Introduction of software testing

- Program નું Testing કરવા માટે program ને test inputs (અથવા test cases) ના set સાથે provide કરવામાં આવે છે અને તે જોવામાં આવે છે કે program જેવો વિચાર્યો હતો તે મુજબ વર્તે છે.
- જો program expect કર્યા મુજબ વર્તન કરવામાં fail જાય, તો failure હેઠળની condition પછીથી debugging અને correction માટે note કરવામાં આવે છે.
- testing સાથે સંકળાયેલ કેટલાક commonly રીતે વપરાયેલા term નીચે મુજબ છે

✓ **Error:** development team દ્વારા કોઈપણ development phase દરમિયાન જો કોઈ error આવે તો તે એક mistake છે. error ને ફોલ્ટ, bug અથવા ડીફેક્ટ પણ કહેવાય છે

✓ **Failure**: failure એ error (અથવા defect અથવા bug) નું એક form છે. Program માં Error ને કારણે તે fail થઈ શકે છે.

✓ **Test case**: આ triplet [I,S,O] છે, જ્યાં I એ system માં data input છે, S એ system નું state છે જેમાં data ને input કરવામાં આવે છે, અને O system નું expected output છે

✓ **Test suite:** આપેલા બધા software product ને test કરવા માટે ના બધા test case નો set છે જેને Test suite કહેવાય છે.

Verification અને Validation Techniques of testing

Verification એ એક એક મોડ્યુલ ના output ને check કરવાની process છે.
Validation એ આખી બનેલી system કે જે તેની requirements પ્રમાણે છે કે નહીં તે check કરવાની process છે આમ, જ્યારે verification એ error ના phase માં containment સાથે related હોય છે, ત્યારે validation નો aim એ છે કે final product એ error free બને.

Verification	Validation
Verification એ documents, design, code અને program ને verify કરવાની static practice છે	**Validation** એ actual product ને validating કરવા અને testing કરવાની dynamic method છે.
તે code ના executing ને involve કરતું નથી	તે code ના executing ને હમેંશા involve કરે છે.
તેમાં documents અને file નું human base checking હોય છે.	તેમાં program નું execution એ computer base છે
Verification એ inspections, reviews, walkthroughs, વગેરે જેવી method નો use કરે છે.	**Validation** એ black box testing અને white box testing વગેરે જેવી method નો use કરે છે.
તે validation પહેલા કરવામાં આવે છે	તે **verification** પછી follow થાય છે

Design of test cases

What is Test Cases? ટેસ્ટ કેસ શું છે?

- ટેસ્ટ કેસ ને condition ના ગ્રુપ તરીકે વ્યાખ્યાયિત કરવામાં આવે છે કે જેના હેઠળ tester નક્કી કરે છે કે સોફ્ટવેર એપ્લિકેશન customer's ની requirement અનુસાર work કરી રહી છે કે નહીં.
- ટેસ્ટ કેસ ડિઝાઇનિંગ માં preconditions, case નું name, input conditions, and આવનારું result નો સમાવેશ થાય છે.

ટેસ્ટ કેસ ના parameter

- Module Name (મોડ્યુલ નામ)
- Test Case Id (ટેસ્ટ કેસ આઈડી)
- Tester Name (ટેસ્ટરનું નામ)
- Test Case Description(ટેસ્ટ કેસનું વર્ણન)
- Actual Result (એકચુઅલ રીઝલ્ટ)
- Comments (કોમેન્ટ્સ)

Why Write Test Cases? (શા માટે ટેસ્ટ કેસ લખવામાં આવે છે)

- ટેસ્ટ કેસો એ તપાસવામાં મદદ કરે છે કે કોઈ ચોક્કસ મોડ્યુલ/સોફ્ટવેર specified requirement પ્રમાણે વર્ક કરી રહ્યું છે કે નહીં.
- ટેસ્ટ કેસો check કરે છે કે કોઈ ચોક્કસ મોડ્યુલ/સોફ્ટવેર આપેલ conditions ના સેટ ને પ્રમાણે વર્ક કરી રહ્યું છે કે નહીં.
- ટેસ્ટ કેસ સરળ અને સિમ્પલ છે કારણ કે તે સ્ટેપ બાય સ્ટેપ અને સારી રીતે documented છે.

Test Case Example

- યુઝરનેમ અને પાસવર્ડ સાથે લોગિન પેજ માટે ટેસ્ટ કેસ કરવા માટે નું ઉદાહરણ
- **યુનિટ ટેસ્ટ કેસ** : અહીં આપડે ફક્ત એ જ ચેક કરીએ છીએ કે username ઓછામાં ઓછા અગિયાર અક્ષરોની લંબાઈ માટે વેલીડ છે કે નહીં.

ટેસ્ટ આઈડી	ટેસ્ટ કંડીશન	ટેસ્ટ સ્ટેપ	ટેસ્ટ ઈનપુટ	ટેસ્ટ એક્સ્પેકટેડ રીઝલ્ટ	એકચુઅલ રીઝલ્ટ	સ્ટેટસ	રીમાર્કસ
1	ચેક કરો કે યુઝરનેમ ની field અગિયાર અક્ષરો નું ઈનપુટ accept કરે છે કે નહી	1.Give input	Hello world	એક્સેપ્ટ અગિયાર character	એક્સેપ્ટ અગિયાર character	પાસ	none

- **Functionality ટેસ્ટ કેસ:** અહીં check કરવામાં આવે છે કે લોગિન પર ક્લિક કરીએ ત્યારે યુઝરનેમ અને પાસવર્ડ બંને એકસાથે વર્ક કરે છે.

ટેસ્ટ આઈડી	ટેસ્ટ કંડીશન	ટેસ્ટ સ્ટેપ	ટેસ્ટ ઈનપુટ	ટેસ્ટ એક્સ્પેકટેડ રીઝલ્ટ	એકચુઅલ રીઝલ્ટ	સ્ટેટસ	રીમાર્કસ
1	યુઝરનેમ અને પાસવર્ડ સાચો એન્ટર કરીને લૉગ ઈન કરીને ચેક કરો.	1. એન્ટર યુઝરનેમ 2. એન્ટર પાસવર્ડ 3. Click લોગીન બટન	username: Hetasvi password: Hetasvi	Login successful	Login successful	પાસ	None
2	યુઝરનેમ અને પાસવર્ડ ખોટો એન્ટર કરીને લૉગ ઈન કરીને ચેક કરો.	1. એન્ટર યુઝરનામ 2. એન્ટર પાસવર્ડ 3. Click લોગીન બટન	username: Hetasvi password: Hetasvi123	Login unsuccessful	Login unsuccessful	Pass	None

- **યુઝર Acceptance ટેસ્ટ કેસ :** લોગિન પેજ પ્રોપર રીતે લોડ થઈ રહ્યું છે કે નહીં તે માટે યુઝર ફીડબેક લેવામાં આવે છે.

Test Id	Test Condition	Test Steps	Test Input	Test Expected Result	Actual Result	Status	Remarks
1	ચેક કરો કે લૉગિન પેજ user માટે efficiently રીતે લોડ કરી રહ્યું છે કે નહી.	1. ક્લિક લોગીન બટન	None	Welcome to login page	Welcome to login page	Pass	None
2	ચેક લૉગિન પેજ user માટે efficiently રીતે લોડ કરી રહ્યું છે	1. ક્લિક લોગીન બટન	None	Welcome to login page	Page સરખું લોડ થતું નથી	Fail	લૉગિન પેજ લોડ પ્રોપર નથી થયું કારણકે યુઝર સાઈડ થી બ્રાઉઝર નો issue છે.

Black-box testing technique

- Black-box testing માં, test cases ફક્ત input/output ની value ઉપરથી design કરવામાં આવે છે અને design અથવા code નું કોઈ knowledge જરૂરી નથી. Black box ના test case ને designing કરવા માટે નીચે આપેલા બે મુખ્ય approach છે.
  - ✓ Equivalence class Partitioning
  - ✓ Boundary value analysis

Equivalence Class Partitioning

- આ approach માં, program ને input value ના domain equivalence classes ના set માં partition કરવામાં આવે છે. આ partitioning એવું થાય છે કે program નું behavior એ સરખા ભાગ ના class ના દરેક input data ને માટે similar હોય છે.
- આ approach માં equivalence class ના કોઈપણ value સાથેના code નું testing કરવું એ equivalence class ના અન્ય કોઈપણ value સાથે software નું testing કરવા જેટલું સારું છે.
- Input data અને output data ની તપાસ કરીને Equivalence class design કરી શકાય છે. equivalence classes નું designing કરવા માટે નીચે આપેલા કેટલીક general guidelines છે
  - ✓ જો system માં input data value ને value ની range દ્વારા specified કરી શકાય છે, તો એક valid અને બે invalid equivalence class ને define કરવા જોઈએ. ઉદાહરણ તરીકે જો equivalence class 1 થી 10 (e.g. [1, 10]) માં integers નો set છે, તો invalid equivalence class $[-\infty, 0]$, $[11, +\infty]$ છે.
  - ✓ જો કોઈ input data કેટલાક domain ના સ્વતંત્ર member ના set માંથી value લે છે, તો valid input value માટે એક equivalence class અને invalid input value માટેના બીજા equivalence class ને define કરવું જોઈએ. ઉદાહરણ તરીકે જો valid equivalence class{ A,B,C} હોય તો invalid equivalence class એ U- {A, B, C} છે જ્યાં U possible input value નું universe છે
- ટૂંકમાં તે તમામ possible test cases લેવા અને class માં placing કરવાની process છે. Testing કરતી વખતે દરેક class માંથી એક test value લેવામાં આવે છે.
- ઉદાહરણ તરીકે, **Equivalence Partitioning** નો ઉપયોગ કરીને 1 થી 1000 વચ્ચેના **input box accept કરવાના** test cases:
  1. એક input data class સાથે બધા valid input એ valid test case તરીકે range 1 થી 1000 ની single value પસંદ કરો. જો તમે 1 થી 1000 ની વચ્ચે બીજી value પસંદ કરો છો, તો result same બનશે. તેથી valid input data માટેનો એક test case sufficient હોવો જોઈએ.
  2. Input data એ invalid input class ને represent કરવા માટે 1 (Lower Limit) કરતાં ઓછી value સાથે ઉપયોગ કરવામાં આવે છે.

3. Invalid input class ને represent કરવા માટે 1000 (Upper Limit) કરતા વધુ value વાળા Input data હોવા જોઈએ.

- તેથી equivalence partitioning નો ઉપયોગ કરીને તમે બધા possible test cases ને ત્રણ class માં categorized કર્યા છે. કોઈપણ class માંથી અન્ય values સાથેની Test cases તમને same result આપે છે.

Boundary value analysis

- Input domain ના end માં input values એ system માં વધુ errors નું કારણ બને છે. Input domain ની boundary પર વધુ application errors થાય છે.
- Boundary value analysis testing ની technique નો ઉપયોગ input domain ના center માં exist હોય તેવી error ને find કરવાના બદલે boundary ની errors ને ઓળખવા માટે થાય છે.
- Boundary value analysis એ test case designing કરવા માટે Equivalence partitioning નો next part છે જ્યાં equivalence classes ના edge પર test cases ને select કરવામાં આવે છે
- Boundary value analysis નો ઉપયોગ કરીને **1** અને **1000** વચ્ચેના input box accept કરવાના Test case:
  1. input domain ની input boundary ની exactly test data સાથે Test cases i.e. values 1 અને 1000.
  2. input domains ની last edges ની નીચે values સાથે Test data i.e. values 0 અને 999.
  3. input domain ની last edges ની ઉપરની value સાથે Test data i.e. values 2 અને 1001.

WHITE BOX TESTING

- Testing ની આ method માં Code coverage, branches coverage, paths coverage, condition Coverage વગેરેના test cases ની calculation એ system ના internal structure ના analysis ના આધારે calculate કરવામાં આવે છે.
- White box testing માં code ના internal structure ને ધ્યાનમાં રાખીને testing કરવામાં આવે છે અને જ્યારે તમે code ના internal structure ને સરખી રીતે જાણતા હોય, ત્યારે તમે તમારા test cases run કરી શકો છો અને check કરી શકો છો કે system બરાબર document માં લખેલી requirements ને પૂરી કરે છે કે નહીં.
- derived કરેલા test cases ના આધારે user એ system માં input આપીને અને actual output સાથે expected output નું checking કરવા માટે test cases નો ઉપયોગ કરીને કરે છે

Statement coverage

- statement coverage strategy નો aim એ test cases ને design કરવાનો છે જેથી program ના દરેક

statement ઓછામાં ઓછા એક વખત execute થાય

- Statement coverage strategy ને કરવાનો main idea એ છે કે જ્યાં સુધી કોઈ statement executes ના થાય ત્યાં સુધી, તે statement માં કોઈ error આવેલી છે કે નહીં તે નક્કી કરવું ખૂબ મુશ્કેલ છે.
- જ્યાં સુધી કોઈ statement execute ન થાય ત્યાં સુધી, તે illegal memory access, wrong result computation, વગેરેને કારણે failure નું કારણ બને છે તે observe કરવું ખૂબ મુશ્કેલ છે
- For example:

```
If(a>b )
printf("a is greater")
else
printf("b is greater than")
```

- ઉપર આપેલા ઉદાહરણ ના Test cases: {a=5,b=10}, {a=10,b=5}

Branch coverage

- Branch coverage આધારિત testing strategy માં, test cases દરેક branch condition ને એક પછી એક એમ true અને false value ને assume કરવા માટે બનાવવામાં આવી છે.
- આ testing scheme માં Branch testing ને edge testing તરીકે પણ ઓળખવામાં આવે છે, program ના control flow એ graph ના દરેક edge માંથી ઓછામાં ઓછા એકવાર પસાર થાય છે.
- તે obvious છે કે branch testing એ statement coverage ની ખાતરી આપે છે અને આ રીતે આ statement coverage-based testing કરતાં સારી testing strategy છે.
- For example, ત્રણ નંબર માંથી મોટો નંબર શોધો :

```
If(a>b && a>c)
{ max=a; }
else if (b>c)
{ max=b; }
else
{ max=c; }
```

- ઉપર આપેલા ઉદાહરણ ના Test cases: {a=5,b=10,c=15}, {a=5,b=15,c=10}, {a=15, b=5, c=10}

Condition coverage

- આ testing માં, test cases એ composite conditional expression ના પ્રત્યેક component ની true અને false value ને assume કરવા માટે લેવા માટે ના છે
- ઉદાહરણ તરીકે, conditional expression ((c1.and.c2).or.c3), components c1, c2 અને c3 એ દરેકને true અને false value એમ assume કરે છે.
- Branch testing એ એક condition testing strategy છે જ્યાં જુદા જુદા branch statement માં દેખાતી compound condition એ true અને false value ને assume કરવા માટે બનાવવામાં આવે છે
- આમ, condition testing એ branch testing કરતાં વધુ મજબૂત testing strategy છે અને branch

testing એ coverage-based testing કરતાં મજબૂત testing strategy છે.

- N components ના composite conditional expression માં, condition coverage માટે, 2^n test cases જરૂરી છે. આમ condition coverage માટે, test cases ની સંખ્યા component condition ની સંખ્યા સાથે ઝડપથી વધે છે
- તેથી, condition coverage-based testing technique ફક્ત ત્યારે જ practical છે જો n (condition ની સંખ્યા) small હોય.

Path coverage

- path coverage-based testing માટે આપણે test case design કરવાની જરૂર છે, જેથી program માં તમામ linearly independent path ઓછામાં ઓછા એકવાર execute કરવામાં આવે
- Path testing એ module અથવા unit testing માટે વપરાય છે.
- તેને program structure ના complete knowledge ની જરૂર છે.

Compare Black box અને White box testing

Black Box Testing	White Box Testing
Black box testing એ Software testing method છે જેનો ઉપયોગ code અથવા program નું internal structure જાણ્યા વગર software નું testing કરવા માટે થાય છે.	White box testing એ software testing method છે જેમાં software નું testing કરવા માટે જે tester હોય તેને internal structure ને જાણવું પડે છે.
આ પ્રકારના testing tester દ્વારા કરવામાં આવે છે	Generally, આ પ્રકારના testing software developer's દ્વારા કરવામાં આવે છે.
Implementation Knowledge હોવું એ Black Box Testing કરવા માટે જરૂરી નથી	Implementation Knowledge હોવું એ White Box Testing કરવા માટે જરૂરી છે
Programming Knowledge હોવું એ Black Box Testing કરવા માટે જરૂરી નથી	Programming Knowledge હોવું એ White Box Testing કરવા માટે જરૂરી છે
Black box testing એટલે functional test અથવા external testing.	White box testing એટલે structural test અથવા interior testing
Black Box testing માં મુખ્યત્વે test દરમિયાન system ની functionality પર concentrate કરવામાં આવે છે	White Box testing મુખ્યત્વે system ના program code ના testing પર concentrate કરે છે જેમ કે code structure, branches, conditions, loops વગેરે.
Requirement Specifications documents ના આધારે Black Box testing શરૂ કરી શકાય છે.	Detail Design documents ના આધારે White Box testing શરૂ કરી શકાય છે.

Unit Testing

- એક module coded થયા પછી અને successfully reviewed કરવામાં આવે તે પછી Unit testing કરવામાં આવે છે
- Unit testing (અથવા module testing) એ અલગ–અલગ ssystem ના different units (અથવા

modules) નું testing છે

- જ્યારે developer software ને કોડિંગ કરી રહ્યું હોય ત્યારે તેવુ થઈ શકે છે કે dependent module ને testing માટે પૂરા કરવામાં આવતાં નથી, આવા cases માં developers એ stubs અને driver નો ઉપયોગ called (stub) અને caller (driver) unit ને follow કરવા માટે કરે છે.
- Unit testing માટે stubs અને driver ની જરૂર છે, stub એ called unit ને represent કરે છે અને driver calling unit ને represent કરે છે.

Software Documentation

- જ્યારે વિવિધ પ્રકારની software product develop કરવામાં આવે છે ત્યારે માત્ર executable file અને source code જ develop થાય એવું નથી પરંતુ user's manual, software requirements specification (SRS) documents, design documents, test documents, installation manual, વગેરે કોઈપણ software engineering process ના part તરીકે develop કરવામાં આવે છે
- Good document એ software product ની understandability અને maintainability ને વધારે છે.
- Document user ને સારી રીતે s ystem નો ઉપયોગ કરવામાં મદદ કરે છે
- સારા document એ manpower ને સારી રીતે handling કરવામાં મદદ કરે છે
- જ્યારે કોઈ engineer organization છોડશે અને એક નવો engineer આવશે ત્યારે તે પણ જરૂરી knowledge ને easily build up કરી શકે છે.
- વિવિધ પ્રકારના software document ને નીચે મુજબ classify કરી શકાય છે
  - ✓ Internal documentation
  - ✓ External documentation

Internal documentation

- Documentation કે જે software code નક્કી કરવા માટે ઉપયોગમાં લેવાયેલી information પર focus કરે છે તે internal documentation તરીકે ઓળખાય છે
- તે program માં data structure, algorithms અને control flow ને describe કરે છે
- તેમાં header comment block અને program comment આપેલી હોય છે.
- Header comment block એ code ના purpose ને ઓળખવા માટે અને program માં function નો ઉપયોગ detail માં કેવી રીતે થાય છે તે ઓળખવામાં ઉપયોગી છે.
- software code ને ઘણીવાર update કરવામાં અને સુધારવામાં આવે છે, તેથી code information નો record રાખવો important છે જેથી internal documentation એ software code માં કરેલા changes ને reflect કરે છે
- Software માં user ની requirement સાથે દરેક code section કેવી રીતે related છે તે internal documentation માં સમજાવવું જોઈએ. સામાન્ય રીતે, internal documentation નીચેની information ને include કરે છે.
  - ✓ code માં ઉપયોગમાં લેવાતા દરેક variable અને data structure ના Name, type અને purpose
  - ✓ algorithms, logic અને error-handling technique નો Brief description

- ✓ program ની જરૂરી input અને expected output વિશેની Information
- ✓ Program માં up gradation ની Information.

External documentation

- Documentation જે software code ના general description પર focus કરે છે અને તેની detail ઉપર related નથી તેને external documentation તરીકે ઓળખવામાં આવે છે
- Developer કે જેણે code લખ્યો છે, programs પર code ની dependency અને software દ્વારા Produce થયેલું output નું format આ બધુ external documentation માં બતાવવામાં આવે છે.
- સામાન્ય રીતે, external documentation એ program ના design ને describe કરે છે
- તેમાં problem solve કરવા માટે લખેલા program સહિત problem નું description જેવી information આવેલી હોય છે
- તે problem ને solve કરવા માટેના approach, program ની operational requirement અને user interface components ને describe કરે છે
- વાંચી શકાય તેવું અને proper સમજણ માટે, detailed description digit અને figure દ્વારા આપવામાં આવે છે કે કેવી રીતે એક component બીજા component થી related છે
- External documentation સમજાવે છે કે શા માટે software માં કોઈ particular solution પસંદ કરવામાં આવે છે અને implemented કરવામાં આવે છે.
- તેમાં documentation ક્યાંથી મેળવવામાં આવે છે તેના formula, conditions અને references આવેલા હોય છે
- External documentation એ software code ચલાવતી વખતે થતી errors વિશે user ને જાગૃત કરે છે. ઉદાહરણ તરીકે, જો પાંચ નંબરોનો array નો ઉપયોગ કરવામાં આવે છે, તો તે external documentation માં ઉલ્લેખ કરવો જોઈએ કે array ની limit five જ છે.