम डिप्लॉयमेंटपूर्वी सॉफ्टवेअर बिझनेस नीड्स (Business Needs) पूर्ण करते का हे तपासते.
- फीडबॅक आणि मॉनिटरिंग (Feedback and Monitoring): युजर फीडबॅक (User Feedback), क्रॅश रिपोर्ट्स (Crash Reports) आणि लॉग्स (Logs) यांच्या माध्यमातून टेस्टिंगमध्ये राहून गेलेले डिफेक्ट्स ओळखले जातात.
- रूट कॉज ॲनालिसिस – आरसीए (Root Cause Analysis – RCA): प्रॉडक्शन (Production) मधील डिफेक्ट्सचे मूळ कारण शोधून त्यांची पुनरावृत्ती टाळली जाते.

इम्पॅक्ट (Impact): कस्टमर सॅटिस्फॅक्शन (Customer Satisfaction) सुनिश्चित होते, कंटिन्युअस इम्प्रूव्हमेंट (Continuous Improvement) साठी इनसाइट्स मिळतात आणि रिपिटेटिव्ह इश्यूज (Repetitive Issues) टाळले जातात.

Table 4.2: स्टॅटिक टेक्निक्स, डायनॅमिक टेक्निक्स आणि ऑपरेशनल टेक्निक्स यांची तुलना (Comparison between Static Techniques, Dynamic Techniques and Operational Techniques)

घटक (Parameter)	स्टॅटिक टेक्निक्स (Static Techniques)	डायनॅमिक टेक्निक्स (Dynamic Techniques)	ऑपरेशनल टेक्निक्स (Operational Techniques)
मूलभूत संकल्पना / Basic Concept	प्रोग्राम एक्झिक्यूट न करता डिफेक्ट्स शोधणे	सॉफ्टवेअर एक्झिक्यूट करून डिफेक्ट्स शोधणे	सॉफ्टवेअर प्रत्यक्ष वापरात आल्यानंतर डिफेक्ट्स शोधणे
SDLC मधील टप्पा / SDLC Phase	रिक्वायरमेंट्स आणि डिझाइन टप्पा	इम्प्लिमेंटेशन आणि टेस्टिंग टप्पा	डिप्लॉयमेंट आणि प्रॉडक्शन टप्पा
कोड एक्झिक्यूशन आवश्यक?	नाही	होय	होय (रिअल-वर्ल्ड वापरात)
मुख्य फोकस / Focus Area	रिक्वायरमेंट्स, डिझाइन, कोड स्ट्रक्चर	फंक्शनॅलिटी, परफॉर्मन्स, सिक्युरिटी	युजर बिहेवियर, फील्ड इश्यूज
उदाहरणे / Examples	रिव्ह्यूज, इन्स्पेक्शन्स, स्टॅटिक कोड ॲनालिसिस	फंक्शनल टेस्टिंग, नॉन-फंक्शनल टेस्टिंग, ऑटोमेटेड टेस्टिंग	यूएटी, युजर फीडबॅक, मॉनिटरिंग, आरसीए
डिफेक्ट्सचा प्रकार / Defect Types	लॉजिक गॅप्स, डिझाइन त्रुटी, कोडिंग स्टँडर्ड उल्लंघन	रनटाइम एरर्स, क्रॅशेस, इनकररेक्ट आउटपुट	प्रॉडक्शन इश्यूज, युजर-रिपोर्टेड डिफेक्ट्स
कॉस्ट इम्पॅक्ट / Cost Impact	कमी (लवकर सापडतात)	मध्यम	जास्त (उशिरा सापडतात)
मुख्य फायदा / Key Advantage	रिवर्क खर्च कमी होतो	प्रत्यक्ष वर्तन तपासता येते	प्रत्यक्ष युजर अनुभव समजतो
मर्यादा / Limitation	रनटाइम इश्यूज सापडत नाहीत	सर्व सिनेरिओज कव्हर करणे कठीण	डिफेक्ट्स उशिरा सापडतात
एकूण प्रभाव / Overall Impact	क्वालिटी फाउंडेशन मजबूत करते	सॉफ्टवेअर स्टॅबिलिटी वाढवते	कस्टमर सॅटिस्फॅक्शन आणि कंटिन्युअस इम्प्रूव्हमेंट सुनिश्चित करते

4.3.3 डिफेक्ट रिपोर्टिंग (Reporting a Defect)

डिफेक्ट रिपोर्टिंग (Defect Reporting) म्हणजे सॉफ्टवेअर टेस्टिंगदरम्यान आढळलेला कोणताही इश्यू, बग किंवा एरर योग्य प्रकारे डॉक्युमेंट करून संबंधित टीमपर्यंत पोहोचवण्याची प्रक्रिया होय, जेणेकरून तो ट्रॅक, असाइन, फिक्स आणि व्हेरिफाय करता येईल. योग्य प्रकारे लिहिलेला डिफेक्ट रिपोर्ट डेव्हलपमेंट टीमला समस्या स्पष्टपणे समजण्यास मदत

करतो आणि ती कार्यक्षमतेने दुरुस्त करता येते. सॉफ्टवेअरचे वर्तन जर रिक्वायरमेंट्स, डिझाइन डॉक्युमेंट्स किंवा युजर एक्स्पेक्टेशन्स यांपासून वेगळे असेल, तर तो डिफेक्ट म्हणून रिपोर्ट केला जातो. उदा. योग्य क्रेडेन्शियल्स वापरून लॉग-इन केल्यावरही सिस्टिमने लॉग-इन नाकारले, तर तो रिपोर्ट करण्याजोगा डिफेक्ट आहे. फक्त फंक्शनॅलिटीमध्येच नाही, तर नॉन-फंक्शनल एरियाज जसे की परफॉर्मन्स, युजेबिलिटी आणि सिक्युरिटी यामध्ये समस्या आढळल्यासही डिफेक्ट रिपोर्ट केला जातो. उदा. पेज लोड होण्यास जास्त वेळ लागणे, जड लोडमध्ये ॲप्लिकेशन क्रॅश होणे किंवा सेन्सिटिव्ह डेटा उघडा पडणे. डॉक्युमेंटेशनमध्ये असलेली अस्पष्टता, मिसिंग रिक्वायरमेंट्स किंवा इनकन्सिस्टन्सीज देखील रिपोर्ट करणे महत्त्वाचे आहे, कारण त्यातून पुढे कोडिंग एरर्स निर्माण होऊ शकतात. थोडक्यात, सॉफ्टवेअरने अपेक्षित फंक्शनॅलिटी, परफॉर्मन्स, सिक्युरिटी किंवा युजर एक्स्पिरियन्स (युजर एक्स्पिरियन्स) पूर्ण केला नाही, तर डिफेक्ट रिपोर्ट करणे आवश्यक आहे. लवकर आणि अचूक डिफेक्ट रिपोर्टिंग मुळे खर्च कमी होतो आणि सॉफ्टवेअर क्वालिटी वाढते.

डिफेक्ट रिपोर्टिंगमधील प्रमुख टप्पे (Key Steps in Reporting a Defect)

1. **डिफेक्ट ओळख / Identification of the Defect):** टेस्टर खात्री करतो की आढळलेली समस्या ही फंक्शन समजण्यात झालेली चूक नसून प्रत्यक्ष एक्स्पेक्टेड बिहेवियर पासूनची तफावत आहे.
2. **डिफेक्ट लॉग करणे / Logging the Defect):** डिफेक्ट जिरा, बगझिला किंवा क्वालिटी सेंटर सारख्या टूलमध्ये नोंदवला जातो. त्यामध्ये डिफेक्ट आयडी, प्रोजेक्ट, प्रॉडक्ट, रिलीज व्हर्जन आणि मॉड्यूलची माहिती दिली जाते.
3. **सारांश आणि वर्णन (Summary and Description)**
   - समरी: एका ओळीत समस्या स्पष्ट करणारे वाक्य
   - डिस्क्रिप्शन: समस्येचे सविस्तर वर्णन, त्याचा परिणाम आणि व्याप्ती
4. **रिप्रोड्यूस करण्याच्या पायऱ्या (Steps to Reproduce):** क्रमवार आणि स्पष्ट स्टेप्स दिल्या जातात, जेणेकरून डेव्हलपर डिफेक्ट पुन्हा निर्माण करू शकेल.
5. **अपेक्षित निकाल विरुद्ध प्रत्यक्ष निकाल (Expected vs Actual Result)**
   - ॲक्च्युअल रिजल्ट: सध्या दिसणारे चुकीचे वर्तन
   - एक्स्पेक्टेड रिजल्ट: स्पेसिफिकेशननुसार अपेक्षित वर्तन
6. **पुरावे आणि ॲटॅचमेंट्स (Attachments and Evidence):** स्क्रीनशॉट्स, लॉग्स किंवा व्हिडिओज जोडले जातात.
7. **सिव्हेरिटी आणि प्रायोरिटी (Severity and Priority)**
   - सिव्हेरिटी: तांत्रिक गंभीरता (उदा. सिस्टिम क्रॅश = क्रिटिकल)
   - प्रायोरिटी: बिझनेस दृष्टीने दुरुस्तीची तातडी
8. **जबाबदारी निश्चित करणे (Assigning Ownership)**
   - रिपोर्टेड बाय: डिफेक्ट लॉग करणारा
   - असाइन्ड टू: डिफेक्ट फिक्स करणारी व्यक्ती
9. **स्टेटस ट्रॅक करणे (Tracking Status)** डिफेक्ट विविध स्टेटसमधून जातो: न्यू → असाइन्ड → ओपन → फिक्स्ड → रिटेस्ट → व्हेरिफाइड → क्लोज्ड → रिओपन / डिफर्ड / रिजेक्टेड

उदाहरण (Example)

परिस्थिती(Scenario): एका युजरने एटीएम सिम्युलेटर मध्ये ₹5000 काढण्याचा प्रयत्न केला, पण डिनॉमिनेशन लिमिटेशन मुळे जास्तीत जास्त ₹3000 इतकीच रक्कम काढता येते.

प्रक्रिया (Process): हा व्यवहार बिझनेस रूल्स नुसार चुकीचा आहे. टेस्टर डिफेक्ट लॉग करतो – योग्य समरी, डिस्क्रिप्शन, स्टेप्स, ॲक्च्युअल रिजल्ट, एक्स्पेक्टेड रिजल्ट आणि स्क्रीनशॉट्स सह, सिव्हेरिटी = हाय, प्रायोरिटी = हाय सेट केली जाते. डिफेक्ट डेव्हलपरकडे असाइन केला जातो आणि लाईफ सायकलनुसार ट्रॅक केला जातो

Field (फील्ड)	Details (तपशील)
ID (आयडी)	Def_02
Project (प्रोजेक्ट)	ATM Simulator
Product (प्रॉडक्ट)	Cash Simulator ATM
Release Version	v1.0

(रिलीज व्हर्जन)	
Module (मॉड्यूल)	Cash Withdrawal → Denomination Selection
Detected Build Version (डिटेक्टेड बिल्ड व्हर्जन)	v1.1
Summary (समरी)	ATM मध्ये मर्यादित डिनॉमिनेशन पर्यायांमुळे ₹3000 पेक्षा जास्त रक्कम काढता येत नाही.
Description (डिस्क्रिप्शन)	Cash Withdrawal मॉड्यूलमध्ये फक्त ₹3000 पर्यंतच निश्चित डिनॉमिनेशन्स उपलब्ध आहेत. युजर्सना ₹5000 सारखी जास्त रक्कम काढता येत नाही, जे प्रत्यक्ष ATM फंक्शनॅलिटीशी सुसंगत नाही.
Steps to Reproduce (रिप्रोड्यूस करण्याच्या पायऱ्या)	1. ATM Simulator उघडा.2. Our Programs निवडा.3. Digital Inclusion Tools → Cash Machine Simulator (ATM) येथे जा.4. भाषा निवडा आणि सिम्युलेटरमध्ये प्रवेश करा.5. कार्ड इन्सर्ट करा आणि Account Type निवडा.6. Other Functions → Cash Withdrawal निवडा.7. ₹5000 रक्कम काढण्याचा प्रयत्न करा.

निरीक्षण (Observation)

Field (फील्ड)	Details (तपशील)
Actual Result (ॲक्च्युअल रिजल्ट)	सिस्टिम फक्त निश्चित डिनॉमिनेशन्स मुळे जास्तीत जास्त ₹3000 इतकीच कॅश विथड्रॉवल परवानगी देते.
Expected Result (एक्स्पेक्टेड रिजल्ट)	खात्यात पुरेसा बॅलन्स असल्यास सिस्टिमने ₹3000 पेक्षा जास्त रक्कम काढण्याची परवानगी द्यावी—अधिक डिनॉमिनेशन्स देऊन किंवा युजर-इनपुट अमाउंट स्वीकारून.
Attachments (ॲटॅचमेंट्स)	None (काहीही नाही)
Remarks (रिमार्क्स)	हा डिफेक्ट सिम्युलेटर युजेबिलिटी मर्यादित करतो आणि युजर इनकन्विनियन्स निर्माण करतो. रिअल ATM यूसेज शी सुसंगत नाही.
Defect Severity (डिफेक्ट सिव्हेरिटी)	High (हाय)
Defect Priority (डिफेक्ट प्रायोरिटी)	High (हाय)
Reported By (रिपोर्टेड बाय)	abc (QA टेस्टर)
Assigned To (असाइन्ड टू)	xyz (डेव्हलपर)
Status (स्टेटस)	Assigned (असाइन्ड)

References

1. Desikan, S., & Ramesh, G. (2016). *Software Testing: Principles and Practices*. Pearson India. ISBN: 9788177581218.
2. Limaye, M. G. (2012). *Software Testing: Principles, Techniques and Tools*. Tata McGraw Hill Education. ISBN: 9780070139909.
3. Chauhan, N. (2016). *Software Testing: Principles and Practices*. Oxford University Press. ISBN: 9780198061847.
4. NPTEL – *Software Testing Course*. Available at: https://nptel.ac.in/courses/106101163
5. GeeksforGeeks – *Sanity vs Smoke Testing*. Available at: https://www.geeksforgeeks.org/difference-between-sanity-testing-and-smoke-testing/
6. W3Schools – *Software Testing Tutorials*. Available at: https://www.w3schools.in/software-testing/tutorials/

युनिट-5
टेस्टिंग टूल्स ॲण्ड मेजरमेंट्स
(Testing Tools and Measurements)

विषय निष्पत्ती (Course Outcome):

CO5: सॉफ्टवेअर टेस्ट करण्यासाठी ऑटोमेशन टेस्टिंग टूल्स वापरा.

घटक निष्पत्ती (Theory Learning Outcome):

1. दिलेल्या ॲप्लिकेशन ची टेस्ट करण्यासाठी वेगवेगळे टेस्टिंग टूल्स ओळखा.
2. दिलेल्या ॲप्लिकेशन साठी ऑटोमेटेड टूल वापरून टेस्टिंग एफिशियन्सी सुधार करा.
3. दिलेल्या ॲप्लिकेशन ची टेस्ट करण्यासाठी टेस्टिंग टूल लागू करा.
4. दिलेल्या ॲप्लिकेशन साठी मेट्रिक्स आणि मेजरमेंट वर्णन करा.

5.1 मॅन्युअल टेस्टिंग विरुद्ध ऑटोमेशन टेस्टिंग, टेस्टिंग टूल्स वापरण्याचे फायदे व तोटे (Manual Testing Verses Automation Testing, Advantages and Disadvantages of Using Testing Tools)

सॉफ्टवेअर टेस्टिंग टूल्स (Software Testing Tools) सॉफ्टवेअर प्रॉडक्ट्सची क्वालिटी (Quality) आणि रिलायबिलिटी (Reliability) सुनिश्चित करण्यात अत्यंत महत्त्वाची भूमिका बजावतात. ही टूल्स मोठ्या प्रमाणावर खालील प्रकारांमध्ये वर्गीकृत केली जातात:

1. टेस्ट मॅनेजमेंट टूल्स (Test Management Tools)
2. डिफेक्ट ट्रॅकिंग टूल्स (Defect Tracking Tools)
3. ऑटोमेशन टूल्स (Automation Tools)
4. परफॉर्मन्स टेस्टिंग टूल्स (Performance Testing Tools)
5. कव्हरेज मेजरमेंट टूल्स (Coverage Measurement Tools)

टेस्ट मॅनेजमेंट टूल्स (Test Management Tools) टेस्टिंग ॲक्टिव्हिटीजचे प्लॅनिंग (Planning), ऑर्गनायझिंग (Organizing) आणि रिपोर्टिंग (Reporting) करण्यात मदत करतात. यामुळे टेस्टिंग प्रक्रिया व्यवस्थित आणि नियंत्रित पद्धतीने राबवता येते. डिफेक्ट ट्रॅकिंग टूल्स (Defect Tracking Tools) सॉफ्टवेअरमधील डिफेक्ट्स (Defects) लॉग करणे, त्यांचा स्टेटस (Status) मॉनिटर करणे आणि योग्य वेळी रिझोल्यूशन (Resolution) करणे सोपे करतात. ऑटोमेशन टूल्स (Automation Tools) पुनरावृत्ती होणाऱ्या टेस्ट केसेस (Test Cases) आपोआप एक्झिक्यूट करतात, ज्यामुळे मॅन्युअल एफर्ट (Manual Effort) कमी होतो आणि टेस्टिंग जलद व सुसंगत होते. यामुळे जलद निकाल (Faster Results) आणि सुसंगत निकाल (Consistent Results) मिळतात. परफॉर्मन्स टेस्टिंग टूल्स (Performance Testing Tools) वेगवेगळ्या लोड (Load) आणि स्ट्रेस कंडिशन्स (Stress Conditions) मध्ये सिस्टिमचे बिहेवियर (Behavior) तपासतात, उदा. रिस्पॉन्स टाइम (Response Time), थ्रूपुट (Throughput) आणि स्केलेबिलिटी (Scalability). कव्हरेज टूल्स (Coverage Tools) टेस्टिंगची व्याप्ती (Extent) आणि पूर्णता (Completeness) मोजण्यास मदत करतात, म्हणजेच सॉफ्टवेअरमधील किती फंक्शनॅलिटी (Functionality) टेस्ट झाली आहे हे स्पष्टपणे कळते. या सर्व टेस्टिंग टूल्स (Testing Tools) चा एकत्रित वापर केल्यामुळे टेस्टिंग प्रोसेसमधील एफिशियन्सी (Efficiency), अचूकता (Accuracy) आणि उत्पादकता (Productivity) लक्षणीयरीत्या सुधारते.

5.1.1 मॅन्युअल टेस्टिंग विरुद्ध ऑटोमेशन टेस्टिंग (Manual Testing Verses Automation Testing)

1. मॅन्युअल टेस्टिंग (Manual Testing)

मॅन्युअल टेस्टिंग (Manual Testing) ही सॉफ्टवेअर टेस्टिंग (Software Testing) ची एक पद्धत आहे, ज्यामध्ये टेस्ट केसेस (Test Cases) टेस्टर्सकडून हाताने (मॅन्युअली) एक्झिक्यूट केल्या जातात आणि ऑटोमेशन टूल्स (Automation Tools) चा वापर केला जात नाही. या पद्धतीमुळे सॉफ्टवेअर ॲप्लिकेशनमधील डिफेक्ट्स (Defects) ओळखता येतात. मॅन्युअल टेस्टिंग ही एक मूलभूत (Fundamental) टेस्टिंग पद्धत आहे. या प्रक्रियेत टेस्टर्स (Testers) हे एंड-युजर्स (End Users) प्रमाणे ॲप्लिकेशनशी थेट संवाद साधतात, सिस्टिमचे बिहेवियर (Behavior) निरीक्षण करतात आणि ऍक्च्युअल आउटकम्स (Actual Outcomes) व एक्स्पेक्टेड रिजल्ट्स (Expected Results) यांची तुलना करून डिफेक्ट्स

शोधतात. ही पद्धत विशेषतः एक्सप्लोरेटरी टेस्टिंग (Exploratory Testing), युजेबिलिटी इव्हॅल्युएशन (Usability Evaluation) आणि ज्या परिस्थितींमध्ये मानवी निर्णय (Human Judgment) आवश्यक असतो अशा ठिकाणी अतिशय उपयुक्त ठरते. कारण यात टेस्टर्सना लवचिकता (Flexibility) आणि ॲडॅप्टेबिलिटी (Adaptability) मिळते, ज्यामुळे ऑटोमेटेड स्क्रिप्ट्स (Automated Scripts) कडून दुर्लक्षित राहू शकणाऱ्या समस्या शोधता येतात.तथापि, मॅन्युअल टेस्टिंग वेळखाऊ (Time-consuming) असू शकते, चुका होण्याची शक्यता (Error-prone) जास्त असते आणि पुनरावृत्ती होणाऱ्या (Repetitive) किंवा मोठ्या प्रमाणावरील टेस्ट केसेस (Large-scale Test Cases) साठी कमी कार्यक्षम ठरते.या मर्यादा असूनही, युजेबिलिटी इश्यूज (Usability Issues) आणि युजर एक्स्पिरियन्स प्रॉब्लेम्स (User Experience Problems) ओळखण्यात प्रभावी असल्यामुळे मॅन्युअल टेस्टिंग ही इंडस्ट्रीतील सर्वात जुनी आणि सर्वाधिक वापरली जाणारी टेस्टिंग पद्धत आहे.

मॅन्युअल टेस्टिंगचे फायदे (Advantages of Manual Testing)

1. एक्सप्लोरेटरी टेस्टिंगसाठी सर्वोत्तम: मानवी टेस्टर्स सर्जनशील विचार करू शकतात, अनपेक्षित सिनेरिओज तपासू शकतात आणि लपलेले बग्स शोधू शकतात.
   उदाहरण: एखादा टेस्टर युजरनेम फील्ड मध्ये इमोजी वापरून लॉग-इन करण्याचा प्रयत्न करू शकतो, असा विचार ऑटोमेशन स्क्रिप्ट करणार नाही.
2. युजेबिलिटी आणि यूआय टेस्टिंगसाठी अधिक योग्य: युजर एक्स्पिरियन्स, लेआउट कलर स्कीम आणि नेव्हिगेशन यांचे मूल्यांकन माणूस टूल्सपेक्षा अधिक चांगल्या प्रकारे करू शकतो.
   उदाहरण: शॉपिंग ॲपमध्ये चेकआउट बटण गोंधळात टाकणाऱ्या ठिकाणी आहे हे ओळखणे.
3. टेस्ट केस एक्झिक्युशनमध्ये लवचिकता: रिक्वायरमेंट्स बदलल्यास टेस्टर्स लगेच टेस्ट केसेस ॲडजस्ट करू शकतात; स्क्रिप्ट पुन्हा लिहिण्याची गरज नसते.
   उदाहरण: लॉग-इन प्रोसेस 2-स्टेपवरून 3-स्टेप झाली, तर टेस्टर त्वरित बदल करू शकतो; पण ऑटोमेशन स्क्रिप्ट्स अपडेट कराव्या लागतात.
4. मोठ्या गुंतवणुकीची गरज नाही: सुरुवातीला महागडी ऑटोमेशन टूल्स किंवा फ्रेमवर्क्स आवश्यक नसतात.
   उदाहरण: नवीन फूड डिलिव्हरी ॲप टेस्ट करणारे स्टार्टअप (स्टार्टअप) मॅन्युअल टेस्टिंगवर अवलंबून राहू शकते, सेलेनियम किंवा क्यूटीपी मध्ये गुंतवणूक न करता.
5. लहान किंवा अल्पकालीन प्रोजेक्टसाठी प्रभावी: कमी फीचर्स असलेल्या ॲप्लिकेशन्ससाठी मॅन्युअल टेस्टिंग अधिक किफायतशीर आणि जलद असते.
   उदाहरण: फक्त 10 फंक्शन्स असलेले कॅल्क्युलेटर ॲप.
6. व्हिज्युअल समस्या सहज ओळखता येतात: मिसअलाईन्ड बटण्स, ब्रोकन लेआउट्स किंवा विसंगत फॉन्ट्स मानवी डोळ्यांनी सहज दिसून येतात.
   उदाहरण: वेबपेजवर कॉन्टॅक्ट अस लिंक एखाद्या इमेजवर ओव्हरलॅप होणे.
7. तात्काळ फीडबॅक: टेस्टर्स थेट डेव्हलपर्स शी रिअल-टाइममध्ये संवाद साधू शकतात.
   उदाहरण: ॲजाईल स्प्रिंट्स दरम्यान डेली बिल्ड मध्ये सापडलेले बग्स त्वरित रिपोर्ट करणे.
8. एकदाच वापरल्या जाणाऱ्या टेस्ट केसेससाठी योग्य: क्वचित चालणाऱ्या टेस्ट्ससाठी ऑटोमेशन स्क्रिप्ट्स लिहिण्यापेक्षा मॅन्युअल टेस्टिंग जलद ठरते.
   उदाहरण: एखाद्या सणासाठी दिलेला प्रमोशनल डिस्काउंट फीचर टेस्ट करणे.

मॅन्युअल टेस्टिंगच्या मर्यादा / तोटे (Disadvantages / Limitations of Manual Testing)

1. धीमे आणि खर्चिक असते: माणसे टेस्ट केसेस स्टेप-बाय-स्टेप चालवतात, त्यामुळे ऑटोमेशन पेक्षा जास्त वेळ लागतो.
   उदाहरण: बँकिंग ॲपमधील लॉग-इन फीचर साठी 100 टेस्ट केसेस मॅन्युअली चालवायला तास लागतात, तर ऑटोमेशन काही मिनिटांत पूर्ण करू शकते.
2. जास्त श्रम लागतात आणि टेस्टिंगला जास्त वेळ लागतो: प्रत्येक बिल्ड किंवा सॉफ्टवेअर अपडेट नंतर तीच टेस्ट केसेस पुन्हा चालवाव्या लागतात.

उदाहरण: ई-कॉमर्स साइटमध्ये नवीन कूपन फीचर आल्यावर रिग्रेशन टेस्टिंग करताना अनेक टेस्ट केसेस पुन्हा चालवाव्या लागतात.

3. स्केलेबल नसते: सॉफ्टवेअर जसे जसे कॉम्प्लेक्स (कॉम्प्लेक्स) होते, तसे टेस्ट केसेसचे प्रमाण वाढते आणि मॅन्युअल मॅनेजमेंट कठीण होते.
   उदाहरण: हेल्थकेअर सिस्टिम (हेल्थकेअर सिस्टिम) मध्ये 100+ मॉड्यूल्स असल्यास हजारो टेस्ट केसेस मॅन्युअली चालवणे अव्यवहार्य ठरते.
4. कन्सिस्टन्सी आणि रिपीटेबिलिटी नसते: वेगवेगळे टेस्टर्स एकाच टेस्ट केस वेगवेगळ्या प्रकारे चालवू शकतात.
   उदाहरण: एक टेस्टर फॉर्म व्हॅलिडेशन नंबरने तपासतो, तर दुसरा सिम्बॉल्स वापरतो निकाल वेगवेगळे येतात.
5. ट्रेनिंगचा अभाव ही मोठी समस्या आहे: कमी अनुभव असलेले टेस्टर्स महत्त्वाचे बग्स चुकवू शकतात.
   उदाहरण: नवीन टेस्टर युजरनेम फील्ड मध्ये स्पेशल कॅरॅक्टर्स टाकून तपासायचे विसरू शकतो.
6. जीयूआय ऑब्जेक्ट साइज आणि कलर डिफरन्सेस ओळखणे कठीण: सूक्ष्म यूजर इंटरफेस (यूजर इंटरफेस) समस्या लक्षात न येऊ शकतात.
   उदाहरण: सबमिट बटण 2 पिक्सेल्सने लहान असणे किंवा निळ्या रंगाची किंचित वेगळी शेड सहज लक्षात येत नाही.
7. मोठ्या किंवा टाइम-बाउंड प्रोजेक्टसाठी योग्य नाही: जलद डिलिव्हरी आणि वारंवार रिलीज असलेल्या प्रोजेक्टसाठी मॅन्युअल टेस्टिंग पुरेशी ठरत नाही.
   उदाहरण: फेसबुक किंवा इन्स्टाग्राम सारख्या प्लॅटफॉर्मसाठी मॅन्युअल टेस्टिंग अव्यवहार्य आहे.
8. बॅच टेस्टिंग शक्य नाही: प्रत्येक टेस्टसाठी मानवी हस्तक्षेप आवश्यक असल्याने टेस्ट्स एकत्र चालवता येत नाहीत.
   उदाहरण:1000 युजर्ससाठी लॉग-इन टेस्ट्स मॅन्युअली एकाच वेळी चालवणे शक्य नाही.
9. मोठ्या प्रमाणात डेटा तुलना करणे अव्यवहार्य: मोठ्या डेटासेट्स ची मॅन्युअल तुलना वेळखाऊ आणि चुका होण्याची शक्यता असते.
   उदाहरण: ईआरपी सिस्टिम मधील 50,000 ट्रान्झॅक्शन्सचा सेल्स रिपोर्ट तपासणे मॅन्युअली जवळजवळ अशक्य आहे.
10. मेंटेनन्स दरम्यान बदल हाताळायला जास्त वेळ लागतो: प्रत्येक चेंज रिक्वेस्ट नंतर संबंधित सर्व टेस्ट्स पुन्हा चालवाव्या लागतात.
    उदाहरण: हॉस्पिटल मॅनेजमेंट सिस्टिम मध्ये बिलिंग लॉजिक बदलल्यास इनव्हॉइस, पेमेंट्स आणि रिफंड्सशी संबंधित सर्व टेस्ट केसेस पुन्हा कराव्या लागतात, ज्याला आठवडे लागू शकतात.

2. ऑटोमेशन टेस्टिंग (Automation Testing)

ऑटोमेशन टेस्टिंग (Automation Testing) म्हणजे ऑटोमेटेड स्क्रिप्ट्स (Automated Scripts) आणि सॉफ्टवेअर टूल्स (Software Tools) वापरून टेस्ट केसेस (Test Cases) आपोआप एक्झिक्यूट करण्याची प्रक्रिया होय. या प्रक्रियेत ऍक्च्युअल रिजल्ट्स (Actual Results) आणि एक्स्पेक्टेड रिजल्ट्स (Expected Results) यांची तुलना केली जाते आणि अत्यल्प मानवी हस्तक्षेपात रिपोर्ट्स (Reports) तयार केले जातात. ऑटोमेशन टेस्टिंग ही ॲडव्हान्स्ड सॉफ्टवेअर टेस्टिंग पद्धत (Advanced Software Testing Approach) आहे, ज्यामध्ये पूर्वनियोजित टेस्ट केसेस ऑटोमेशन टूल्स (Automation Tools), फ्रेमवर्क्स (Frameworks) आणि स्क्रिप्ट्स (Scripts) वापरून स्वयंचलितपणे चालवल्या जातात. मॅन्युअल टेस्टिंग (Manual Testing) मध्ये जिथे टेस्टर्सच्या मानवी प्रयत्नांवर अवलंबून राहावे लागते, तिथे ऑटोमेशन टेस्टिंग मध्ये सॉफ्टवेअरच टेस्ट्स रन करते, निकालांची तुलना करते आणि सविस्तर रिपोर्ट्स तयार करते.

ऑटोमेशन टेस्टिंग विशेषतः खालील प्रकारच्या टेस्टिंगसाठी अतिशय प्रभावी ठरते:

- रिग्रेशन टेस्टिंग (Regression Testing)
- परफॉर्मन्स टेस्टिंग (Performance Testing)
- मोठ्या प्रमाणावर पुनरावृत्ती होणाऱ्या टेस्ट्स (Large-scale Repetitive Test Execution)

जिथे वेग (Speed), सुसंगतता (Consistency) आणि विश्वसनीयता (Reliability) अत्यंत महत्त्वाची असते, तिथे ऑटोमेशन टेस्टिंग मोठा फायदा देते. तसेच, ही पद्धत कंटिन्युअस इंटिग्रेशन / कंटिन्युअस डिलिव्हरी – सीआय / सीडी (CI/CD) पाइपलाईन्सना सपोर्ट करते, ज्यामुळे सॉफ्टवेअर क्वालिटी न घालवता जलद रिलीज सायकल्स (Faster Release Cycles) साध्य करता येतात. तथापि, ऑटोमेशन टेस्टिंगच्या काही मर्यादा देखील आहेत. सुरुवातीला टूल्स

(Tools), फ्रेमवर्क सेटअप (Framework Setup) आणि स्क्रिप्ट डेव्हलपमेंट (Script Development) साठी जास्त खर्च (High Initial Investment) येतो. तसेच, यासाठी कुशल संसाधने (Skilled Resources) आवश्यक असतात. याशिवाय, ज्या ठिकाणी मानवी निर्णय (Human Judgment) आवश्यक असतो—उदा. युजेबिलिटी टेस्टिंग (Usability Testing) किंवा युजर एक्स्पिरियन्स इव्हॅल्युएशन (User Experience Evaluation)—तेथे ऑटोमेशन मर्यादित ठरते. तरीसुद्धा, आधुनिक सॉफ्टवेअर डेव्हलपमेंटमध्ये ऑटोमेशन टेस्टिंग (Automation Testing) हा एक अत्यावश्यक घटक बनला आहे. तो स्केलेबिलिटी (Scalability) वाढवतो, उत्पादकता (Productivity) सुधारतो आणि मोठ्या, वेगाने बदलणाऱ्या प्रोजेक्ट्समध्ये उच्च दर्जाची सॉफ्टवेअर क्वालिटी सुनिश्चित करतो.

ऑटोमेशन टेस्टिंग टूल्सचे फायदे (Advantages of Automation Testing Tools)

1. वेळ वाचतो / वेग वाढतो: ऑटोमेटेड टूल्स मानवी टेस्टर्सपेक्षा टेस्ट केसेस खूप जलद एक्झिक्यूट करतात. मोठ्या रिग्रेशन टेस्ट सूट्स साठी टेस्टिंग वेळ लक्षणीयरीत्या कमी होतो.
   उदाहरण: सेलेनियम रातोरात शेकडो टेस्ट केसेस चालवू शकते, ज्या मॅन्युअली करायला अनेक दिवस लागले असते.
2. टेस्टर्सचा सहभाग कमी होतो: एकदा स्क्रिप्ट्स तयार झाल्यानंतर टेस्ट्स आपोआप रन होतात. त्यामुळे टेस्टर्स एक्सप्लोरेटरी टेस्टिंग, युजेबिलिटी टेस्टिंग आणि क्लिष्ट सिनेरिओजवर लक्ष केंद्रित करू शकतात.
   उदाहरण: रिग्रेशन टेस्ट्स ऑटोमॅटिक चालू असताना टेस्टर्स नवीन फीचर्स टेस्ट करू शकतात.
3. पुनरावृत्तीक्षम आणि सुसंगत: प्रत्येक वेळी टेस्ट्स अगदी त्याच पद्धतीने चालतात, त्यामुळे स्टेप्स चुकण्याची शक्यता नसते. यामुळे अचूक निकाल (Accurate Results) मिळतात.
   उदाहरण: चेकआउट फ्लो साठीचा ऑटोमेशन स्क्रिप्ट नेहमी तीच स्टेप्स फॉलो करतो; माणसाप्रमाणे कूपन विसरत नाही.
4. सिम्युलेटेड टेस्टिंग: टूल्स हजारो व्हर्च्युअल युजर्स, जड लोड आणि मोठे डेटासेट्स सिम्युलेट करू शकतात. यामुळे रिलीजपूर्वी सिस्टिम परफॉर्मन्स तपासता येतो.
   उदाहरण: जे-मीटर फेस्टिव्ह सेल्स दरम्यान फ्लिपकार्ट वर 10,000 युजर्स लॉग-इन करत असल्याचे सिम्युलेट करते.
5. टेस्ट स्क्रिप्ट्स पुन्हा वापरता येतात: एकदाच तयार केलेल्या ऑटोमेटेड टेस्ट्स वेगवेगळ्या व्हर्जन्स साठी वापरता येतात. यामुळे रिवर्क कमी होतो.
   उदाहरण: बँकिंग ॲपचा लॉग-इन टेस्ट व्हर्जन 1.0 साठी तयार केलेला, थोड्या बदलांसह 2.0 मध्येही वापरता येतो.
6. मानवी चुका टाळता येतात: थकवा किंवा दुर्लक्षामुळे होणाऱ्या चुका ऑटोमेशनमध्ये नसतात. त्यामुळे अचूकता(Accuracy) आणि विश्वसनीयता(Reliability) वाढते.
   उदाहरण: पेरोल टेस्टिंग साठीचे ऑटोमेटेड स्क्रिप्ट्स पगार नेहमी अचूक कॅलक्युलेट करतात.
7. जास्त टेस्ट कव्हरेज: ऑटोमेशन टूल्समुळे जास्त संख्येने टेस्ट केसेस डिझाइन आणि एक्झिक्यूट करता येतात. त्यामुळे सॉफ्टवेअरचे अधिक भाग टेस्ट होतात.
   उदाहरण: सेलेनियम सोबत टेस्टएनजी वापरल्यास कोड कव्हरेज रिपोर्ट्स मिळतात.
8. इंटरनल टेस्टिंगला सपोर्ट: मेमरी लीकेज परफॉर्मन्स मॉनिटरिंग आणि कोड कव्हरेज सहज तपासता येते.
   उदाहरण: जॅकोको सारखी टूल्स कोणता कोड टेस्ट झाला आहे हे दाखवतात.
9. दीर्घकाळात खर्च कमी होतो: सुरुवातीचा खर्च जास्त असला तरी, स्क्रिप्ट्स पुन्हा वापरल्यामुळे एकूण प्रोजेक्ट कॉस्ट कमी होते.
   उदाहरण: दर आठवड्याला रिलीज देणाऱ्या प्रॉडक्ट कंपन्या ऑटोमेशनमुळे मनुष्यबळ खर्च वाचवतात.
10. सविस्तर रिपोर्टिंग: बहुतांश ऑटोमेशन टूल्स लॉग्स, रिपोर्ट्स आणि फेल्युअर स्क्रीनशॉट्स तयार करतात. यामुळे डेव्हलपर्सना डिफेक्ट्स पटकन ओळखता येतात.
    उदाहरण: सेलेनियम + टेस्टएनजी इंटिग्रेशनमुळे प्रत्येक रननंतर HTML रिपोर्ट्स मिळतात.

ऑटोमेशन टेस्टिंग टूल्सचे तोटे / मर्यादा (Disadvantages of Automation Testing Tools)

1. उच्च सुरुवातीचा खर्च: ऑटोमेशन सेटअपसाठी टूल्स स्किल्ड रिसोर्सेस आणि इन्फ्रास्ट्रक्चर यामध्ये मोठी गुंतवणूक करावी लागते.

उदाहरण: यूएफटी (UFT) किंवा लोडरनर (LoadRunner) सारखी लायसन्स्ड टूल्स खरेदी करण्यासाठी लाखो रुपये खर्च येऊ शकतो, जो स्टार्टअप्स साठी परवडणारा नसतो.

2. एक्सप्लोरेटरी आणि युजेबिलिटी टेस्टिंगसाठी योग्य नाही: ऑटोमेशन टूल्स लुक अँड फील, डिझाइन किंवा इंट्युटिव्हनेस तपासू शकत नाहीत. युजेबिलिटी (चे मूल्यांकन फक्त माणूसच करू शकतो.
   उदाहरण: एखादे ॲप गोंधळात टाकणारे आहे का किंवा कलर स्कीम त्रासदायक आहे का, हे टूल ठरवू शकत नाही.
3. मेंटेनन्सचा जादा भार: ॲप्लिकेशनमध्ये बदल झाला की संबंधित टेस्ट स्क्रिप्ट्स अपडेट कराव्या लागतात. यामुळे मेंटेनन्सचा वेळ आणि मेहनत वाढते.
   उदाहरण: लॉग-इन बटण चे नाव "लॉगिन" (Login) वरून "साइन इन" (Sign In) झाले, तर त्या बटणाशी संबंधित सर्व स्क्रिप्ट्स बदलाव्या लागतात.
4. तांत्रिक कौशल्य आवश्यक: टेस्टर्सना प्रोग्रामिंग लँग्वेजेस, फ्रेमवर्क्स आणि डिबगिंग शिकावे लागते.
   उदाहरण: सेलेनियम वापरण्यासाठी जावा किंवा पायथन चे ज्ञान आवश्यक असते, जे नॉन-प्रोग्रामर्ससाठी अवघड ठरू शकते.
5. फक्त पूर्वनियोजित सिनेरिओजपुरते मर्यादित: ऑटोमेशन फक्त जे कोडमध्ये लिहिले आहे तेच चालवू शकते; अनपेक्षित परिस्थितींवर विचार करू शकत नाही.
   उदाहरण**:** एखादा अनपेक्षित पॉप-अप आल्यास, तो हँडल न केलेला स्क्रिप्ट फेल होऊ शकतो किंवा बग स्किप करू शकतो.
6. खोट्या सुरक्षिततेची भावना: सर्व ऑटोमेशन स्क्रिप्ट्स पास झाल्या म्हणजे सॉफ्टवेअर पूर्णपणे बग-फ्री आहे असे नाही. टूल्स प्रत्यक्ष युजर समस्यांकडे दुर्लक्ष करू शकतात.
   उदाहरण: स्क्रिप्ट "लॉगिन सक्सेसफुल" (Login Successful) व्हेरिफाय करते, पण डॅशबोर्ड चुकीचा लोड होत आहे हे ओळखत नाही.
7. लहान प्रोजेक्टसाठी खर्चिक ठरते: कमी फीचर्स असलेल्या ॲप्लिकेशन्ससाठी ऑटोमेशनचा फायदा खर्चापेक्षा कमी ठरतो.
   उदाहरण**:**5–6 पेजेस असलेल्या छोट्या वेबसाईटसाठी ऑटोमेशन सेटअप करण्यापेक्षा मॅन्युअल टेस्टिंग योग्य ठरते.
8. एन्व्हायर्नमेंट आणि कम्पॅटिबिलिटी समस्या: ब्राउझर्स, ऑपरेटिंग सिस्टिम्स किंवा थर्ड-पार्टी लायब्ररीज अपडेट झाल्यावर स्क्रिप्ट्स फेल होऊ शकतात.
   उदाहरण: क्रोम (Chrome) व्हर्जन 100 मध्ये चालणारा सेलेनियम टेस्ट, क्रोम व्हर्जन 101 मध्ये फेल होऊ शकतो.

Table 5.1: मॅन्युअल टेस्टिंग वि. ऑटोमेशन टेस्टिंग (Manual Testing Verses Automation Testing)

Criterion (निकष)	**Manual Testing (मॅन्युअल टेस्टिंग)**	**Automation Testing (ऑटोमेशन टेस्टिंग)**
Definition (व्याख्या)	टेस्ट केसेस (टेस्ट केसेस) टेस्टर्सकडून हाताने एक्झिक्यूट केल्या जातात; कोणतेही ऑटोमेशन टूल्स वापरले जात नाहीत.	टेस्ट केसेस ऑटोमेशन टूल्स आणि स्क्रिप्ट्स वापरून आपोआप एक्झिक्यूट केल्या जातात.
Execution (एक्झिक्यूशन)	टेस्ट केसेस मॅन्युअली चालवल्या जातात.	टेस्ट केसेस टूल्स / स्क्रिप्ट्सच्या मदतीने चालवल्या जातात.
Time Requirement (वेळेची गरज)	प्रक्रिया मॅन्युअल असल्यामुळे जास्त वेळ लागतो.	ऑटोमेशनमुळे एक्झिक्यूशन जलद होते.
Initial Investment (सुरुवातीची गुंतवणूक)	सुरुवातीचा खर्च कमी असतो.	सुरुवातीचा खर्च जास्त असतो (टूल्स, सेटअप, स्किल्ड टेस्टर्स).
Accuracy (अचूकता)	मानवी चुका होऊ शकतात, त्यामुळे अचूकता कमी असू शकते.	टूल / स्क्रिप्ट आधारित असल्यामुळे जास्त अचूक आणि विश्वसनीय.

Human Observation (मानवी निरीक्षण)	मानवी निरीक्षण मिळते; युजेबिलिटी (युजेबिलिटी) आणि कस्टमर एक्सिपरियन्स (कस्टमर एक्सिपरियन्स) साठी उपयुक्त.	युजर-फ्रेंडलीनेस किंवा कस्टमर एक्सिपरियन्सचे मूल्यांकन करू शकत नाही.
Suitability (योग्यता)	जवळजवळ सर्व प्रकारच्या सॉफ्टवेअर आणि ॲड-हॉक टेस्टिंग (ॲड-हॉक टेस्टिंग) साठी योग्य.	स्थिर सिस्टिम्स, रिपिटेटिव्ह रिग्रेशन टेस्टिंग (रिपिटेटिव्ह रिग्रेशन टेस्टिंग) आणि परफॉर्मन्स टेस्टिंग (परफॉर्मन्स टेस्टिंग) साठी योग्य.
Flexibility (लवचिकता)	बदल आणि नवीन सिनेरिओजसाठी सहज ॲडॅप्ट करता येते.	रिक्वायरमेंट्स बदलल्यास स्क्रिप्ट्स बदलावी लागतात; लवचिकता कमी.
Example (उदाहरण)	युजरनेम आणि पासवर्ड टाकून लॉग-इन फंक्शनॅलिटी (लॉग-इन फंक्शनॅलिटी) मॅन्युअली तपासणे.	सेलेनियम वेबड्रायव्हर (सेलेनियम वेबड्रायव्हर) वापरून लॉग-इन टेस्ट केसेस ऑटोमेट करणे.

5.2 सेलेक्टिंग अ टेस्ट टूल (Selecting a Test Tool):

टेस्टिंग टूल्स (Testing Tools) ची निवड ही सॉफ्टवेअर टेस्टिंग प्रक्रिया (Software Testing Process) मधील एक अत्यंत महत्त्वाची बाब आहे, कारण टेस्टिंगची प्रभावीता फक्त स्ट्रॅटेजीज (Strategies) आणि तंत्रे (Techniques) यांवरच नाही तर योग्य टूल्स (Tools) च्या निवडीवर देखील अवलंबून असते.

वेगवेगळी टेस्टिंग टूल्स वेगवेगळ्या उद्देशांसाठी वापरली जातात, जसे की:

- टेस्ट मॅनेजमेंट टूल्स (Test Management Tools)
- ऑटोमेशन टूल्स (Automation Tools)
- परफॉर्मन्स टेस्टिंग टूल्स (Performance Testing Tools)
- डिफेक्ट ट्रॅकिंग टूल्स (Defect Tracking Tools)

ही सर्व टूल्स प्रोजेक्ट गरजा (Project Requirements), टेक्नॉलॉजी स्टॅक (Technology Stack), बजेट (Budget) आणि दीर्घकालीन उद्दिष्टे (Long-term Objectives) यांच्याशी जुळणारी आहेत का, हे काळजीपूर्वक तपासणे आवश्यक असते. चुकीच्या टूल निवडी (Tool Selection) मुळे संसाधने (Resources) वाया जाणे, खर्च (Costs) वाढणे आणि उत्पादकता (Productivity) कमी होणे असे दुष्परिणाम होऊ शकतात. याचा थेट परिणाम उत्पादन गुणवत्ता (Product Quality) वर होतो. म्हणूनच, टेस्ट टूल निवडणे हा सॉफ्टवेअर टेस्टिंग लाईफ सायकल – STLC मधील एक धोरणात्मक निर्णय (Strategic Decision) मानला जातो.

एक पद्धतशीर टूल निवड प्रक्रिया (Systematic Tool Selection Process) वापरल्यास निवडलेले टूल:

- बिझनेस उद्दिष्टे (Business Goals)
- तांत्रिक गरजा (Technical Needs)
- टीम क्षमता (Team Capabilities)

यांच्याशी योग्य प्रकारे सुसंगत राहते. यामुळे टेस्टिंगची कार्यक्षमता (Efficiency) वाढते, एकूण खर्च (Overall Cost) कमी होतो आणि शेवटी अधिक उच्च दर्जाचे सॉफ्टवेअर उत्पादन (High-Quality Software Product) तयार होते.

टेस्ट टूल निवडीतील महत्त्वाचे घटक (Important Factors in Tool Selection)

1. **संस्थेच्या परिपक्वतेचे मूल्यांकन (Assessment of the Organization's Maturity):** नवीन टूल स्वीकारण्यापूर्वी संस्थेने बदलांसाठीची तयारी तपासणे आवश्यक असते. यामध्ये टेस्टर्सचे कौशल्य, इन्फ्रास्ट्रक्चर आणि ऑटोमेशन स्वीकारण्याची संस्कृती यांचा समावेश होतो.
   उदाहरण: ऑटोमेशनचा कोणताही अनुभव नसलेल्या स्टार्टअप साठी थेट यूएफटी (UFT) किंवा लोडरनर (LoadRunner) सारखी क्लिष्ट टूल्स स्वीकारणे अवघड ठरू शकते.
2. **टूल सपोर्ट आवश्यक असलेल्या क्षेत्रांची ओळख (Identification of Areas for Tool Support):** टूल कुठे व्हॅल्यू ॲड (Value Add) करेल हे ओळखणे महत्त्वाचे आहे. उदा. टेस्ट एक्झिक्यूशन, डिफेक्ट मॅनेजमेंट, रिग्रेशन टेस्टिंग, परफॉर्मन्स टेस्टिंग किंवा रिपोर्टिंग.

उदाहरण: सर्व काही ऑटोमेट करण्याऐवजी परफॉर्मन्स टेस्टिंग साठी खास जे-मीटर (JMeter) वापरणे.

3. **गरजा आणि निकषांनुसार मूल्यांकन (Evaluation Against Requirements and Criteria):** टूलची तुलना टेक्निकल रिक्वायरमेंट्स (Technical Requirements) आणि बिझनेस रिक्वायरमेंट्स (Business Requirements) यांच्याशी करावी. प्लॅटफॉर्म कम्पॅटिबिलिटी (Platform Compatibility), वापरण्याची सोय (Ease of Use), इंटिग्रेशन क्षमता (Integration Capability) आणि रिपोर्टिंग फीचर्स (Reporting Features).
   उदाहरण: क्रॉस-ब्राउझर टेस्टिंग आवश्यक असल्यास सेलेनियम मल्टिपल ब्राउझर्स सपोर्ट करते का हे तपासणे.
4. **प्रूफ-ऑफ-कॉन्सेप्ट (Proof-of-Concept (PoC)):** प्रत्यक्ष एन्व्हायर्नमेंट मध्ये टूल अपेक्षित उद्दिष्टे पूर्ण करते का हे तपासण्यासाठी ट्रायल इम्प्लिमेंटेशन करणे आवश्यक आहे.
   उदाहरण: मॅन्युअल टेस्टिंग पूर्णपणे सोडण्यापूर्वी सेलेनियम मध्ये छोटा रिग्रेशन टेस्ट सूट रन करून पाहणे.
5. **व्हेंडर किंवा कम्युनिटी सपोर्टचे मूल्यांकन (Evaluation of Vendor or Community Support):** लायसन्स्ड टूल्स साठी व्हेंडर सपोर्ट जसे ट्रेनिंग, अपग्रेड्स आणि मेंटेनन्स खूप महत्त्वाचा असतो. ओपन-सोर्स टूल्स साठी मजबूत कम्युनिटी सपोर्ट आणि डॉक्युमेंटेशन आवश्यक असते.
   उदाहरण: सेलेनियम कडे सक्रिय ओपन-सोर्स कम्युनिटी आहे, तर यूएफटी (UFT) ला मायक्रो फोकस (Micro Focus) कडून अधिकृत व्हेंडर सपोर्ट मिळतो.
6. **अंतर्गत अंमलबजावणीचे नियोजन (Internal Implementation Planning):** टूल डिप्लॉयमेंट साठी योग्य प्लॅन तयार करणे गरजेचे आहे. यामध्ये टेस्टर्सचे ट्रेनिंग, मेंटॉरिंग आणि विद्यमान प्रोसेसमध्ये टूलचे इंटिग्रेश) समाविष्ट असते.
   उदाहरण: प्रत्यक्ष प्रोजेक्टमध्ये वापर सुरू करण्यापूर्वी सेलेनियम वेबड्रायव्हर (Selenium WebDriver) आणि टेस्टएनजी फ्रेमवर्क (TestNG Framework) साठी वर्कशॉप्स आयोजित करणे.

5.2.1 टेस्ट टूल निवडणे: टेस्ट टूल निवडीचे निकष (Criteria for Selecting Test Tools)

फॅक्टर्स (Factors) जे संस्थांना त्यांच्या टेस्टिंग गरजा (Testing Needs) साठी सर्वात योग्य टेस्टिंग टूल (Testing Tool) निवडण्यात मार्गदर्शन करतात. निवडलेले टूल प्रोजेक्ट गरजा (Project Requirements), उपलब्ध तंत्रज्ञान (Available Technology), टीम कौशल्ये (Team Skills) आणि मॅनेजमेंट सपोर्ट (Management Support) यांच्याशी सुसंगत असणे अत्यावश्यक आहे, जेणेकरून टेस्टिंग प्रभावी आणि कार्यक्षम होईल. टेस्ट टूल निवड (Test Tool Selection) हा सॉफ्टवेअर टेस्टिंग लाईफ सायकल – STLC मधील एक अत्यंत महत्त्वाचा निर्णय आहे. आज उपलब्ध असलेल्या असंख्य टूल्स—जसे की ऑटोमेशन टूल्स (Automation Tools), परफॉर्मन्स टेस्टिंग टूल्स (Performance Testing Tools), डिफेक्ट ट्रॅकिंग टूल्स (Defect Tracking Tools) आणि टेस्ट मॅनेजमेंट टूल्स (Test Management Tools) यांमधून योग्य टूल निवडणे गरजेचे असते.योग्य टूल निवडताना ते प्रोजेक्ट उद्दिष्टे (Project Objectives), टेक्नॉलॉजी स्टॅक (Technology Stack), बजेट (Budget) आणि टीम कौशल्ये (Team Skills) यांच्याशी जुळते का हे तपासले पाहिजे.स्पष्टपणे ठरवलेले निकष (Well-defined Criteria) वापरल्यास निवडलेली टूल्स:

- दीर्घकालीन मूल्य (Long-term Value) देतात.
- टेस्टिंग कार्यक्षमता (Testing Efficiency) वाढवतात.
- सॉफ्टवेअर गुणवत्ता (Software Quality) सुधारतात.

म्हणूनच, निकषांवर आधारित टूल निवड (Criteria-based Tool Selection) ही यशस्वी आणि परिणामकारक टेस्टिंगसाठी अत्यावश्यक आहे.

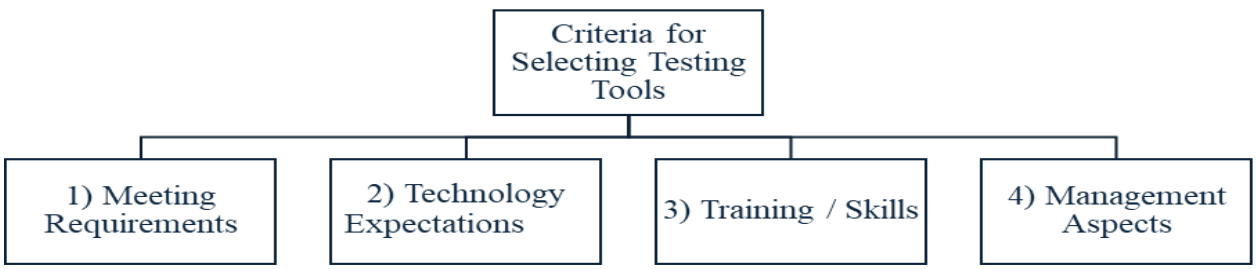

Fig 5.1: क्रायटेरिया फॉर सेलेक्टिंग टेस्ट टूल्स Criteria for Selecting Testing Tools

1. **रिक्वायरमेंट्स पूर्ण करणे (Meeting Requirements):** निवडलेली टूल्स प्रोजेक्ट किंवा संस्थेच्या विशिष्ट गरजा पूर्ण करतात का, याचे मूल्यांकन करणे आवश्यक आहे. सर्व टूल्स प्रत्येक रिक्वायरमेंट कव्हर करत नाहीत; त्यामुळे योग्य रिक्वायरमेंट ॲनालिसिस अत्यंत महत्त्वाचे आहे. काही टूल्स मध्ये प्रॉडक्ट-अंडर-टेस्ट – पीयूटी सोबत बॅकवर्ड कम्पॅटिबिलिटी किंवा फॉरवर्ड नसते. अनेक वेळा टूल्स प्रॉडक्ट फेल्युअर आणि टेस्ट फेल्युअर यामधील फरक ओळखू शकत नाहीत, ज्यामुळे जास्त ॲनालिसिस आणि मॅन्युअल इंटरव्हेन्शन करावे लागते.
   उदाहरण: क्रॉस-ब्राउझर कम्पॅटिबिलिटी आवश्यक असलेल्या वेब ॲप्लिकेशन साठी, जर निवडलेले टूल सर्व आवश्यक ब्राउझर्स सपोर्ट करत नसेल, तर अचूक निकाल मिळत नाहीत. अशा परिस्थितीत सेलेनियम हे डेस्कटॉप-बेस्ड टूल पेक्षा अधिक योग्य ठरते.
2. **टेक्नॉलॉजीविषयक अपेक्षा (Technology Expectations):** टेस्ट टूलने प्रोजेक्टमध्ये वापरल्या जाणाऱ्या टेक्नॉलॉजी स्टॅक ला सपोर्ट करणे आवश्यक आहे, जसे की प्रोग्रामिंग लँग्वेजेस, ऑपरेटिंग सिस्टिम्स, वेब सर्व्हिसेस, डेटाबेसेस, मोबाइल प्लॅटफॉर्म्स. टेस्ट टूल आणि टेक्नॉलॉजी यांच्यात कम्पॅटिबिलिटी नसल्यास टेस्टिंग प्रभावी ठरत नाही.
   उदाहरण: अँड्रॉइड ॲप प्रोजेक्टसाठी सेलेनियम योग्य नाही, कारण ते फक्त वेब ॲप्लिकेशन्स साठी मर्यादित आहे. अशा वेळी ॲपियम किंवा एस्प्रेसो वापरणे अधिक योग्य ठरते.
3. **ट्रेनिंग / स्किल्स (Training / Skills):** टूल वापरण्याची सुलभता ही टेस्टर्सच्या नॉलेज आणि स्किल्स वर अवलंबून असते. ज्या टूल्स साठी क्लिष्ट प्रोग्रामिंग स्किल्स आवश्यक असतात, त्यासाठी ट्रेनिंगचा वेळ आणि खर्च वाढतो. टेस्टर्स योग्य प्रकारे प्रशिक्षित नसतील, तर टूल अनयूज्ड किंवा अंडरयुटिलाइज्ड राहू शकते.
   उदाहरण: यूएफटी – युनिफाइड फंक्शनल टेस्टिंग साठी व्हीबीस्क्रिप्ट चे ज्ञान आवश्यक असते, तर सेलेनियम साठी जावा किंवा पायथन आवश्यक असते. जर टीमकडे मजबूत जावा स्किल् असतील, तर सेलेनियम हा नैसर्गिक पर्याय ठरतो.
4. **मॅनेजमेंटशी संबंधित बाबी (Management Aspects):** टूल लायसन्सिंग कॉस्ट, मेंटेनन्स कॉस्ट, व्हेंडर सपोर्ट आणि अपग्रेड्स यांचा विचार करणे आवश्यक आहे. मॅनेजमेंट ने टूल वापरल्यामुळे मिळणारा आरओआय – रिटर्न ऑन इन्व्हेस्टमेंट देखील तपासला पाहिजे. दीर्घकालीन यशासाठी मजबूत व्हेंडर सपोर्ट किंवा ओपन-सोर्स कम्युनिटी सपोर्ट अत्यंत महत्त्वाचा असतो.
   उदाहरण: यू एफटी हे चांगल्या व्हेंडर सपोर्ट सह येणारे लायसन्स्ड टूल आहे, पण ते खर्चिक आहे. तर सेलेनियम हे ओपन-सोर्स आणि मोफत आहे, पण ते कम्युनिटी सपोर्ट वर अवलंबून असते. त्यामुळे मॅनेजमेंट ने बजेट आणि सपोर्ट नीड्स यांचा समतोल साधणे आवश्यक आहे.

5.2.2 स्टेप्स फॉर टूल सेलेक्शन अँड डिप्लॉयमेंट (Steps for Tool Selection and Deployment)

सॉफ्टवेअर टेस्टिंग टूल (Software Testing Tool) निवडणे आणि ते वापरात आणणे ही सॉफ्टवेअर टेस्टिंग लाईफ सायकल – एसटीएलसी (Software Testing Life Cycle – STLC) मधील एक अत्यंत महत्त्वाची प्रक्रिया आहे. योग्य प्रकारे निवडलेले टूल टेस्ट कार्यक्षमता (Test Efficiency) वाढवते, डिफेक्ट शोध (Defect Detection) सुधारते आणि कंटिन्युअस इंटिग्रेशन (Continuous Integration) ला सपोर्ट करते. याउलट, चुकीच्या प्रकारे निवडलेले टूल खर्च (Cost) वाढवू शकते आणि उत्पादकता (Productivity) कमी करू शकते. त्यामुळे टूल सिलेक्शन आणि डिप्लॉयमेंट ही प्रक्रिया काळजीपूर्वक आणि पद्धतशीर रितीने करणे आवश्यक आहे. ही प्रक्रिया प्रोजेक्ट रिक्वायरमेंट्स (Project Requirements) ओळखण्यापासून सुरू होते आणि योग्य टूल्सची शॉर्टलिस्टिंग (Shortlisting of Tools) केली जाते. त्यानंतर फिजिबिलिटी स्टडी (Feasibility Study) आणि प्रूफ ऑफ कॉन्सेप्ट – पीओसी (Proof of Concept – POC) राबवून प्रत्यक्ष एन्व्हायर्नमेंट (Environment) मध्ये टूल उपयुक्त आहे का हे तपासले जाते. यानंतर कम्पॅरेटिव्ह ॲनालिसिस (Comparative Analysis) करून टूल्सचे मूल्यांकन फंक्शनॅलिटी (Functionality), कॉस्ट (Cost), युजेबिलिटी (Usability) आणि स्केलेबिलिटी (Scalability) या निकषांवर केले जाते. एकदा सर्वात योग्य टूल निश्चित झाले की, डिप्लॉयमेंट फेज (Deployment Phase) सुरू होतो.

डिप्लॉयमेंटमध्ये पुढील बाबींचा समावेश असतो: प्लॅनिंग (Planning), इन्स्टॉलेशन (Installation), कन्फिगरेशन (Configuration), ट्रेनिंग (Training), फुल-स्केल रोलआउट (Full-scale Rollout) शेवटी, कंटिन्युअस इव्हॅल्युएशन (Continuous Evaluation) करून हे सुनिश्चित केले जाते की प्रोजेक्टच्या बदलत्या गरजांनुसार टूल प्रभावी राहते.

स्टेप्स फॉर टूल सेलेक्शन (Steps for Tool Selection)

योग्य टेस्ट टूल निवडण्यासाठी एक सिस्टेमॅटिक इव्हॅल्युएशन प्रोसेस आवश्यक असते. खाली दिलेली पावले सामान्यतः पाळली जातात:

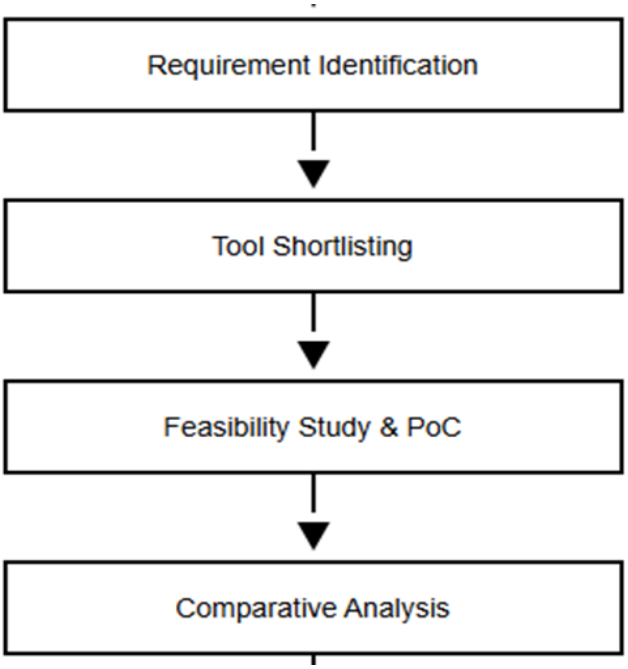

Fig 5.2: टूल सिलेक्शन प्रोसेस (Tool Selection Process)

1. रिक्वायरमेंट आयडेंटिफिकेशन (Requirement Identification): प्रोजेक्टच्या टेस्टिंग नीड्स स्पष्टपणे निश्चित करणे. यामध्ये टेस्टिंगचा प्रकार, सपोर्टेड प्लॅटफॉर्म्स आणि इंटिग्रेशन रिक्वायरमेंट्स यांचा समावेश होतो.
2. टूल शॉर्टलिस्टिंग (Tool Shortlisting): ओळखलेल्या रिक्वायरमेंट्सशी जुळणारी संभाव्य टूल्स निवडणे. यामध्ये कमर्शियल टूल्स आणि ओपन-सोर्स टूल्स दोन्हींचा विचार केला जातो.
3. फिजिबिलिटी स्टडी आणि प्रूफ ऑफ कॉन्सेप्ट – पीओसी (Feasibility Study and Proof of Concept (PoC): शॉर्टलिस्ट केलेल्या टूल्सवर ट्रायल रन करणे. सॅम्पल टेस्ट केसेस वापरून टूल प्रत्यक्ष एन्व्हायर्नमेंट मध्ये उपयुक्त आहे का हे तपासणे.
4. कम्पॅरेटिव्ह ॲनालिसिस (Comparative Analysis): शॉर्टलिस्ट केलेल्या टूल्सची तुलना पुढील पॅरामीटर्सवर करणे: कॉस्ट, युजेबिलिटी, स्केलेबिलिटी, व्हेंडर सपोर्ट, रिपोर्टिंग फीचर्स.
5. फायनल सिलेक्शन (Final Selection): ज्या टूलमधून जास्तीत जास्त बेनेफिट्स मिळतात आणि जे ऑर्गनायझेशनल ऑब्जेक्टिव्ह्स शी सुसंगत आहे, ते टूल निवडणे.

स्टेप्स फॉर टूल डिप्लॉयमेंट (Steps for Tool Deployment)

एकदा योग्य टूल निवडले की, त्याचा प्रभावी वापर सुनिश्चित करण्यासाठी एक स्ट्रक्चर्ड डिप्लॉयमेंट प्रोसेस आवश्यक असते.

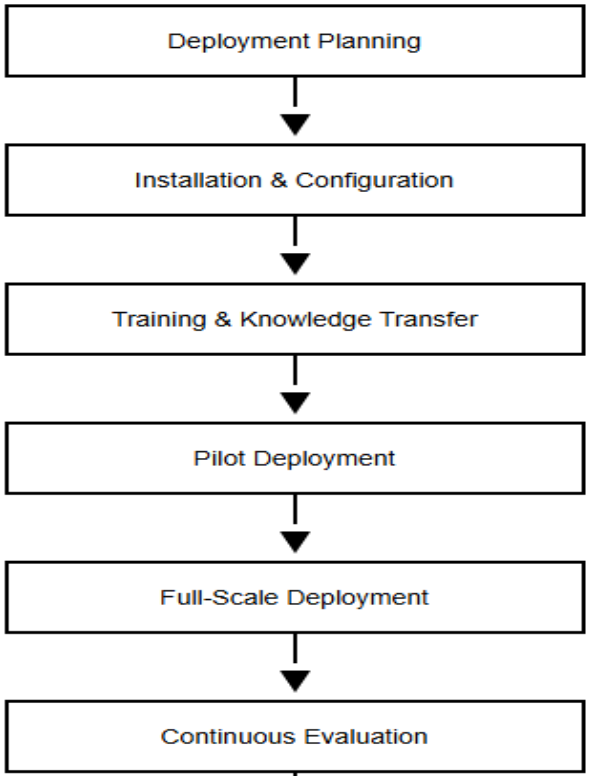

Fig 5.3: टूल डिप्लॉयमेंट प्रोसेस (Tool Deployment Process)

1. **डिप्लॉयमेंट प्लॅनिंग (Deployment Planning):** इन्स्टॉलेशन, कन्फिगरेशन, ट्रेनिंग आणि रोलआउट स्ट्रॅटेजी यांचा समावेश असलेला सविस्तर प्लॅन तयार करणे.
2. **इन्स्टॉलेशन आणि कन्फिगरेशन (Installation and Configuration):** टेस्ट एन्व्हायर्नमेंट मध्ये टूल इन्स्टॉल करणे. टूलला सीआय / सीडी – कंटिन्युअस इंटिग्रेशन / कंटिन्युअस डिप्लॉयमेंट व्हर्जन कंट्रोल आणि डिफेक्ट ट्रॅकिंग सिस्टिम्स सोबत इंटिग्रेट करणे.
3. **ट्रेनिंग आणि नॉलेज ट्रान्सफर (Training and Knowledge Transfer):** टेस्टर्स आणि डेव्हलपर्स साठी ट्रेनिंग सेशन्स आयोजित करणे. योग्य वापरासाठी डॉक्युमेंटेशन उपलब्ध करून देणे.
4. **पायलट डिप्लॉयमेंट (Pilot Deployment):** मर्यादित प्रोजेक्ट / मॉड्यूल वर टूल डिप्लॉय करणे. टूलची इफेक्टिव्हनेस तपासणे आणि सुरुवातीच्या समस्या सोडवणे.
5. **फुल-स्केल डिप्लॉयमेंट (Full-Scale Deployment):** संपूर्ण ऑर्गनायझेशन किंवा सर्व प्रोजेक्ट टीम्स मध्ये टूल रोलआउट करणे.
6. **कंटिन्युअस इव्हॅल्युएशन (Continuous Evaluation):** टूलची परफॉर्मन्स सतत मॉनिटर करणे. येणाऱ्या चॅलेंजेस हाताळणे आणि आवश्यकतेनुसार अपडेट्स / अपग्रेड्स लागू करणे.

उदाहरण (Example)

सिनेरिओ (Scenario): एका बँकिंग ॲप्लिकेशन डेव्हलपमेंट टीमला त्यांच्या वेब-बेस्ड ऑनलाइन बँकिंग सिस्टिम साठी रिग्रेशन टेस्टिंग करण्यासाठी ऑटोमेटेड टेस्टिंग टूल इम्प्लिमेंट करायचे आहे.

टूल सिलेक्शन प्रोसेस (Tool Selection Process)

1. रिक्वायरमेंट आयडेंटिफिकेशन (Requirement Identification): टीमला असे टूल आवश्यक आहे जे वेब ॲप्लिकेशन ऑटोमेशन सपोर्ट करते, जेनकिन्स सीआय / सीडी सोबत इंटिग्रेट होते, आणि डिटेल्ड रिपोर्ट्स जनरेट करते.
2. टूल शॉर्टलिस्टिंग (Tool Shortlisting): सेलेनियम, सायप्रेस आणि यूएफटी – युनिफाइड फंक्शनल टेस्टिंग ही टूल्स शॉर्टलिस्ट केली जातात.

3. फिजिबिलिटी स्टडी आणि प्रूफ ऑफ कॉन्सेप्ट – पीओसी (Feasibility Study and PoC): लॉग-इन आणि फंड ट्रान्सफर मॉड्यूल्सवर तिन्ही टूल्स वापरून टेस्टिंग केली जाते.
   निरीक्षण: सेलेनियम क्रॉस-ब्राउझर टेस्टिंग उत्तमरीत्या हाताळते. सायप्रेस वेगवान आहे पण ब्राउझर सपोर्ट मर्यादित आहे. यूएफटी शक्तिशाली आहे पण खर्चिक आहे.
4. कम्पॅरेटिव्ह ॲनालिसिस (Comparative Analysis): पॅरामीटर्स: कॉस्ट: सेलेनियम – फ्री, सायप्रेस – फ्री, यूएफटी – खर्चिक इंटिग्रेशन: सेलेनियम आणि यूएफटी चांगले, सायप्रेस मर्यादित – युजेबिलिटी: सायप्रेस सोपे, सेलेनियम मध्यम, यूएफटी शिकायला अवघड आहे.
5. फायनल सिलेक्शन (Final Selection): सेलेनियम हे ओपन-सोर्स असणे, वाइड ब्राउझर कम्पॅटिबिलिटी आणि स्ट्रॉंग कम्युनिटी सपोर्ट मुळे निवडले जाते.

टूल डिप्लॉयमेंट प्रोसेस (Tool Deployment Process)

1. **डिप्लॉयमेंट प्लॅनिंग (Deployment Planning):** आयटी मॅनेजर रोल्स, इन्स्टॉलेशन प्लॅन आणि ट्रेनिंग शेड्यूल निश्चित करतो.
2. **इन्स्टॉलेशन आणि कन्फिगरेशन (Installation and Configuration):** सेलेनियम वेबड्रायव्हर इन्स्टॉल केला जातो. तो टेस्टएनजी फ्रेमवर्क सोबत इंटिग्रेट केला जातो. जेनकिन्स सीआय पाईपलाईन शी कनेक्ट केला जातो.
3. **ट्रेनिंग आणि नॉलेज ट्रान्सफर (Training and Knowledge Transfer):** टेस्टर्ससाठी सेलेनियम स्क्रिप्टिंग आणि फ्रेमवर्क वापरावर ट्रेनिंग दिले जाते.
4. **पायलट डिप्लॉयमेंट (Pilot Deployment):** फंड ट्रान्सफर आणि बॅलन्स चेक मॉड्यूल्सवर पायलट रन केला जातो. सिंक्रोनायझेशन इश्यूज. एक्स्प्लिसिट वेट्स वापरून सोडवले जातात.
5. **फुल-स्केल डिप्लॉयमेंट (Full-Scale Deployment):** सेलेनियम फ्रेमवर्क सर्व मॉड्यूल्सवर रोलआउट केला जातो: लॉग-इन, ट्रान्झॅक्शन, रिपोर्ट्स, ॲडमिन.
6. **फुल-स्केल डिप्लॉयमेंट (Continuous Evaluation):** नियमित फीडबॅक सेशन्स घेतले जातात. जलद निकालांसाठी पॅरॅलल टेस्ट एक्झिक्युशन फ्रेमवर्कमध्ये समाविष्ट केले जाते.

निरीक्षण (Observation): फुल डिप्लॉयमेंटनंतर टेस्टिंग टीमने नोंदवले:

- मॅन्युअल रिग्रेशन एफर्ट मध्ये 60% घट.
- सीआय / सीडी पाईपलाईन मधील ऑटोमेशनमुळे जलद रिलीज सायकल्स.
- सर्व ब्राउझर्समध्ये सुसंगत टेस्ट एक्झिक्युशनमुळे रिलायबिलिटी सुधारली.

5.3 सेलेनियम(Selenium)

5.3.1 सेलेनियम: परिचय (Selenium: Introduction)

सेलेनियम (Selenium) हे वेब ॲप्लिकेशन्स (Web Applications) साठी खास डिझाइन केलेले, सर्वाधिक वापरले जाणारे ओपन-सोर्स ऑटोमेशन टेस्टिंग फ्रेमवर्क (Open-Source Automation Testing Framework) आहे. हे टेस्टर्स (Testers) आणि डेव्हलपर्स (Developers) यांना ब्राउझर ॲक्शन्स (Browser Actions) ऑटोमेट करण्यासाठी, यूजर इंटरॅक्शन्स (User Interactions) व्हॅलिडेट करण्यासाठी आणि क्रॉस-ब्राउझर टेस्टिंग (Cross-Browser Testing) प्रभावीपणे करण्यासाठी एक मजबूत प्लॅटफॉर्म प्रदान करते. सेलेनियम (Selenium) अनेक प्रोग्रामिंग लँग्वेजेस (Programming Languages) सपोर्ट करते, जसे की: जावा (Java), पायथन (Python), सी-शार्प (C#), जावास्क्रिप्ट (JavaScript) यामुळे वेगवेगळ्या प्रोजेक्ट रिक्वायरमेंट्स (Project Requirements) साठी सेलेनियम अत्यंत फ्लेक्सिबल (Flexible) ठरते. कमर्शियल ऑटोमेशन टूल्स (Commercial Automation Tools) पेक्षा वेगळे, सेलेनियम (Selenium) हे पूर्णपणे फ्री (Free) आहे आणि त्याला एक मोठा व सक्रिय कम्युनिटी सपोर्ट (Community Support) उपलब्ध आहे. या सक्रिय कम्युनिटीमुळे सेलेनियममध्ये सतत सुधारणा होत राहतात आणि ते ॲजाईल (Agile) व डेव्हऑप्स (DevOps) सारख्या आधुनिक सॉफ्टवेअर इंजिनियरिंग प्रॅक्टिसेस (Software Engineering Practices) सोबत सहजपणे इंटिग्रेट (Integrate) होते. आजच्या काळात कंटिन्युअस इंटिग्रेशन / कंटिन्युअस डिप्लॉयमेंट – सीआय / सीडी (Continuous Integration / Continuous Deployment – CI/CD) पाईपलाईन्समध्ये सेलेनियम महत्त्वाची

भूमिका बजावते. यामुळे संस्थांना फास्टर (Faster) आणि मोअर रिलायबल (More Reliable) सॉफ्टवेअर रिलीजेस साध्य करता येतात.

5.3.2 सेलेनियमचे घटक (Components of Selenium)

सेलेनियम सूट (Selenium Suite) मध्ये एकूण चार प्रमुख कॉम्पोनंट्स (Four Major Components) असतात. हे प्रत्येक कॉम्पोनंट वेब ऑटोमेशन (Web Automation) च्या वेगवेगळ्या गरजांना पूर्ण करतात.

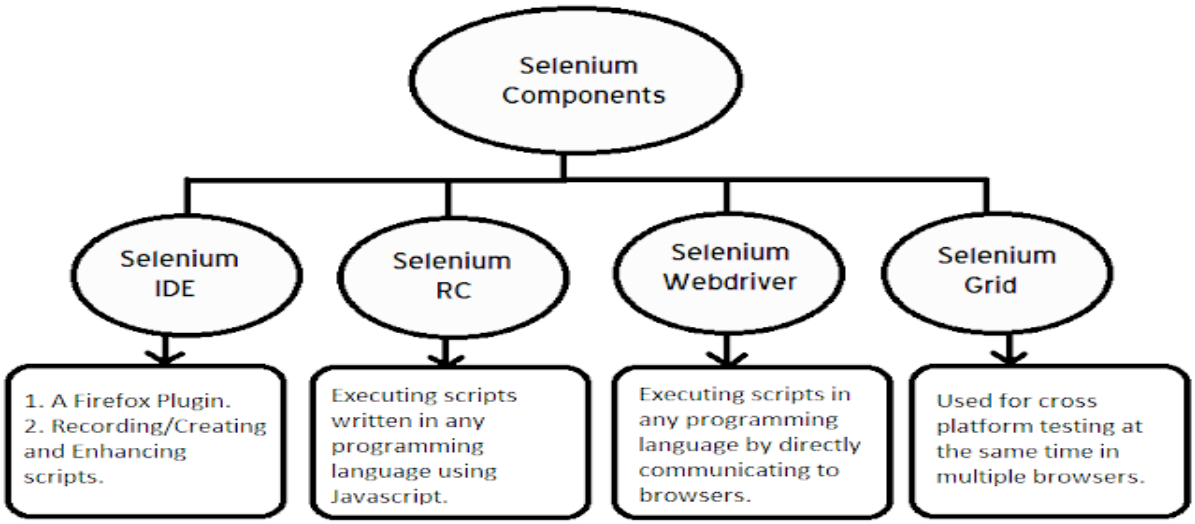

Fig 5.4: कॉम्पोनन्ट्स ऑफ सेलेनियम (Components of Selenium)

1. **सेलेनियम आयडीई** – इंटिग्रेटेड डेव्हलपमेंट एन्व्हायर्नमेंट (Selenium IDE (Integrated Development Environment)): ब्राउझर एक्स्टेन्शन (Browser Extension) आहे, जे क्रोम (Chrome) आणि फायरफॉक्स (Firefox) साठी उपलब्ध आहे. रिकॉर्ड-अँड-प्लेबॅक (Record-and-Playback) पद्धतीने टेस्ट स्क्रिप्ट्स (Test Scripts) तयार करता येतात. बिगिनर्स (Beginners) साठी आणि क्विक टेस्ट प्रोटोटाइप्स (Quick Test Prototypes) तयार करण्यासाठी उपयुक्त आहे. बेसिक डिबगिंग (Basic Debugging) आणि टेस्ट केस मॅनेजमेंट (Test Case Management) सुविधा प्रदान करते.
2. **सेलेनियम आरसी – रिमोट कंट्रोल डिप्रिकेटेड (Selenium RC (Remote Control) – Deprecated):** वेबड्रायव्हर (WebDriver) येण्यापूर्वी वापरला जाणारा जुना सेलेनियम कॉम्पोनंट (Selenium Component) होता. टेस्ट एक्झिक्युशन (Test Execution) साठी स्वतंत्र सर्व्हर इन्स्टन्स (Server Instance) आवश्यक होता. विविध लिमिटेशन्स (Limitations) मुळे हळूहळू वापरातून काढून टाकण्यात आला. सध्या याची जागा पूर्णपणे सेलेनियम वेबड्रायव्हर (Selenium WebDriver) ने घेतली आहे.
3. **सेलेनियम वेबड्रायव्हर (Selenium WebDriver):** सेलेनियममधील सर्वात पॉवरफुल (Powerful) आणि सर्वाधिक वापरला जाणारा कॉम्पोनंट (Component) आहे. नेटिव ब्राउझर ड्रायव्हर्स (Native Browser Drivers) द्वारे थेट ब्राउझर्सशी संवाद साधतो, उदा.: क्रोमड्रायव्हर (ChromeDriver), गेकोड्रायव्हर (GeckoDriver). पुढील ॲडव्हान्स्ड ऑटोमेशन फीचर्स (Advanced Automation Features) सपोर्ट करतो:डायनॅमिक एलिमेंट्स (Dynamic Elements), अजॅक्स कॉल्स (AJAX Calls), मल्टी-टॅब ब्राउझिंग (Multi-Tab Browsing)
   अनेक प्रोग्रामिंग लँग्वेजेस (Programming Languages) आणि टेस्टिंग फ्रेमवर्क्स (Testing Frameworks) सोबत सुसंगत आहे, जसे की: ज्युनिट (JUnit), टेस्टएनजी (TestNG), पायटेस्ट (PyTest)
4. **सेलेनियम ग्रिड (Selenium Grid):** डिस्ट्रिब्युटेड टेस्टिंग (Distributed Testing) आणि पॅरॅलल टेस्टिंग (Parallel Testing) सुलभ करते. एकाच वेळी अनेक मशीन्स (Machines) आणि ब्राउझर्स (Browsers) वर टेस्ट स्क्रिप्ट्स (Test Scripts) एक्झिक्युट करता येतात. टेस्ट एक्झिक्युशन टाइम (Test Execution Time) कमी करते. क्रॉस-प्लॅटफॉर्म कम्पॅटिबिलिटी (Cross-Platform Compatibility) सपोर्ट करते. सीआय / सीडी एन्व्हायर्नमेंट्स (CI/CD Environments) मध्ये एंटरप्राइज-लेव्हल रिग्रेशन टेस्टिंग (Enterprise-Level Regression Testing) साठी अत्यंत महत्त्वाची आहे.

5.3.3 ऑटोमेशन टेस्टिंग टूल्स (Automation Testing Tools)

सेलेनियम (Selenium) व्यतिरिक्त, इंडस्ट्रीमध्ये अनेक इतर ऑटोमेशन टेस्टिंग टूल्स (Automation Testing Tools) वापरली जातात. ही टूल्स मुख्यतः दोन प्रकारांमध्ये विभागली जातात: ओपन-सोर्स टूल्स (Open-Source Tools) आणि कमर्शियल टूल्स (Commercial Tools).

ओपन-सोर्स टूल्स (Open-Source Tools): सेलेनियम (Selenium) – वेब ॲप्लिकेशन ऑटोमेशन (Web Application Automation) साठी सर्वाधिक वापरले जाणारे टूल.

- ॲपियम (Appium) – मोबाइल ॲप्लिकेशन्स (Mobile Applications) साठी टेस्ट ऑटोमेशन करते.
- सायप्रेस (Cypress) – जावास्क्रिप्ट (JavaScript) आधारित फ्रंट-एंड टेस्टिंग (Front-End Testing) वर लक्ष केंद्रित करते.
- ज्युनिट / टेस्टएनजी (JUnit / TestNG) – जावा (Java) साठी युनिट टेस्टिंग (Unit Testing) आणि इंटिग्रेशन टेस्टिंग (Integration Testing) फ्रेमवर्क्स.

कमर्शियल टूल्स (Commercial Tools): एचपी यूएफटी – युनिफाइड फंक्शनल टेस्टिंग (HP UFT – Unified Functional Testing) – एंटरप्राइज-लेव्हल (Enterprise-Level) आणि फीचर-रिच (Feature-Rich) ऑटोमेशन टूल.

- टेस्टकम्प्लिट (TestComplete) – जीयूआय-बेस्ड टेस्ट ऑटोमेशन (GUI-Based Test Automation) सुविधा देते.
- रॅनॉरेक्स (Ranorex) – क्रॉस-प्लॅटफॉर्म टेस्टिंग (Cross-Platform Testing) सपोर्ट करते (डेस्कटॉप, वेब, मोबाइल).
- टोस्का – ट्रायसेंटिस (Tosca – Tricentis) – मॉडेल-बेस्ड ऑटोमेशन टेस्टिंग (Model-Based Automation Testing) टूल.

वरील सर्व टूल्सच्या तुलनेत, सेलेनियम (Selenium) हे वेब ॲप्लिकेशन ऑटोमेशन (Web Application Automation) साठी आजही सर्वाधिक लोकप्रिय पर्याय आहे.

याची कारणे म्हणजे: कॉस्ट-इफेक्टिव्हनेस (Cost-Effectiveness), स्केलेबिलिटी (Scalability), इंटिग्रेशन कॅपॅबिलिटीज (Integration Capabilities)

सेलेनियमचे फायदे (Advantages of Selenium)

1. फ्री (Free) आणि ओपन-सोर्स (Open-Source) आहे.
2. क्रॉस-ब्राउझर (Cross-Browser) आणि क्रॉस-प्लॅटफॉर्म (Cross-Platform) सपोर्ट देते.
3. अनेक प्रोग्रामिंग लँग्वेजेस (Programming Languages) सपोर्ट करते.
4. सीआय / सीडी पाईपलाईन्स (CI/CD Pipelines) सोबत मजबूत इंटिग्रेशन (Integration).
5. मोठा कम्युनिटी सपोर्ट (Community Support) आणि वारंवार अपडेट्स (Updates).

सेलेनियमच्या मर्यादा / तोटे (Disadvantages / Limitations of Selenium)

1. फक्त वेब ॲप्लिकेशन्स (Web Applications) साठी मर्यादित (डेस्कटॉप ॲप टेस्टिंगसाठी सपोर्ट नाही).
2. क्लिष्ट टेस्ट स्क्रिप्ट्स (Test Scripts) साठी प्रोग्रामिंग नॉलेज (Programming Knowledge) आवश्यक.
3. बिल्ट-इन टेस्ट रिपोर्टिंग (Built-in Test Reporting) नाही (यासाठी टेस्टएनजी (TestNG), अल्यूअर (Allure) किंवा इतर थर्ड-पार्टी टूल्स (Third-Party Tools) लागतात).
4. मोठ्या टेस्ट सूट्स (Test Suites) साठी हाय मेंटेनन्स एफर्ट (High Maintenance Effort) लागतो.

5.4 सेलेनियम आयडीई (Selenium IDE)

5.4.1 सेलेनियम आयडीई: परिचय (Selenium IDE: Introduction)

सेलेनियम इंटिग्रेटेड डेव्हलपमेंट एन्व्हायर्नमेंट – आयडीई (Selenium Integrated Development Environment – IDE) हे सेलेनियम सूट (Selenium Suite) मधील सर्वात जुने आणि सर्वात सोपे टूल आहे. हे क्रोम (Chrome) आणि फायरफॉक्स (Firefox) साठी ब्राउझर एक्स्टेन्शन (Browser Extension) म्हणून उपलब्ध आहे आणि रिकॉर्ड-अँड-प्लेबॅक (Record-and-Playback) यंत्रणा वापरून टेस्ट केसेस (Test Cases) तयार करण्याची सुविधा देते. सेलेनियम आयडीई (Selenium IDE) हे मुख्यतः बिगिनर्स (Beginners) आणि अशा टीम्ससाठी तयार करण्यात आले आहे ज्यांना प्रोग्रामिंग नॉलेज (Programming Knowledge) शिवाय पटकन ऑटोमेशन स्क्रिप्ट्स (Automation Scripts) तयार

करायच्या असतात. यामध्ये टेस्टर्स (Testers) वेब ॲप्लिकेशनवर केलेल्या यूजर ॲक्शन्स (User Actions) रेकॉर्ड करू शकतात आणि नंतर त्या प्लेबॅक करून फंक्शनॅलिटी व्हॅलिडेशन (Functionality Validation) करू शकतात. जरी सेलेनियम आयडीई (Selenium IDE) हे सेलेनियम वेबड्रायव्हर (Selenium WebDriver) इतके शक्तिशाली नसले, तरी रॅपिड टेस्ट क्रिएशन (Rapid Test Creation) आणि एक्सप्लोरेटरी टेस्टिंग (Exploratory Testing) साठी हे एक उत्तम एंट्री-लेव्हल टूल (Entry-Level Tool) मानले जाते.

5.4.2 सेलेनियम आयडीईची वैशिष्ट्ये (Features of Selenium IDE)

सेलेनियम आयडीई (Selenium IDE) मध्ये अनेक महत्त्वाची फीचर्स (Features) आहेत, ज्यामुळे ते जलद आणि सोप्या ऑटोमेशन टास्क्स (Automation Tasks) साठी उपयुक्त ठरते.

1. **रिकॉर्ड आणि प्लेबॅक (Record and Playback):** ब्राउझर (Browser) मधील यूजर इंटरॅक्शन्स (User Interactions) कॅप्चर करते. आपोआप टेस्ट स्क्रिप्ट्स (Test Scripts) जनरेट करते. क्विक प्रोटोटाइप्स (Quick Prototypes) आणि फंक्शनल व्हॅलिडेशन टेस्ट्स (Functional Validation Tests) तयार करण्यासाठी उपयुक्त.
2. **क्रॉस-ब्राउझर सपोर्ट (Cross-Browser Support):** क्रोम (Chrome) आणि फायरफॉक्स (Firefox) साठी एक्स्टेन्शन म्हणून उपलब्ध. वेगवेगळ्या एन्व्हायर्नमेंट्स (Environments) मध्ये सुसंगतता सुनिश्चित करते.
3. **कमांड सेट (Command Set):** सेलेनीज कमांड्स (Selenese Commands) चा समृद्ध संच प्रदान करते. यामध्ये खालील ॲक्शन्सचा समावेश होतो:
   - बटण क्लिक करणे (Clicking Buttons)
   - टेक्स्ट एंटर करणे (Entering Text)
   - ड्रॉपडाऊन सिलेक्ट करणे (Selecting Dropdowns)
   - एलिमेंट्स व्हेरिफाय करणे (Verifying Elements)
4. **डिबगिंग आणि एडिटिंग (Debugging and Editing):** टेस्ट स्क्रिप्ट्स (Test Scripts) पॉज (Pause) करणे, स्टेप-बाय-स्टेप (Step-by-Step) रन करणे आणि डिबग (Debug) करणे शक्य. ब्रेकपॉइंट्स (Breakpoints) आणि लॉग मेसेजेस (Log Messages) सारखी फीचर्स उपलब्ध.
5. **डेटा-ड्रिव्हन टेस्टिंग सपोर्ट (Data-Driven Testing Support):** पॅरामीटरायझेशन (Parameterization) सपोर्ट करते. एक्स्टर्नल फायली (External Files) जसे की सीएसव्ही (CSV) मधून डेटा इम्पोर्ट करता येतो.
6. **टेस्ट केसेस एक्सपोर्ट करणे (Exporting Test Cases):** तयार केलेले टेस्ट केसेस (Test Cases) खालील प्रोग्रामिंग लँग्वेजेस (Programming Languages) मध्ये एक्सपोर्ट करता येतात: जावा (Java), पायथन (Python), सी-शार्प (C#), जावास्क्रिप्ट (JavaScript) हे टेस्ट केसेस पुढे सेलेनियम वेबड्रायव्हर (Selenium WebDriver) सोबत वापरता येतात.
7. **वापरण्यास सोपे (Ease of Use):** इंट्युटिव ग्राफिकल इंटरफेस (Intuitive Graphical Interface) उपलब्ध. नॉन-प्रोग्रामर्स (Non-Programmers) आणि बिगिनर्स (Beginners) साठी अतिशय सोपे.

5.4.3 सेलेनियम आयडीईच्या मर्यादा (Limitations of Selenium IDE)

सेलेनियम आयडीई (Selenium IDE) वापरण्यास सोपे आणि युजर-फ्रेंडली असले तरी, लार्ज-स्केल ऑटोमेशन प्रोजेक्ट्स (Large-Scale Automation Projects) साठी वापरताना काही महत्त्वाच्या मर्यादा आढळतात.

1. **रिकॉर्ड-अँड-प्लेबॅकपुरते मर्यादित (Limited to Record-and-Playback):** तयार होणाऱ्या स्क्रिप्ट्स (Scripts) बहुतेक वेळा फ्लेक्सिबल (Flexible) नसतात. कस्टम वेबड्रायव्हर स्क्रिप्ट्स (Custom WebDriver Scripts) च्या तुलनेत त्या मेंटेन (Maintain) करणे कठीण असते.
2. **ब्राउझर सपोर्टची मर्यादा (Browser Support Restriction):** फक्त क्रोम (Chrome) आणि फायरफॉक्स (Firefox) साठी उपलब्ध. सफारी (Safari) किंवा एज (Edge) सारख्या इतर ब्राउझर्सना सपोर्ट नाही.
3. **क्लिष्ट टेस्ट सिनेरिओसाठी अनुपयुक्त (Not Suitable for Complex Test Scenarios):** डायनॅमिक एलिमेंट्स (Dynamic Elements) हाताळण्यासाठी मर्यादित क्षमता. अजॅक्स रिक्वेस्ट्स (AJAX Requests) आणि मल्टी-टॅब ब्राउझिंग (Multi-Tab Browsing) साठी प्रभावी सपोर्ट नाही.

4. **परफॉर्मन्स समस्या (Performance Issues):** मोठ्या रिग्रेशन टेस्ट सूट्स (Regression Test Suites) साठी कार्यक्षम नाही. एंटरप्राइज-लेव्हल प्रोजेक्ट्स (Enterprise-Level Projects) मध्ये परफॉर्मन्स कमी पडतो.
5. **बिल्ट-इन रिपोर्टिंग नाही (No Built-in Reporting):** डिटेल्ड रिपोर्टिंग (Detailed Reporting) किंवा टेस्ट ॲनालिटिक्स (Test Analytics) सुविधा उपलब्ध नाही. यासाठी एक्स्टर्नल इंटिग्रेशन्स (External Integrations) आवश्यक असतात.
6. **मेंटेनन्सशी संबंधित अडचणी (Maintenance Challenges):** रिकॉर्ड-अँड-प्लेबॅक (Record-and-Playback) वापरून तयार केलेल्या टेस्ट्स यूआय चेंजेस (UI Changes) मुळे वारंवार फेल होतात. त्यामुळे हाय मेंटेनन्स एफर्ट (High Maintenance Effort) लागतो.

उदाहरण (Example): सेलेनियम आयडीई वापरून ऑनलाइन नोटपॅडसाठी टेस्ट केसेस लिहिणे आणि रन करणे
सिनेरिओ (Scenario): एका टेस्टर (Tester) ला ऑनलाइन नोटपॅड ॲप्लिकेशन (Online Notepad Application) ची फंक्शनॅलिटी (Functionality) व्हॅलिडेट करण्याचे काम दिले आहे.
रिक्वायरमेंट्स खालीलप्रमाणे आहेत:

- नोटपॅड वेब पेज (Notepad Web Page) लॉन्च करणे.
- टेक्स्ट एंटर करणे (Text Entering).
- कंटेंट सेव्ह करणे (Saving Content).
- सेव्ह केलेला टेक्स्ट आणि इनपुट टेक्स्ट जुळतो का ते व्हेरिफाय करणे (Verify).

प्रक्रिया (Process)

1. सेलेनियम आयडीई (Selenium IDE) हे ब्राउझर एक्स्टेन्शन (Browser Extension) म्हणून लॉन्च करणे.
2. नवीन प्रोजेक्ट (Project) तयार करणे आणि त्याला योग्य नाव देणे, उदा. नोटपॅड_ऑटोमेशन Notepad_Automation).
3. नोटपॅड ॲप्लिकेशन यूआरएल (Notepad Application URL) ओपन करणे (उदा. https://anotepad.com/).
4. ॲक्शन्स रेकॉर्ड करणे (Record Actions):

- टेक्स्ट एरिया (Text Area) मध्ये टेक्स्ट टाइप करणे.
- सेव्ह बटण (Save Button) क्लिक करून नोट सेव्ह करणे.
- सेव्ह केलेली नोट रिट्रीव्ह करणे (Retrieve Saved Note).

5. रेकॉर्ड केलेला टेस्ट प्लेबॅक (Test Playback) करून तपासणे की नोटपॅड टेक्स्ट योग्यरीत्या सेव्ह आणि डिस्प्ले करतो का.
6. आवश्यक असल्यास टेस्ट केस (Test Case) सेलेनियम वेबड्रायव्हर (Selenium WebDriver) सोबत वापरण्यासाठी एक्सपोर्ट (Export) करणे.

निरीक्षण (Observation): सेलेनियम आयडीई (Selenium IDE) यशस्वीरित्या यूजर इंटरॅक्शन्स (User Interactions) जसे की टेक्स्ट टाइप करणे, सेव्ह बटण क्लिक करणे आणि टेक्स्ट व्हेरिफाय करणे रेकॉर्ड करते. प्लेबॅक (Playback) दरम्यान, टेस्ट आपोआप रन होते आणि सेव्ह केलेली नोट योग्यरीत्या दिसते का हे व्हॅलिडेट करते. कोणताही मिसमॅच (Mismatch) आढळल्यास (उदा. चुकीचा सेव्ह झालेला टेक्स्ट), तो सेलेनियम आयडीई रिपोर्ट (Selenium IDE Report) मध्ये टेस्ट फेल्युअर (Test Failure) म्हणून लॉग केला जातो.

टेस्ट केसेस (Test Cases)

Test Case ID	Test Description (टेस्ट डिस्क्रिप्शन)	Steps (स्टेप्स)	Expected Result (अपेक्षित निकाल)	Status (स्टेटस)
TC01	नोटपॅड ॲप्लिकेशन लॉन्च करणे (Launch Notepad Application)	ब्राउझर ओपन करणे → https://anotepad.com/ वर नेव्हिगेट करणे	नोटपॅड होम पेज (Notepad Home Page) डिस्प्ले झाले पाहिजे	पास / फेल
TC02	नोटपॅडमध्ये टेक्स्ट एंटर करणे (Enter Text in Notepad)	टेक्स्ट एरिया (Text Area) मध्ये “Selenium IDE Test” टाइप करणे	टेक्स्ट योग्यरीत्या नोटपॅड एडिटर (Notepad Editor) मध्ये दिसला पाहिजे	पास / फेल
TC03	नोट सेव्ह करणे (Save the Note)	सेव्ह बटण (Save Button) क्लिक करणे	कन्फर्मेशन मेसेज (Confirmation Message) दिसावा आणि नोट सेव्ह झाली पाहिजे	पास / फेल
TC04	सेव्ह केलेली नोट रिट्रीव्ह करणे (Retrieve Saved Note)	सेव्ह नोट्स लिस्ट (Saved Notes List) ओपन करणे → सेव्ह केलेली नोट सिलेक्ट करणे	सेव्ह केलेली नोट “Selenium IDE Test” टेक्स्टसह डिस्प्ले झाली पाहिजे	पास / फेल
TC05	नोट पर्सिस्टन्स व्हॅलिडेट करणे (Validate Note Persistence)	ब्राउझर रिफ्रेश (Browser Refresh) करणे → सेव्ह नोट्स ओपन करणे	नोट कायम (Persist) राहिली पाहिजे आणि तोच कंटेंट दिसला पाहिजे	पास / फेल

5.5 सेलेनियम वेबड्रायव्हर (Selenium WebDriver):

5.5.1 सेलेनियम वेबड्रायव्हर: परिचय (Selenium WebDriver: Introduction)

सेलेनियम वेबड्रायव्हर (Selenium WebDriver) हे एक अत्यंत शक्तिशाली ऑटोमेशन टेस्टिंग टूल (Automation Testing Tool) आहे, जे विविध ब्राउझर्स (Browsers) आणि प्लॅटफॉर्म्स (Platforms) वर वेब ॲप्लिकेशन्स (Web Applications) ची टेस्टिंग करण्यासाठी वापरले जाते. सेलेनियम आयडीई (Selenium IDE) प्रमाणे रिकॉर्ड-अँड-प्लेबॅक (Record-and-Playback) मॉडेलवर अवलंबून न राहता, वेबड्रायव्हर (WebDriver) थेट एपीआयज (APIs) प्रदान करते. या एपीआयजच्या मदतीने ब्राउझर एलिमेंट्स (Browser Elements) शी थेट संवाद साधता येतो, ज्यामुळे टेस्टर्सना अधिक नियंत्रण आणि लवचिकता मिळते. सेलेनियम वेबड्रायव्हर (Selenium WebDriver) वापरून टेस्ट स्क्रिप्ट्स खालील प्रोग्रामिंग लँग्वेजेस (Programming Languages) मध्ये लिहिता येतात: जावा (Java), पायथन (Python), सी-शार्प (C#), जावास्क्रिप्ट (JavaScript), रुबी (Ruby) यामुळे वेबड्रायव्हर विविध प्रोजेक्ट रिक्वायरमेंट्स (Project Requirements) साठी अत्यंत फ्लेक्सिबल (Flexible) ठरतो. सेलेनियम आयडीई (Selenium IDE) च्या तुलनेत, सेलेनियम वेबड्रायव्हर (Selenium WebDriver) हे अधिक रॉबस्ट (Robust), स्केलेबल (Scalable) आणि फ्लेक्सिबल (Flexible) आहे. हे सेलेनियम सूट (Selenium Suite) चा एक महत्त्वाचा भाग असून, फंक्शनल टेस्टिंग (Functional Testing), रिग्रेशन टेस्टिंग (Regression Testing) आणि क्रॉस-ब्राउझर टेस्टिंग (Cross-Browser Testing) साठी मोठ्या प्रमाणावर वापरले जाते. थेट ब्राउझर्स (Browsers) मध्ये एंड-टू-एंड ऑटोमेटेड टेस्ट्स (End-to-End Automated Tests) एक्झिक्युट करण्याची क्षमता असल्यामुळे, सेलेनियम वेबड्रायव्हर (Selenium WebDriver) हे आधुनिक सॉफ्टवेअर टेस्टिंग प्रॅक्टिसेस (Software Testing Practices) साठी अत्यावश्यक टूल मानले जाते.

आर्किटेक्चर ओव्हरव्ह्यू (Architecture Overview)

सेलेनियम वेबड्रायव्हर (Selenium WebDriver) हे क्लायंट–सर्व्हर आर्किटेक्चर (Client–Server Architecture) फॉलो करते. या आर्किटेक्चरमध्ये टेस्ट स्क्रिप्ट्स आणि ब्राउझर यांच्यात थेट संवाद साधण्यासाठी वेगवेगळे घटक एकत्र काम करतात.

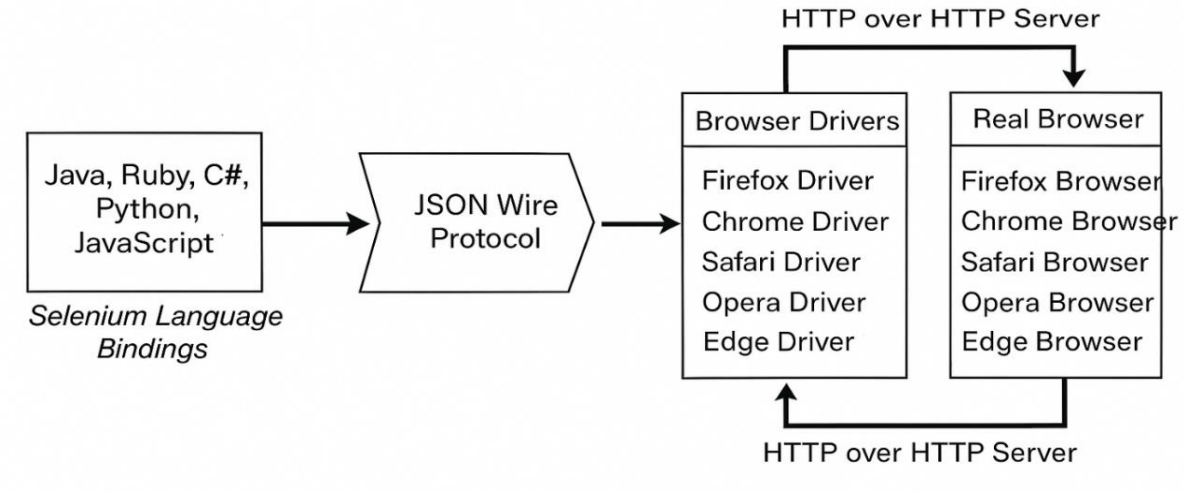

Fig 5.5: सेलेनियम वेबड्रायव्हर आर्किटेक्चर (Selenium WebDriver Architecture)

1. **सेलेनियम लँग्वेज बाइंडिंग्स (Selenium Language Bindings):** टेस्ट स्क्रिप्ट्स (Test Scripts) अनेक प्रोग्रामिंग लँग्वेजेस (Programming Languages) मध्ये लिहिता येतात, जसे की: जावा (Java), रुबी (Ruby), सी-शार्प (C#), पायथन (Python) आणि जावास्क्रिप्ट (JavaScript).
2. **जेसन वायर प्रोटोकॉल (JSON Wire Protocol):** हे लँग्वेज बाइंडिंग्स (Language Bindings) सेलेनियम वेबड्रायव्हर (Selenium WebDriver) सोबत जेसन वायर प्रोटोकॉल (JSON Wire Protocol) वापरून संवाद साधतात. हा प्रोटोकॉल क्लायंट (Client) आणि सर्व्हर (Server) यांच्यात जेसन फॉरमॅट (JSON Format) मध्ये डेटा ट्रान्सफर करण्यासाठी वापरला जातो.
3. **ब्राउझर ड्रायव्हर्स (Browser Drivers):** जेसन वायर प्रोटोकॉल (JSON Wire Protocol) मधून आलेल्या कमांड्स संबंधित ब्राउझर-स्पेसिफिक ड्रायव्हर्स (Browser-Specific Drivers) कडे पाठवल्या जातात,
   उदा.: फायरफॉक्स ड्रायव्हर (Firefox Driver), क्रोम ड्रायव्हर (Chrome Driver), सफारी ड्रायव्हर (Safari Driver), ओपेरा ड्रायव्हर (Opera Driver), एज ड्रायव्हर (Edge Driver)
4. **रिअल ब्राउझर (Real Browser):** प्रत्येक ब्राउझर ड्रायव्हर (Browser Driver) संबंधित रिअल ब्राउझर (Real Browser) सोबत एचटीटीपी (HTTP) द्वारे संवाद साधतो.

उदाहरण: फायरफॉक्स ड्रायव्हर (Firefox Driver): फायरफॉक्स ब्राउझर (Firefox Browser) या प्रक्रियेद्वारे प्रत्यक्ष ब्राउझरमध्ये टेस्ट कमांड्स (Test Commands) एक्झिक्युट होतात. कम्युनिकेशन फ्लो (Communication Flow) संपूर्ण प्रक्रिया पुढील प्रमाणे चालते: सेलेनियम स्क्रिप्ट्स (Selenium Scripts), एचटीटीपी रिक्वेस्ट्स (HTTP Requests) ड्रायव्हरकडे, आणि ड्रायव्हरकडून ब्राउझरकडे. या प्रक्रियेमुळे डायरेक्ट ब्राउझर ऑटोमेशन (Direct Browser Automation) शक्य होते.

5.5.2 सेलेनियम वेबड्रायव्हरचे फायदे (Advantages of Selenium WebDriver)

1. **क्रॉस-ब्राउझर कम्पॅटिबिलिटी (Cross-Browser Compatibility):** क्रोम (Chrome), फायरफॉक्स (Firefox), एज (Edge), सफारी (Safari) आणि ओपेरा (Opera) यांसारख्या प्रमुख ब्राउझर्सना सपोर्ट होते.
2. **मल्टी-लँग्वेज सपोर्ट (Multi-Language Support):** टेस्ट स्क्रिप्ट्स खालील भाषांमध्ये लिहिता येतात: जावा (Java), पायथन (Python), सी-शार्प (C#), रुबी (Ruby), पीएचपी (PHP), जावास्क्रिप्ट (JavaScript).
3. **रिअल-टाइम इंटरॅक्शन (Real-Time Interaction):** बटण (Buttons), टेक्स्ट फील्ड्स (Text Fields), ड्रॉपडाऊन्स (Dropdowns) यांसारख्या ब्राउझर एलिमेंट्स (Browser Elements) शी थेट संवाद होतो. प्रत्यक्ष युजरप्रमाणे वर्तन (Real User Simulation) होते.

4. **टेस्टिंग फ्रेमवर्क्ससोबत इंटिग्रेशन (Integration with Testing Frameworks):** ज्युनिट (JUnit), टेस्टएनजी (TestNG), पायटेस्ट (PyTest), एनयुनिट (NUnit) यांसोबत सहजपणे काम करते. टेस्ट मॅनेजमेंट (Test Management) आणि रिपोर्टिंग (Reporting) सुलभ होते.
5. **पॅरॅलल आणि रिमोट एक्झिक्युशन सपोर्ट (Support for Parallel & Remote Execution):** सेलेनियम ग्रिड (Selenium Grid) वापरून डिस्ट्रिब्युटेड टेस्टिंग (Distributed Testing) शक्य होते. लॅम्ब्डाटेस्ट (LambdaTest), ब्राउझरस्टॅक (BrowserStack) सारख्या क्लाऊड-बेस्ड प्लॅटफॉर्म्स (Cloud-Based Platforms) सोबत इंटिग्रेशन होते.
6. **कम्युनिटी आणि ओपन-सोर्स (Community & Open Source):** मोठा सपोर्ट बेस (Support Base). वारंवार अपडेट्स (Updates) होते. पूर्णपणे ओपन-सोर्स (Open-Source) उपलब्ध होते.

5.5.3. सेलेनियम वेबड्रायव्हरचे तोटे / मर्यादा (Disadvantages of Selenium WebDriver)

1. **शिकण्याची जास्त अवघड प्रक्रिया (Steeper Learning Curve):** सेलेनियम आयडीई (Selenium IDE) पेक्षा, प्रोग्रामिंग (Programming) आणि टेस्टिंग फ्रेमवर्क्स (Testing Frameworks) चे नॉलेज आवश्यक.
2. **बिल्ट-इन रिपोर्टिंग नाही (No Built-in Reporting):** टेस्टएनजी (TestNG), अल्यूअर (Allure), एक्स्टेंट रिपोर्ट्स (ExtentReports) यांसारख्या एक्स्टर्नल टूल्स (External Tools) सोबत इंटिग्रेशन करावे लागते.
3. **डायनॅमिक एलिमेंट्स हाताळणे क्लिष्ट (Handling Dynamic Elements is Complex):** अजॅक्स-हेवी ॲप्लिकेशन्स (AJAX-Heavy Applications) साठी सिंक्रोनायझेशन (Synchronization) आणि वेट्स (Waits) वापरणे आवश्यक.
4. **स्क्रीनशॉट्स किंवा व्हिडिओसाठी नेटिव सपोर्ट नाही (No Native Support for Screenshots or Videos):** स्क्रीनशॉट कॅप्चर (Screenshot Capture) आणि टेस्ट एव्हिडन्स (Test Evidence) साठी अतिरिक्त लायब्ररीज (Libraries) किंवा प्लगिन्स (Plugins) लागतात.
5. **डेस्कटॉप किंवा मोबाइल ॲप्ससाठी थेट योग्य नाही (Not Suitable for Desktop or Mobile Apps Directly):** फक्त वेब-बेस्ड ॲप्लिकेशन्स (Web-Based Applications) साठी योग्य. मोबाइल टेस्टिंग (Mobile Testing) साठी ॲपियम (Appium) इंटिग्रेशन आवश्यक.

5.6 मेट्रिक्स अँड मेजरमेंट (Metrics and Measurement)

सॉफ्टवेअर इंजिनियरिंग (Software Engineering) मध्ये मेट्रिक्स (Metrics) आणि मेजरमेंट (Measurement) यांचा सॉफ्टवेअर क्वालिटी (Quality), परफॉर्मन्स (Performance) आणि एफिशियन्सी (Efficiency) मोजण्यासाठी अत्यंत महत्त्वाचा वाटा असतो. मेट्रिक्सद्वारे मिळणारा क्वांटिटेटिव्ह डेटा (Quantitative Data) डिसीजन-मेकिंग (Decision Making), प्लॅनिंग (Planning) आणि सॉफ्टवेअर डेव्हलपमेंट प्रॅक्टिसेस (Software Development Practices) सुधारण्यासाठी उपयोगी ठरतो. सॉफ्टवेअर मेजरमेंट (Software Measurement) आणि मेट्रिक्स (Metrics) यांचा वापर सॉफ्टवेअर सिस्टिमच्या विविध ॲट्रिब्यूट्स (Attributes) चे मूल्यांकन करण्यासाठी केला जातो, जसे की: साईज (Size), म्प्लेक्सिटी (Complexity), रिलायबिलिटी (Reliability), मेंटेनेबिलिटी (Maintainability).

मेट्रिक्स सामान्यतः खालील तीन प्रकारांमध्ये वर्गीकृत केल्या जातात:

- प्रॉडक्ट मेट्रिक्स (Product Metrics)
- प्रोसेस मेट्रिक्स (Process Metrics)
- प्रोजेक्ट मेट्रिक्स (Project Metrics)

हे तिन्ही प्रकार वेगवेगळ्या उद्देशांसाठी वापरले जातात, पण एकमेकांना पूरक (Complementary) असतात.

1. मोजमाप / मेजरमेंट (Measurement)

मेजरमेंट (Measurement) म्हणजे सॉफ्टवेअर प्रॉडक्ट (Software Product) किंवा सॉफ्टवेअर प्रोसेस (Software Process) च्या विविध ॲट्रिब्यूट्सबाबत क्वांटिटेटिव्ह डेटा (Quantitative Data) गोळा करण्याची प्रक्रिया होय. हा डेटा थेट ऑब्झर्वेशन (Observation), टूल्स (Tools) किंवा टेस्टिंग ॲक्टिव्हिटीज (Testing Activities) मधून मिळतो. मेजरमेंट हा रॉ डेटा (Raw Data) असतो.

उदाहरणे (Examples)

- सापडलेल्या डिफेक्ट्सची संख्या (Number of Defects Found)
- लिहिलेल्या लाईन्स ऑफ कोड (Lines of Code – LOC)
- एक्झिक्युट केलेल्या टेस्ट केसेसची संख्या (Test Cases Executed)

2. मेट्रिक (Metric)

मेट्रिक (Metric) म्हणजे एक किंवा अधिक मेजरमेंट्स (Measurements) वर आधारित कॅल्क्युलेटेड किंवा डिराइव्ह्ड व्हॅल्यू (Calculated / Derived Value).मेट्रिक रॉ डेटाला अर्थपूर्ण माहितीमध्ये रूपांतरित करते, ज्यामुळे क्वालिटी (Quality), परफॉर्मन्स (Performance) आणि इफेक्टिव्हनेस (Effectiveness) चे मूल्यांकन करता येते.

उदाहरणे (Examples)

- डिफेक्ट डेन्सिटी (Defect Density): प्रति हजार लाईन्स ऑफ कोडमागे डिफेक्ट्स.
- टेस्ट कव्हरेज (Test Coverage): टेस्ट केलेल्या कोडची टक्केवारी.
- प्रोडक्टिव्हिटी रेट (Productivity Rate): प्रति पर्सन-आवर लिहिलेल्या लाईन्स ऑफ कोड.

Table 5.2: सामान्य टेस्टिंग मेट्रिक्स – फॉर्म्युला आणि उदाहरणासह (Common Testing Metrics with Formula and Example)

Metric (मेट्रिक)	Formula (फॉर्म्युला)	Example (उदाहरण)
Defect Density (डिफेक्ट डेन्सिटी)	डिफेक्ट्स (Defects) ÷ केएलओसी (KLOC – Kilo Lines of Code) किंवा एफपी (FP – Function Point)	50 डिफेक्ट्स / 10 केएलओसी = 5 डिफेक्ट्स/केएलओसी
Test Coverage (टेस्ट कव्हरेज)	(एक्झिक्युटेड टेस्ट केसेस (Executed Test Cases) ÷ टोटल टेस्ट केसेस (Total Test Cases)) × 100	(1000 / 1200) × 100 = 83.3%
Defect Removal Efficiency (डिफेक्ट रिमूव्हल एफिशियन्सी)	(टेस्टिंगदरम्यान काढलेले डिफेक्ट्स ÷ टोटल डिफेक्ट्स) × 100	(45 / 50) × 100 = 90%
Test Execution Rate (टेस्ट एक्झिक्युशन रेट)	(एक्झिक्युट केलेले टेस्ट केसेस ÷ प्लॅन्ड टेस्ट केसेस) × 100	(800 / 1000) × 100 = 80%
Defect Detection Rate (डिफेक्ट डिटेक्शन रेट)	डिटेक्ट केलेले डिफेक्ट्स ÷ टेस्ट एक्झिक्युशन टाइम (Test Execution Time)	20 डिफेक्ट्स / 40 तास = 0.5 डिफेक्ट्स/तास

5.6.2. मेट्रिक्सचे प्रकार (Types of Metrics)

1. प्रॉडक्ट मेट्रिक्स (Product Metrics)

प्रॉडक्ट मेट्रिक्स (Product Metrics) या अंतिम सॉफ्टवेअर प्रॉडक्ट (Software Product) च्या वैशिष्ट्यांवर लक्ष केंद्रित करतात, जसे की साईज (Size), कॉम्प्लेक्सिटी (Complexity), क्वालिटी (Quality) आणि मेंटेनेबिलिटी (Maintainability).हे मेट्रिक्स एफर्ट एस्टिमेशन (Effort Estimation),कॉस्ट एस्टिमेशन (Cost Estimation) आणि सॉफ्टवेअरची देखभाल सुधारण्यासाठी उपयुक्त ठरतात.

प्रॉडक्ट मेट्रिक्सची उदाहरणे (Examples of Product Metrics):

a. लाईन्स ऑफ कोड – LOC (Lines of Code): लिहिलेल्या कोडच्या एकूण ओळी
b. थाऊजंड्स ऑफ लाईन्स ऑफ कोड (Thousands of Lines of Code(KLOC)): हजारांमध्ये व्यक्त केलेले LOC
c. फंक्शन पॉइंट्स – FP (Function Points): युजर्सना दिलेल्या फंक्शनॅलिटीचे मोजमाप
d. सायक्लोमॅटिक कॉम्प्लेक्सिटी (Cyclomatic Complexity): ब्रांचिंगवर आधारित कोडची गुंतागुंत
e. कॉमेंट डेन्सिटी (Comment Density): कोडच्या तुलनेत कॉमेंट्सचे प्रमाण
f. रीडेबिलिटी मेजर्स (Readability Measures): कोड समजण्याची सुलभता

उदाहरण (Example):
लाईन्स ऑफ कोड (LOC) चा वापर सिस्टिमचा साईज मोजण्यासाठी केला जाऊ शकतो (उदा. स्टुडंट मॅनेजमेंट सिस्टिम साठी 12 KLOC). तसेच डिफेक्ट डेन्सिटी (Defect Density) चा वापर सॉफ्टवेअरची क्वालिटी (Quality) मोजण्यासाठी होतो.

2. प्रोसेस मेट्रिक्स (Process Metrics)
प्रोसेस मेट्रिक्स (Process Metrics) या सॉफ्टवेअर डेव्हलपमेंट प्रोसेस (Software Development Process) आणि टेस्टिंग प्रोसेस (Testing Process) ची एफिशियन्सी (Efficiency) आणि इफेक्टिव्हनेस (Effectiveness) मोजतात. हे मेट्रिक्स प्रोसेस बॉटलनेक्स (Process Bottlenecks) ओळखण्यास, टेस्टिंग इफेक्टिव्हनेस (Testing Effectiveness) तपासण्यास, क्वालिटी ॲश्युरन्स (Quality Assurance) सुनिश्चित करण्यास मदत करतात.

प्रोसेस मेट्रिक्सची उदाहरणे (Examples of Process Metrics):
- डिफेक्ट डेन्सिटी (Defect Density): प्रति युनिट साईज डिफेक्ट्स
- डिफेक्ट रिमूव्हल एफिशियन्सी – DRE (Defect Removal Efficiency): रिलीजपूर्वी काढलेले डिफेक्ट्सचे टक्केवारी
- टेस्ट कव्हरेज (Test Coverage): सिस्टिमपैकी किती भाग टेस्ट झाला आहे
- एफर्ट व्हेरियन्स (Effort Variance): अंदाजित आणि प्रत्यक्ष एफर्टमधील फरक
- सायकल टाइम (Cycle Time): एखादे टास्क पूर्ण होण्यासाठी लागणारा वेळ

उदाहरण (Example): जर 50 डिफेक्ट्स रिलीजपूर्वी आणि 10 डिफेक्ट्स रिलीजनंतर सापडले,
तर: DRE = (50 ÷ 60) × 100 = 83.3%

3. प्रोजेक्ट मेट्रिक्स (Project Metrics)
प्रोजेक्ट मेट्रिक्स (Project Metrics) हे संपूर्ण प्रोजेक्ट मॅनेजमेंट (Project Management) आणि प्रोजेक्टच्या परफॉर्मन्सवर लक्ष केंद्रित करतात. हे मेट्रिक्स कॉस्ट (Cost), एफर्ट (Effort), रिसोर्सेस (Resources) आणि शेड्यूल (Schedule) मॉनिटर करण्यासाठी वापरले जातात.

प्रोजेक्ट मेट्रिक्सची उदाहरणे (Examples of Project Metrics):
- शेड्यूल व्हेरियन्स (Schedule Variance): नियोजित व प्रत्यक्ष वेळेतील फरक
- कॉस्ट व्हेरियन्स (Cost Variance): नियोजित व प्रत्यक्ष खर्चातील फरक
- प्रोडक्टिव्हिटी (Productivity): प्रति पर्सन-मंथ LOC किंवा FP
- रिसोर्स युटिलायझेशन (Resource Utilization): उपलब्ध संसाधनांचा प्रभावी वापर

उदाहरण (Example): एखादा प्रोजेक्ट 6 महिने नियोजित होता पण 7 महिने लागले,
तर: शेड्यूल व्हेरियन्स = –1 महिना

4. ऑब्जेक्ट-ओरिएंटेड मेट्रिक्स (Object-Oriented (OOP) Metrics)
आजकाल बहुतेक सिस्टिम्स ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग (Object-Oriented Programming OOP)) वर आधारित असल्यामुळे OOP मेट्रिक्स अत्यंत महत्त्वाचे झाले आहेत. हे मेट्रिक्स: क्लास लेव्हल (Class-Level) आणि रिलेशनशिप लेव्हल (Relationship-Level) वैशिष्ट्ये मोजतात आणि मेंटेनेबिलिटी (Maintainability), रीयूजेबिलिटी (Reusability)आणि टेस्टिंग एफर्ट (Testing Effort) साठी उपयुक्त ठरतात.

OOP मेट्रिक्सची उदाहरणे (Examples of OOP Metrics):
- लाईन्स ऑफ कोड – LOC आणि इफेक्टिव्ह लाईन्स ऑफ कोड – eLOC: साईज मोजमाप
- सायक्लोमॅटिक कॉम्प्लेक्सिटी (Cyclomatic Complexity): डिसीजन पाथ्सची गुंतागुंत
- वेटेड मेथड्स पर क्लास – WMC (Weighted Methods per Class): क्लास कॉम्प्लेक्सिटी
- नंबर ऑफ मेथड्स पर क्लास (Number of Methods per Class): क्लास स्ट्रक्चर
- अफरेंट कपलिंग – Ca आणि इफरेंट कपलिंग – Ce (Afferent & Efferent Coupling): डिपेन्डन्सी विश्लेषण
- हाइट ऑफ इनहेरिटन्स ट्री – HIT (Height of Inheritance Tree): इनहेरिटन्सची खोली

उदाहरण (Example): जास्त वेटेड मेथड्स पर क्लास (Weighted Methods per Class (WMC)) असलेला क्लास अधिक कॉम्प्लेक्स असतो, त्यामुळे तो टेस्ट आणि मेंटेन करणे कठीण होते. खूप खोल इनहेरिटन्स ट्री कॉम्प्लेक्सिटी वाढवते, पण रीयूज (Reuse) ला प्रोत्साहन देते.

Table 5.3: प्रॉडक्ट, प्रोसेस आणि प्रोजेक्ट मेट्रिक्समधील तुलना (Comparison between Product Metrics, Process Metrics and Project Metrics)

Criterion (निकष)	Product Metrics (प्रॉडक्ट मेट्रिक्स)	Process Metrics (प्रोसेस मेट्रिक्स)	Project Metrics (प्रोजेक्ट मेट्रिक्स)
Definition (परिभाषा)	सॉफ्टवेअर प्रॉडक्ट (Software Product) च्या अॅट्रिब्यूट्सचे मोजमाप करते	डेव्हलपमेंट / टेस्टिंग प्रोसेसेस (Development / Testing Processes) ची एफिशियन्सी मोजते	कॉस्ट (Cost), शेड्यूल (Schedule) आणि रिसोर्सेस (Resources) यांसारख्या मॅनेजमेंट बाबी मोजते
Focus (फोकस)	अंतिम प्रॉडक्टची क्वालिटी (Quality) आणि साईज (Size)	वापरलेल्या अॅक्टिव्हिटीज (Activities) आणि मेथड्स (Methods)	प्रोजेक्ट प्लॅनिंग (Project Planning) आणि कंट्रोल (Control)
Used For (वापरासाठी)	मेंटेनेन्स एस्टिमेशन (Maintenance Estimation), क्वालिटी इव्हॅल्युएशन (Quality Evaluation)	प्रोसेस इम्प्रुव्हमेंट (Process Improvement), डिफेक्ट प्रिव्हेंशन (Defect Prevention)	प्रोग्रेस मॉनिटरिंग (Progress Monitoring), कॉस्ट आणि रिसोर्स मॅनेजमेंट (Resource Management)
Examples (उदाहरणे)	LOC, FP, कॉम्प्लेक्सिटी (Complexity), Defect Density (डिफेक्ट डेन्सिटी)	DRE, टेस्ट कव्हरेज (Test Coverage), मीन टाइम टू रिपेअर (Mean Time To Repair(MTTR))	शेड्यूल व्हेरियन्स (Schedule Variance), कॉस्ट व्हेरियन्स (Cost Variance), प्रोडक्टिव्हिटी (Productivity)
Stakeholders (स्टेकहोल्डर्स)	डेव्हलपर्स (Developers), टेस्टर्स (Testers)	QA टीम (QA Team), प्रोसेस इंजिनियर्स (Process Engineers)	प्रोजेक्ट मॅनेजर्स (Project Managers), क्लायंट्स (Clients)

References

1. Desikan, S., & Ramesh, G. (2016). *Software Testing: Principles and Practices*. Pearson India. ISBN: 9788177581218.
2. Limaye, M. G. (2012). *Software Testing: Principles, Techniques and Tools*. Tata McGraw Hill Education. ISBN: 9780070139909.
3. Chauhan, N. (2016). *Software Testing: Principles and Practices*. Oxford University Press. ISBN: 9780198061847.
4. Singh, Y. (2012). *Software Testing*. Cambridge University Press. ISBN: 9781107652781.
5. Infosys Springboard – *Software Testing Fundamentals*. Available at: https://infyspringboard.onwingspan.com
6. NPTEL – *Software Testing Course*. Available at: https://nptel.ac.in/courses/106101163
7. GeeksforGeeks – *Sanity vs Smoke Testing*. Available at: https://www.geeksforgeeks.org/difference-between-sanity-testing-and-smoke-testing/
8. Guru99 – *Smoke Testing vs Sanity Testing*. Available at: https://www.guru99.com/smoke-testing.html
9. W3Schools – *Software Testing Tutorials*. Available at: https://www.w3schools.in/software-testing/tutorials/