



NovoTest

V1.00.00

By NovoSync Mobility, Inc

Contents

Prerequisites	2
Overview	2
Licensing.....	2
New or Saved Projects	2
Features	3
Test Case Generation	3
State Machine to Test Cases	3
Boundary Conditions.....	4
Combinations	5
Equivalence Partitioning	6
Test Tools	7
Compatibility Test Platform Priorities.....	7
Automation Defect Avoidance and Stability Control.....	10
Network Bandwidth Utilizer	11
Regression Test via Defect Probability.....	11
Test Report.....	13
Export Test Cases	13

Prerequisites

- Microsoft Windows
- .NET Framework 3.5
- Microsoft Word (for .docx Test Reports)

Overview

NovoTest is a powerful test tool to supplement your current test tool set and test management system. It uses QA industry best practices and proprietary test designs, techniques and methodologies. It allows test case generation for finite state machine models, boundary conditions, equivalence partitioning, and combinations. For test toolsets, it determines and prioritizes target platforms for compatibility testing, assists in ensuring test automation code is stable and thus not allowing false negatives or false positives, determines which product features to perform test execution for regression cycles based on the probability calculated for finding a defect for a set of features, and allows testing under variable network conditions via a bandwidth utilizer feature. Also, export your generated test cases to TestRail Test Management System. With such features, NovoTest's aim is to ensure there is higher test coverage and also to ensure the appropriate testing is comprehensive so that the product can be released with higher quality.

Licensing

The license for NovoTest is a single-node license (see EULA agreement for details). If the project has been activated, the Project menu item will be enabled. Once the activation key is provided (contact fqureshi@novosync.biz) and the software is activated, DO NOT ALTER OR CHANGE THE LOCATION OF THE GENERATED LICENSE FILE).

New or Saved Projects

Once NovoTest is activated, you need to create a new project. After the project has been created and used, save the project for future use. This generates a project XML file. When returning to a saved project, simply open the previously saved XML project file. Only when creating new projects or

opening existing projects, will the features be enabled. The project files are stored in the working directory.

Features

Test Case Generation

State Machine to Test Cases

This features generates test cases from a provided finite state machine. The test cases will test the paths of every possible state from the start state given the correct inputs and will have an expected result of the final state based on the inputs. To generate the test cases, select “Finite State Machine” from the “Test Case Creation” menu item and select your file containing the state machine information.

The file must be a .csv file with **no comma** at the end of last value on each line. State #1 should be the start state. The format should be as follows:

<# of states>,<# of transitions>

<state 1 name>, <state #>

<state 2 name>,<state #>

...

<state *n* name>, <state #>

<input value for going from state x to state y>,<state x #>,<state y #>

<input value for going from state x to state y>,<state x #>,<state y #>

<input value for going from state x to state y>,<state x #>,<state y #>

...

<input value for going from state x to state y>,<state x #>,<state y #>

Example FSM file .csv

6,10

state1,1

state2,2

state3,3

state4,4

state5,5

state6,6

2,1,3

3,1,5

4,1,6

6,2,4

10,2,5

5,3,2

7,4,6

11,5,3

8,6,5

9,6,4

Boundary Conditions

The test cases generated are those based on the input range entered where the test cases will test $(-1, at, +1)$ from the lower bound, 2 random values within range, and $(-1, at, +1)$ for the upper bound.

When entering the test steps, use the character 'X' for where the value will apply. Also, select the data type of the input values.

For example:

Boundary Conditions

Enter Test Steps for generic test case. Use character 'X' to denote variable input (not actual value):

Step	Test Step	Variable Range	Variable Type
Step 1:	initialize product		
Step 2:	click Submit		
Step 3:	enter X		
Step 4:			
Step 5:			
Step 6:			
Step 7:			
Step 8:			
Step 9:			
Step 10:			

Enter variable valid range:

Lower bound value: 5

Upper bound value: 100

Enter variable type: Integer

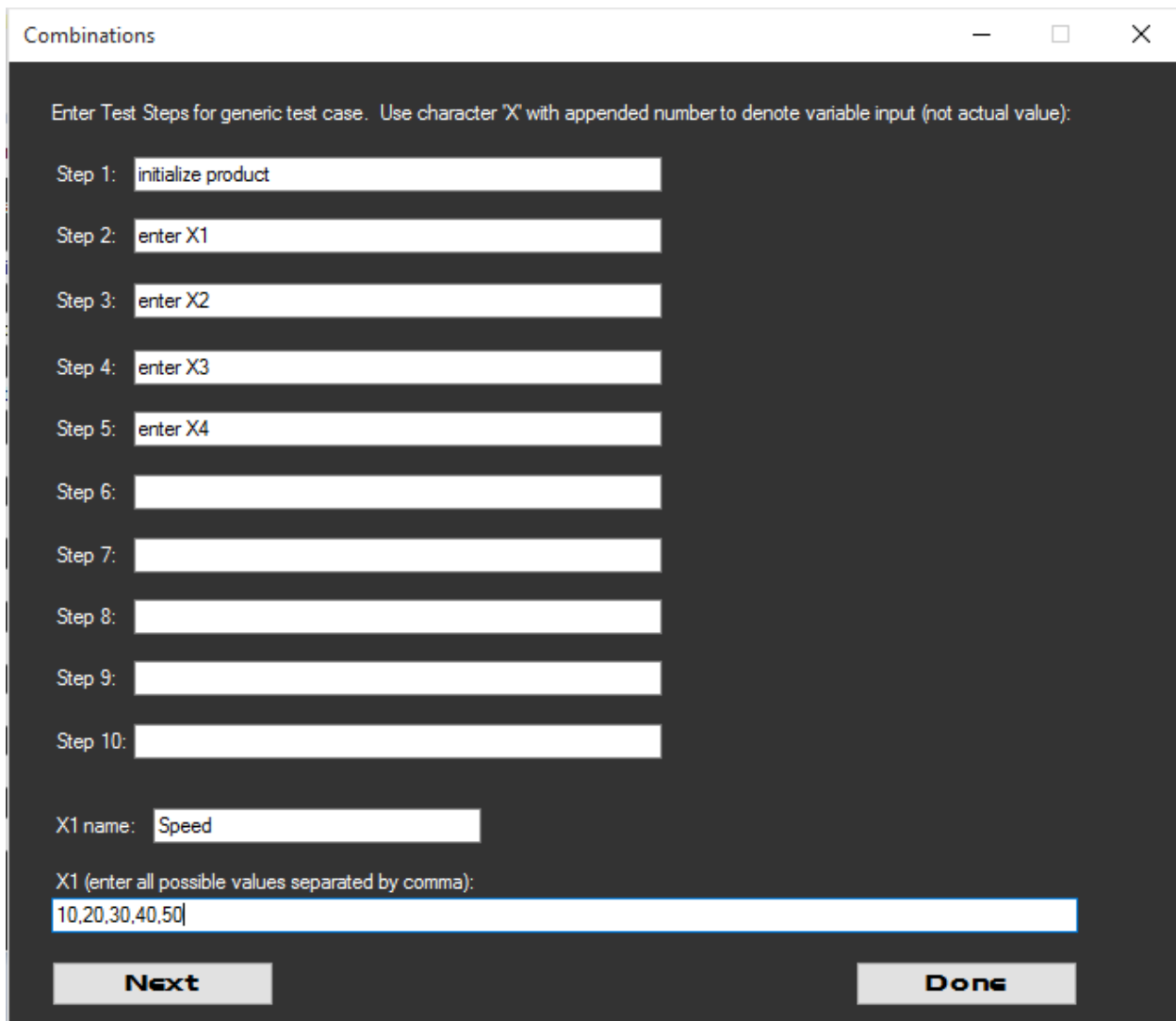
CREATE

Combinations

The tests generated here will generate every possible combination of provided value sets. When entering the test steps, use the character 'X' with a number (starting with one and incrementally increasing). Such as 'X1', 'X2', 'X3'... When X1 values are enter, click "Next" to enter values for X2 and so on. When finished, click "Done".

Note, the complexity of combination is $n!$ so keep this in mind.

For example:



The screenshot shows a window titled "Combinations" with standard window controls (minimize, maximize, close). Inside, there is a text instruction: "Enter Test Steps for generic test case. Use character 'X' with appended number to denote variable input (not actual value):". Below this, there are ten input fields labeled "Step 1:" through "Step 10:". Step 1 contains "initialize product", Step 2 contains "enter X1", Step 3 contains "enter X2", Step 4 contains "enter X3", and Step 5 contains "enter X4". Steps 6 through 10 are empty. Below the steps, there is a field for "X1 name:" containing "Speed". Below that, a larger field is labeled "X1 (enter all possible values separated by comma):" and contains the text "10,20,30,40,50". At the bottom, there are two buttons: "Next" and "Done".

Equivalence Partitioning

https://en.wikipedia.org/wiki/Equivalence_partitioning

When entering the generic test steps, use the character 'X' (only, without a number appended) where the value will replace it. Click "Next" for the next partition range and "Done" when complete.

For example:

Equivalence Partitioning

Enter Test Steps for generic test case. Use character 'X' to denote equivalence group data input (not actual value):

Step 1:	<input type="text" value="initialize product"/>	Enter equivalence group 1 data range:
Step 2:	<input type="text" value="enter X"/>	Lower bound value: <input type="text" value="1"/>
Step 3:	<input type="text" value="click output"/>	Upper bound value: <input type="text" value="5"/>
Step 4:	<input type="text"/>	Enter group data type: <input type="text" value="Integer"/>
Step 5:	<input type="text"/>	
Step 6:	<input type="text"/>	<input type="button" value="Next"/>
Step 7:	<input type="text"/>	
Step 8:	<input type="text"/>	
Step 9:	<input type="text"/>	<input type="button" value="Done"/>
Step 10:	<input type="text"/>	

Test Tools

Compatibility Test Platform Priorities

This is a proprietary algorithm that determines which platforms to test on in priority order when testing for compatibility which gives the widest possible test coverage as it would be impossible to test every platform and variant given time and resources. Examples of platforms could be mobile devices, web browsers, etc. Examples of attributes could be screen resolution, memory, processing speed, OS

version, etc. The algorithm bases the wide coverage based on Q-distance. The weights used give more or less importance to certain attributes.

Q-distance determines the “distance” from a “seed” to another platform. Based on the value, the set of platforms to test on can be systematically chosen. The method used to calculate the Q-distance utilizes a relative scaling times a priority weight. Multiplying the scale value by the priority weight is performed on a chosen set of attributes of each smart-phone. The priority weight is chosen by the tester which should represent how important the attribute is for the product. After this is done on the chosen set of attributes, the results are summed for that particular platform. The result of the sum is called the Q-Value (a new unit of measure required for Q-distance calculations). Once all Q-Values have been calculated, the “seed” is the platform that has a Q-Value as a median (or closest to median). This platform is called S. The Q-distance is the difference of S and the other platform. Once this is determined, the highest priority set of platforms to test on are the ones with the largest Q-distance, including S, and are categorized as P0 devices. The Q-distance is marked as positive or negative relative to S. At each selection, one device must be from positive, and the other from negative. The next in line are the platforms that have the next largest Q-distance. This continues till constrained by time and/or resources. This way, there is the widest test coverage and have reduced the probability of failure on non-tested platforms. If the attribute is not quantifiable, it can be assigned sequential integer values relatively from smallest to largest.

Example

Devices:

D1, D2, D3, D4, D5, D6

Attributes:

A1 = dpi, A2 = Memory, A3 = OS version

	A1	A2	A3
D1	96	256 MB	2
D2	126	512 MB	1
D3	200	1024 MB	3
D4	220	256 MB	3
D5	150	512 MB	1
D6	200	512 MB	2

Weights: $W1 = 3$, $W2 = 1$, $W3 = 2$

$S = D6$ (closest to the middle)

Q-distance($S,1$)	0.417	-
Q-distance($S,2$)	0.414	-
Q-distance($S,3$)	0.329	+
Q-distance($S,4$)	0.139	+
Q-distance($S,5$)	0.324	-

Thus:

P0 devices = D6, D1, D3

P1 devices = D2, D4

No time to test D5!

Compatibility Test Platform Priorities

	Name	Attribute #1 Val	Attribute #2 Val	Attribute #3 Val	Attribute #4 Val	Weight #1	Weight #2	Weight #3	Weight #4
Platform - 1	Samsung Galaxy	100	1	10	22.5	2	1	3	2
Platform - 2	Moto Droid	200	3	20	33.3				
Platform - 3	Fire Phone	200	2	15	10.3				
Platform - 4	iPhone 4	300	3	20	30.5				
Platform - 5	iPhone 4S	100	4	5	20.1				
Platform - 6	iPhone 5	200	5	10	40.4				
Platform - 7	iPhone 6	200	2	5	50.6				
Platform - 8	iPad Air 2	300	3	10	30.5				
Platform - 9	Kindle Fire HDX	100	4	30	20				
Platform - 10	Nexus 4	400	2	20	10				
Platform - 11	Nexus 5	200	1	10	50				
Platform - 12	Samsung Galaxy S6	300	3	15	2.2				
Platform - 13	Moto X	100	4	30	31.2				
Platform - 14	LG G4	400	2	10	30				
Platform - 15	iPhone 6 Plus	200	1	30	10				
Platform - 16	Samsung Galaxy Note	300	3	10	30				

Weight #1Weight #2Weight #3Weight #4

CALCULATE

Automation Defect Avoidance and Stability Control

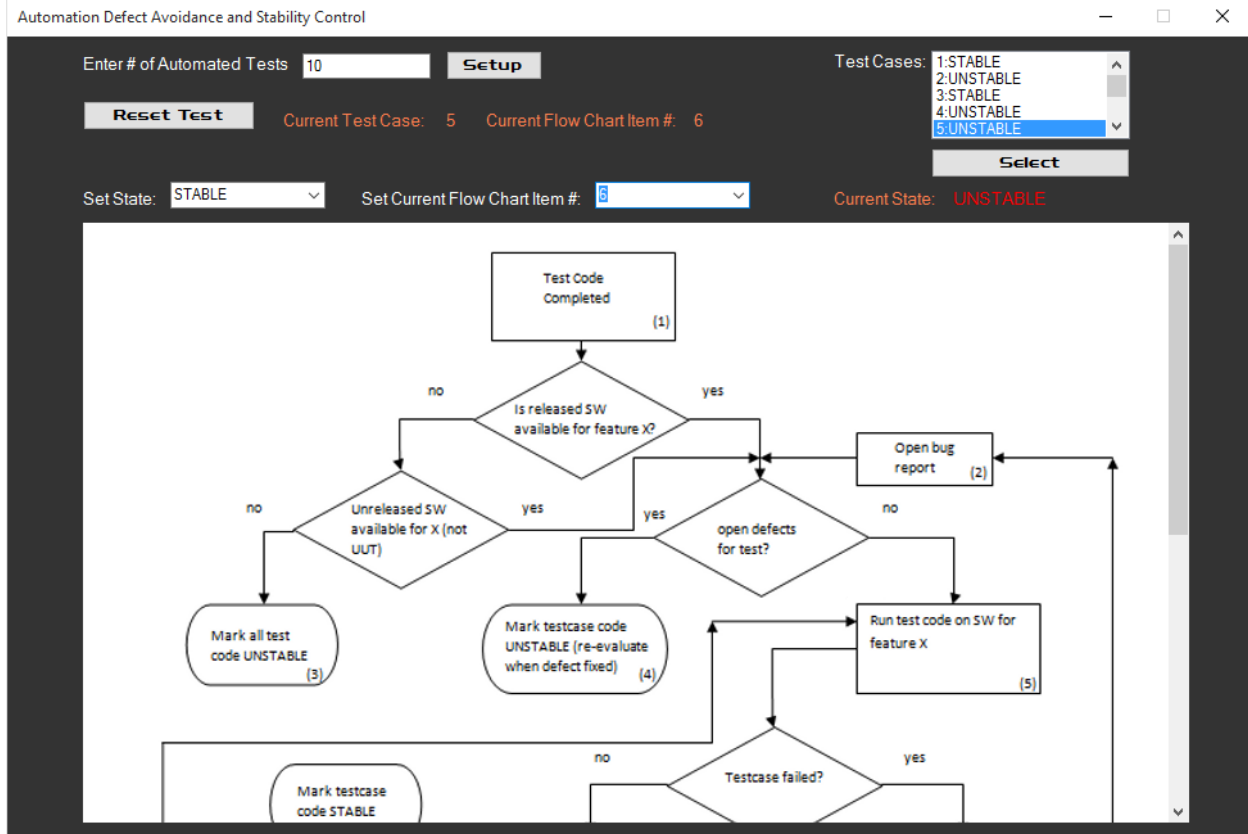
Automation is a critical part of Software Testing. Before test cycles begin, whether it be for Agile or Waterfall models, there is a test development phase for creating either manual or automated tests, or both. Automation is vital as it can allow wide test coverage that executes rapidly without the need for additional resources. It can prove to be a tremendous time-saver as the tests can run in parallel with manual test execution. However, there can be a detrimental drawback to automation if not developed accurately; bugs in the test code that create false positives and false negatives.

If there are bugs in the code for automation, the false positives and false negatives are worse than a defect in the product itself. The reason for this is that the impact affects the company internally as well as externally during production, whereas a defect in the product software has a direct impact externally. First, a false positive allows the defect to be missed. Second, false negatives waste testers' and developers' time and resources internally during development, due to incorrect bug reports generated.

Therefore, robust test code is more difficult to develop as there can absolutely be defects in it. This is opposed to the product code, where defects are initially expected. Only a perfect system can "destroy" correctly, but "creation" is allowed to be flawed. This is proven by the necessity of Test and Validation, and no "Test and Validation" for the testers automated code in the same sense.

However, to minimize the defects in the automation code, a simple process can be followed to lower the probability of serious internal and external impacts. The model is called Automation Defect Avoidance and Stability Control, or ADASC.

This feature assists in ensuring automation test code is stable by following a guided process. To use the tool, enter the number of automated test cases and click "Setup". Then SELECT the test case being worked on. As you follow through the flow chart/process for ensuring the code is stable, mark the test as "STABLE" or "UNSTABLE" and the current step you left off at for returning to the tool at a later time. You can also "Reset" the test to restart the process for that test case.



Network Bandwidth Utilizer

This tool allows testing the product while network bandwidth is being used at the set level by the trackbar on the main application. This is done by setting the gateway IP address and then start the tool which will use the amount of bandwidth set by the trackbar. This simulates testing the product as networks vary.

Regression Test via Defect Probability

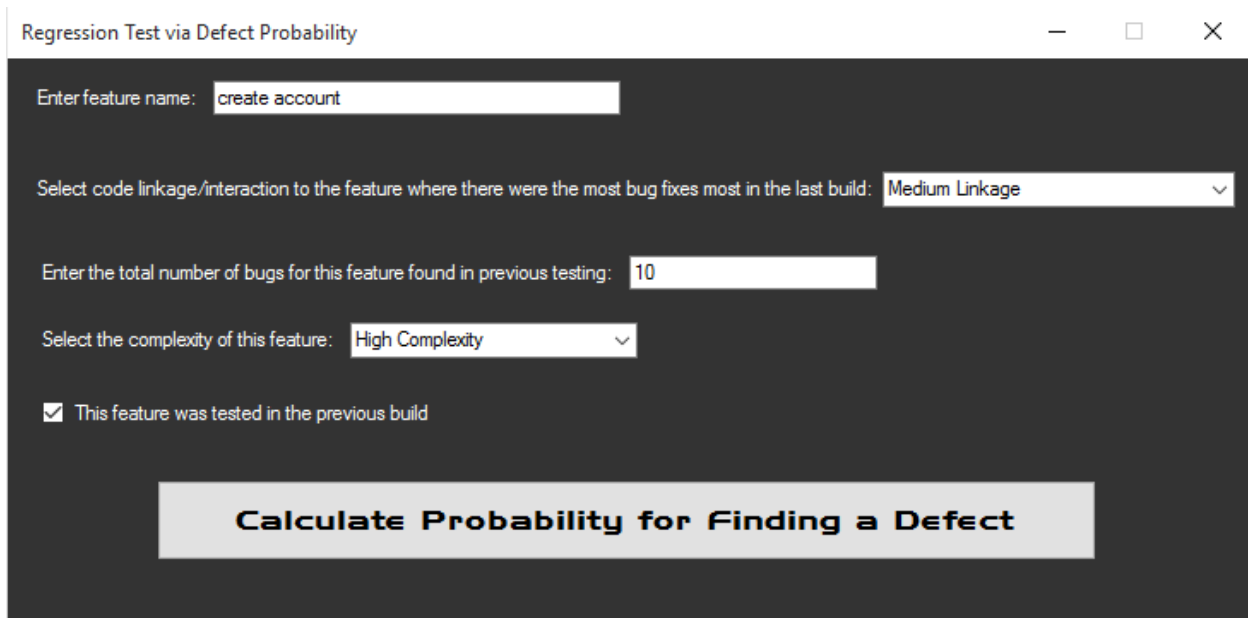
This feature determines how likely it is to find a regression defect for a particular feature. It uses a proprietary algorithm to determine the probability of finding a defect in that feature. To use the tool:

- 1) enter the feature name
- 2) set the code linkage level to the feature where there were the most bug fixes in the previous build
- 3) enter the total number of bugs found in this feature in previous testing

4) enter the code complexity of the feature

5) check whether this feature was tested in the previous build (do not check if it was not)

This process can be repeated for any of the product features, thus resulting in which of those features have higher priority to test given the probability of finding a bug.



Regression Test via Defect Probability

Enter feature name:

Select code linkage/interaction to the feature where there were the most bug fixes most in the last build:

Enter the total number of bugs for this feature found in previous testing:

Select the complexity of this feature:

☒ This feature was tested in the previous build

Calculate Probability for Finding a Defect

Sometimes, it is not so easy to determine what to test for regression when a new build of software resolves certain defects. Also, in many cases, it is not viable to retest everything due to limited time and/or resources. Thus, smart decisions have to be made as to what to perform regression testing on to reduce the risk of releasing with potential defects.

One solution for determining what to perform regression testing on is based on what parts of code the defect fix most closely couples with or interacts with. Another solution would be to base the regression testing on the feature's likelihood of failure and its impact of failure, which should be determined prior to starting any test execution. There are other criteria, but any one criterion on its own, is inefficient and ultimately inaccurate to the reality of what other areas are truly impacted by the defect resolution.

There should be a systematic mechanism to easily and accurately determine which features of software should be tested for regression. Thus, using a formula that incorporates multiple criteria would provide a better and more efficient solution for the determination. The

formula is probabilistic due to the uncertainty of whether or not a defect will actually be discovered as it is based on future and unknown events.

Test Report

Once test cases have been generated and executed, you can create a Test Report. This creates a Word .docx document in the working directory.

Export Test Cases

Once test cases have been generated, you can export the tests to TestRail XML format. The file is stored in the working directory. After the tests have been exported, they can be imported by TestRail. TestRail is a Test Management System by Gurock.

NOTE: DO NOT CHANGE THE LOCATION OF THIS DOCUMENT