Object Based Framework for Risk Driven Test Automation

Reghunath Balaraman
Test Lead
Testree, Nous Infosystems, Bangalore, India
b.reghunath@gmail.com

Object Based Framework for Risk Driven Test Automation

Reghunath Balaraman
Testree, Nous Infosystems, Bangalore, India
b.reghunath@gmail.com

Abstract

The paper presents an object based dynamic framework for dynamically selecting and automating the regression test to address business risk. The framework is built so that the test is driven by the risk assessment of the functions that constitute the application and the decision on the extent to which risks are to be covered during each test cycle.

The framework is built on a modular approach and is built to provide high degree of flexibility and to provide a good degree of abstraction. The organization of the framework is such that the users can build scenarios, modify tests and define alternate paths for execution without modifying the script so that even personnel not familiar with scripting can use the test suite for regression testing. It also generates a consolidated result file and a matrix for re-assessing the risks based on the test results.

The test scenarios are dynamically built during test execution. Also, it has built in intelligence for dynamic sequencing of scripts based on the result of each test execution and are generated based on the risk coverage required during that round of execution based on the result of the preceding tests.

Introduction

The goal of the testing process is to minimize the business risk. The testing effort should concentrate in finding high priority defects which can thus minimize the risk. All the traditional testing processes have addressed the issue of risk, either explicitly or implicitly. Since the basic objective of testing is minimizing risk, the quality of a test can be evaluated based on the coverage of risks.

This in turn will mean that the test strategy shall depend on the risks identified for different functions. And with the changes, modifications and additions being done in the application before each regression test round, the risk assessment also will change. Also, the previous failures of test cases also will increase the risk assessment of functions. So, an appropriate method for calculating the risk shall be in place. A simple such method based of the probability of failure and impact of failure (cost) is briefly mentioned in the succeeding section.

With the assumption that we have identified the risks while evaluating the manual test cases, we base risks as the driver for the automated test suite. For every test cycle, based on the test changes/fixes/additions, we have to identify the risk and drive the test based on the identified risks.

This framework is designed and developed for automating regression test using Quick Test Professional version 8.2 (Quick Test Professional is a very popular test

automation tool from Mercury Interactive), but can be tailored for use with other industry standard test automation tools as well. This paper does not discuss the details of tailoring required for use with tools other than Quick Test Professional.

Risk and Risk Based Test Selection

Risk is an undesired event which has some probability of happening. This event will have some impact on the system which can be referred to as damage. Damage caused by the occurrence of such an event can be in the form of financial loss, damage to other functions or subsystems and/or cost of repair or rework.

Risk associated with a function can be calculated by assessing Probability of fault in a function and the impact of fault using the relationship:

Risk of a function = Probability of a fault x **Impact of fault**

This relationship can be further enhanced by introducing multiple impact factors and probabilities that each problem factor might occur. Suitable weights also shall be assigned to each on these, so that a weighted sum of impacts and weighted sum of probabilities are calculated. The product of these two weighted sums will give the risk. This needs to be calculated for all the functions of the application to be tested, and can be tabulated as in the Risk Matrix (refer table -T1) given below.

Weights / Functions	Impact Factors				Probabilityof problem				Risk
	Fact 1	Fact 2	..	Fact n	Pro 1	Pro 2	...	Pro n	
Function 1									
Function 2									
Function 3									
Function 4									
Function 5									
Function 6									
Function 7									
Function 8									
Function 9									

Table - T1 Risk Matrix

Based on the risk calculated for each of the functions, the decision as to which of these functions has to be regression tested can be made. For example, we can divide the risk into three groups of high, medium and low risk and the test can cover all the high risk functionalities only. Alternatively, a more exhaustive regression round will cover all the high and medium risk functionalities.

This decision has to be made based on the criticality of the application, confidence level required after each regression round, and the perceived severity of the addition and/or modification done to the application before the regression round etc. This decision can form a part of the test strategy. One or a set of test scenarios will be

executed to test each of these functions, depending on the nature and complexity of the function. While selecting scenarios, careful planning is required so that a minimum number of scenarios cover all the critical test cases and thus, adequate coverage of the functionality is obtained.

It is possible that one or many of the test cases that form part of a scenario may fail during the test suite execution. In a static scenario where a definite set of test cases are executed in a pre-defined order, the scenario may successive failures if one test fails. In an analogous manual test, the manual tester may select a test case which the tester from his experience understand as an alternate path for continuing the scenario or set the necessary pre-conditions for the next test case, if it were dependant on the previous test.

This framework tries to build this intelligence into the test suite by dynamically building scenarios rather than statically defining it. It uses this intelligence provided to the suite in the form of "Scenario Map" to select test cases depending on the pass/fail condition of the previous test case, thus making the regression suite more dynamic and more robust.

A sample "Scenario Map" that is used in the framework for automatic selection of test cases for each scenario is shown in Table- T2.

Function Name	Scenario Name	Test Case List		
		Execute	On Pass	On Fail
Function 01	Scenario_01	TC_01	TC_02	TCF_01
		TC_02	TC_03	TCF_02
		TC_03	TC_07	TCF_03
		TC_07		TCF_04
		TCF_01	TC_02	
		TCF_02	TC_03	
		TCF_03		
		TCF_04	TC_09	TCF_05
	Scenario_02			

Table - T2 Scenario Map

It can be observed that there can be many scenarios to be tested against each function, and each scenario has several test cases. The matrix maps the test cases with the next in the sequence based on the outcome of the test execution of each test case. This is dynamically handled by the "Scenario Builder" module of the automation framework described below.

Framework Overview

This framework is originally developed for use with Quick Test Professional of Mercury Interactive, and hence assumes the use of external VBScript files. But, this framework can be suitably tailored for other test automation tools as well. A high level overview of the framework is given in Fig – F1 below.

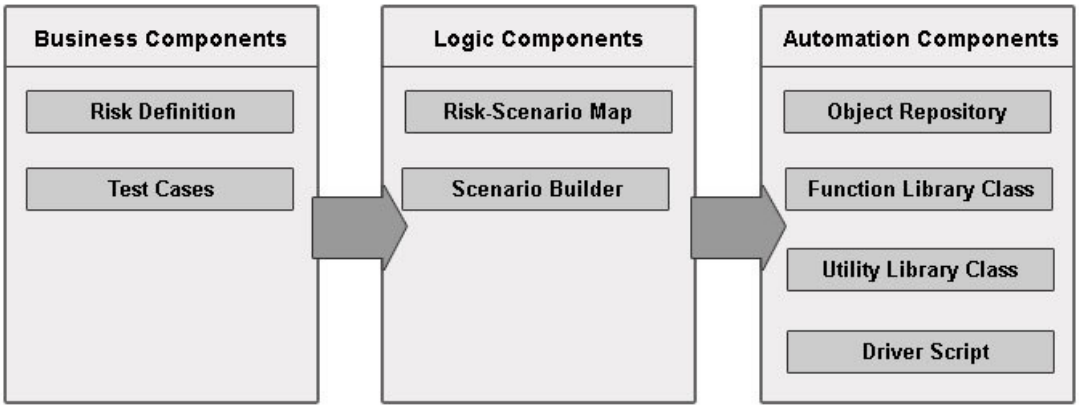

Fig. F1 Overview of the Framework

The framework essentially has three component clusters:

(1) The Business Components

(2) The Logic Components

(3) The Automation Components.

Business Component Cluster

Business Components consist of the Risk Matrix (refer Table-T1) which identifies and quantifies the business risk of each of the functions in the application and the functional test cases which validate these functions. Each test case is uniquely identified by a test case number. The test cases follow the conventions of the keyword driven framework and hence will consist of action words and associated parameters. Table-T3 illustrates the sample of a typical test case using action words (key words) and the associated parameters.

Test Case ID	TC_01
Description	Verify navigation to the e-mail login page.

Keyword	Param_1	Param_2	Param_3	Param_4
Comment	**Begin Test Case**			
Test Case	TC_01			
Comment	**Invoke the given URL**			
Invoke	www.yahoo.com			
Comment	**Verify any given page to see if it is displayed correctly**			
Verify_Page	Yahoo_Home_Page			
Comment	**Navigate to any of the links in the page**			
Navigate	mail			
Comment	**Verify any given page to see if it is displayed correctly**			
Verify_Page	Email_Login_Page			
Comment	**End of test case**			
END				

Table - T3 Test Case in Keyword Format

Logic Component Cluster

Logic Component Cluster consists of the risk-scenario map and the scenario builder. Risk-scenario map (referred as Scenario Map earlier in this paper) is illustrated in Table-T2 above. Scenario builder is a script that dynamically controls the test cases to be called in the scenario. Scenario builder continuously monitors the execution of each test case, and calls the appropriate test case on completion of each test case based on the result of the execution of the current test case.

At any given point of time, the test suite maintains the information about the last run test scenario, the test case last executed and its result and the test case currently being executed. With this information, at the end of a test case execution, it scans the scenario file for the next test case to be executed. The “Scenario Builder” script hands the control to the driver script to execute the test case selected. On completion of the test case execution, the driver updates the status of the test execution and hands the control back to the “Scenario Builder”.

For Example, in Table-T2, when Scenario_01 is to be executed, the scenario builder first calls test case TC_01 for execution. If this test case is successfully executed, TC_02 is called for execution. In case TC_02 fails, TCF_02 is called for execution as mentioned in the scenario map. On completing TCF_02 successfully, TC_03 is called for execution and the scenario continues. If TCF_02 fails for some reason, the scenario execution is stopped at that stage as no alternate path is specified in the map when TCF_02 fails.

Automation Component Cluster

Automation Component cluster includes all the automation elements like Object Repository, Function Library, Utility Library, Driver Script and all the environment

settings in an XML file. These components are very much similar to the corresponding components in a keyword driven framework.

All the keywords are abstract representation of a set of actions or unit functions, and scripted in a VBScript class file. All functions that are application independent and are of more generic nature are scripted in the Utility class. The driver script in this case has very limited functionality as the actual flow of test execution is controlled by the scenario builder in the logic component cluster. The driver script will call the appropriate keyword implementation with appropriate set of parameters. The result of each test case execution and test suite flow control are handled back to the driver at the end of each test case.

Framework Implementation Details

Risk Matrix is the driving force of the framework. So, calculating and preparing the Risk Matrix is the most important part in implementing the framework. Risk for all the business functions are to be calculated initially using the risk calculation method described earlier. Thus, we will arrive at a value of risk (probability of fault x impact of fault) for each function.

This value of the risk will be used as selection criteria to decide whether those functions are to be included in the regression round or not. For example, if a quick regression round is being planned with only the high risk functions being covered, we can set a value of 25% for the risk coverage in the environment settings (an XML file which will be used by the test suite), which will mean that only those functions falling in the top 25% category will be selected.

The scenario builder script reads the value for risk coverage defined in the environment settings XML file, and selects the functions which are qualified for testing the present regression round. The scenarios to be executed for each function is defined in the scenario map file (refer Table –T2). Each of these scenarios to be executed for the functions are further mapped to a set of test cases. But the test cases executed and the actual sequence of execution will dynamically change because if a test case fails, the scenario can still continue with an alternate path defined in the scenario map file.

This dynamic selection of test execution is very important not only because it will enable alternate paths for scenario execution, but also because the alternate test cases mapped can be planned to ascertain failures, satisfy pre-conditions for the succeeding test if it is dependant on its preceding test case etc.

This is made possible by the use of user defined environment variables, which stores the unique ID of the test case being executed, and store the test execution status. Based on this information, the Scenario Builder will appropriately select the next test case from the scenario map. Once a test case is selected for execution, the Scenario Builder passes the information to a driver script which will execute the test cases based on keywords. If, on completion of a test case execution, no further test case is mapped for execution, the Scenario Builder script understands it as the end of the scenario and proceeds to the next scenario to be executed.

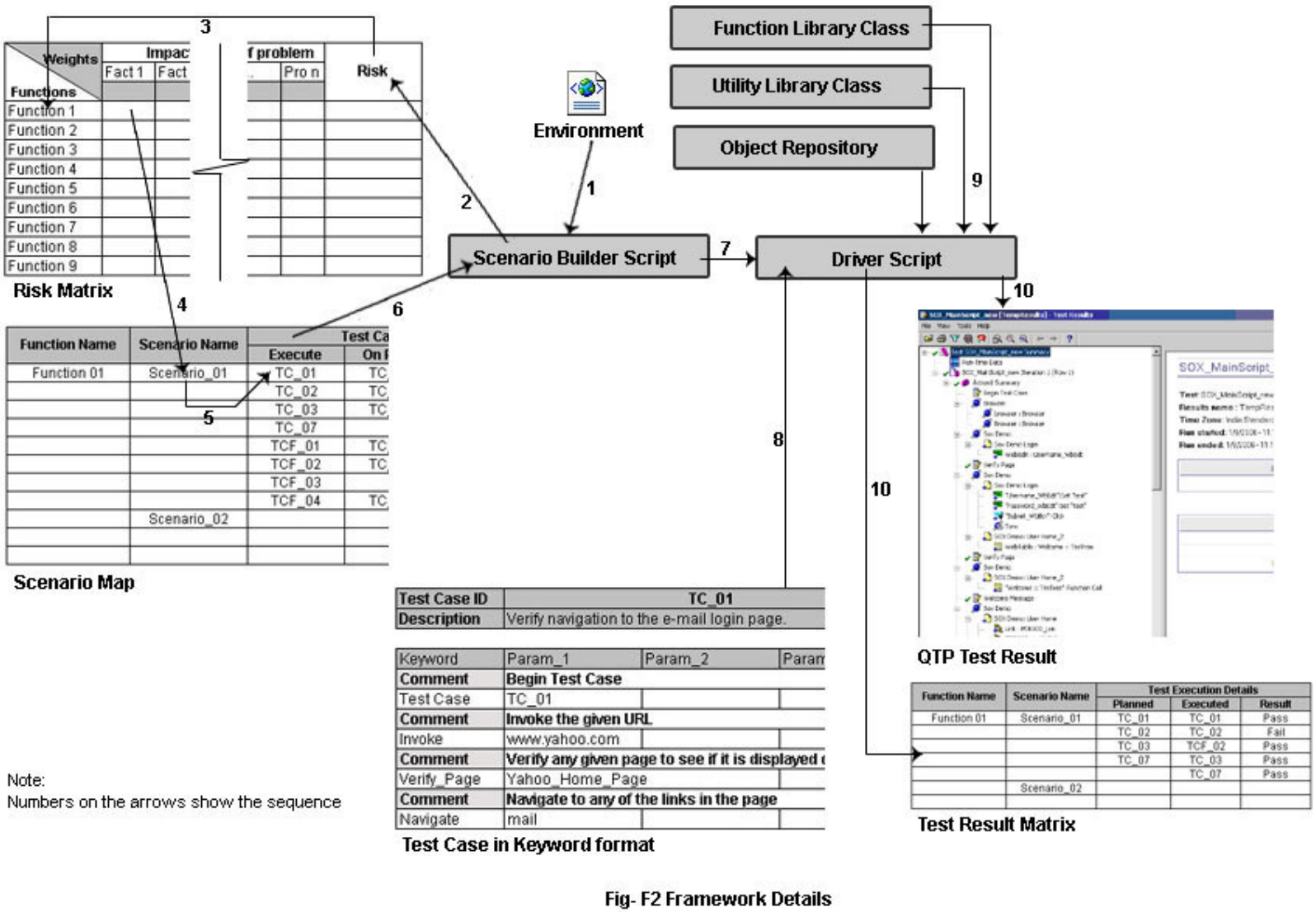

Fig- F2 Framework Details

Driver Script is another important part of the framework and is very much similar to the driver script of a keyword driven framework. The main job of the driver script is to read the test cases which are represented as a set of action words known as keywords. Each of these keywords can take many parameters, and in some cases, some of these parameters will be optional parameters. The driver script will read the parameter list and call the appropriate function from the library class file.

Test Case document is an Excel sheet as shown in Table -T3. The Test case ID is the unique identifier for a test case. The first column of the excel sheet will contain the keywords which represent a unit function or a user action which has a definite implementation in the function library or utility library. All the parameters are listed one in each cell, in the same row as the corresponding keyword. There are a few predefined keywords like “COMMENT” (to insert a comment row, describing the succeeding keyword), “END” (to mark the end of test case etc. Except for these few pre-defined keywords, all keywords are defined based on the functionality of the application/ action steps to be executed.

Utility library contains all application-independent functions while the Function Library contains application dependant functions. These libraries are defined as class files and are scripted using VBScript. This helps in encapsulating the function implementation from the end user. VBScript does not allow overloading of functions. So, calling the functions with optional parameters is achieved by using a dynamic parameter array, and deciding the implementation based on the contents of the array. So, Object orientation is maintained to a limited extent as permitted by the scripting language.

While the test cases are executed, a Test Result Matrix also gets updated with the Function Name, Risk assessed, Scenario name, and the test cases executed and the result of execution of each test case. This is in addition to the tool generated test result, and is very helpful in assessing the risk exposure of each function and in re-visiting the risk assessment. The frame work design integrates all the test cases into one test suite, and hence the tool will generate a single result file which will include all the test cases in the suite those were executed.

Test Results

There are two test results that can be referred for each test execution, one generated by the tool (in this case, QTP) and a result matrix generated by the framework.

Tool generated result

The framework is organized in such a way that the whole test execution cycle is identified as a single test by the test automation tool (QTP). Hence a single consolidated result file is generated. The implementation of the keywords is done such that adequate information is written to the result file for further reference. Figure (Fig – F3) given below will show a sample result file generated by the tool.

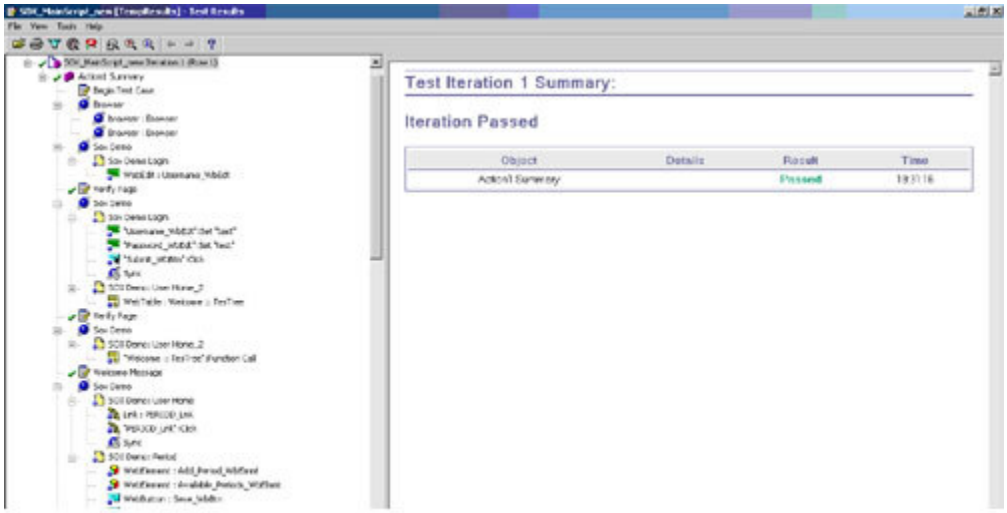

Fig- F3 QTP Result File

Test Result Matrix

Test Result Matrix is the consolidation of the result of the execution of each of the individual test cases executed against the corresponding scenario(s) in which they were executed (refer Table -T4). This helps in analyzing the alternate paths executed, and also in revisiting the risk assessment for the individual functions.

Function Name	Scenario Name	Test Execution Details		
		Planned	Executed	Result
Function 01	Scenario_01	TC_01	TC_01	Pass
		TC_02	TC_02	Fail
		TC_03	TCF_02	Pass
		TC_07	TC_03	Pass
			TC_07	Pass
	Scenario_02			

Table - T4 Test Result Matrix

Benefits of using the framework

The framework was developed to facilitate dynamic selection of regression test cases and execution of the same, ensuring adequate risk coverage. The following are the major benefits of using this framework for test automation:

Appropriate risk coverage

The process of selecting test cases is in several cases based on the tester's assessment of risk and his/her familiarity with the test cases. But, this framework makes the test case selection objective and accurate, thus ensuring that the test cases selected are as per the level of risk addressed.

Flexibility in test selection

The extent to which the risks are to be covered can be defined in environment settings itself. All test selection is done automatically based on these settings, thus eliminating the requirement to alter any script. This provides a high flexibility in test selection during different rounds of test execution.

Dynamically built scenarios

As the script dynamically builds scenarios rather that runs a static set of test cases, the facility of ascertaining failures, defining alternate paths for failed tests etc is possible, making the test suite more robust.

Ease of operation

The test suite developed based on this framework can be executed by anyone who does not have any scripting experience. Modifying the scenarios, defining alternate paths for test cases etc can be done by a person with no prior scripting experience.

Test case modification without scripting

As the test cases are built of action words, the test cases can be modified to add/delete keywords or change the parameters with which the action words are called without any scripting. This makes the test case modification easier. This is possible because the actual implementation of the keywords is in a VBScript class file.

Test results matrix as input to re-assess risk

The test execution generates a test result matrix that can be used to re-visit the risk assessment, thus continuously enhancing the risk assessment and in-turn the test effectiveness.

Conclusion

Since VBScript is not an Object Oriented Programming language, the object orientation is implemented only to the extent possible with the scripting language. An automation tool with a more powerful external scripting language can further enhance the framework. Also, based on the risk to be addressed, all the scenarios related to a function are being selected. This can be further broken down by adding another coverage factor, which can decide which all scenarios related to a function (if there is more than one scenario for testing that function) are to be considered for test execution. Further enhancements to the framework can take these directions, thus enhancing its capabilities.

References

1. Elfriede Dustin, Jeff Rashka, John Paul, Automated Software Testing: Introduction, Management, and Performance, Addison Wiley

2. Hans Shaefer, Risk Based Testing – How to choose what to test more and less, Software Test Consulting

3. Paul Gerrard, Neil Thompson, Risk Based E-Business Testing, Artech House

4. Robert Binder, Testing Object-Oriented Systems: Models, Patterns and Tools, Addison Wiley

5. William E. Perry, Effective methods of software testing, John Wiley & sons

6. Yanping Chen, Manage Risk by Risk-Driven Continual Regression Testing, CITO Innovators Showcase, September 2003

Author's biography

The author, Reghunath Balaraman, is a post graduate in Planning and Systems from university of Indore, India and has over nine years of experience. He has hands on experience in test planning, effort-estimation, test scheduling, test framework development, test suite development, test execution and issue management during various phases of SDLC. Since last four and a half years, he is actively involved in test architecting, test Planning and management and process tailoring.

He has worked in India and overseas, on several test automation projects in various domains like Financial Services (Treasury and Forex), E-commerce applications, Inventory Management systems, Supply Chain Management software and CRM applications. He has extensively worked on test automation and test management products from Mercury Interactive, and has also exposure to several other Industry standard testing tools.

He is a certified tester from International Software Testing Quality Board (ISTQB) and has Job-Role certification in software testing from Brainbench. He also holds a certification in Project Management from Brainbench.

Currently, he is working as a Test Lead for automation projects with Testree, a strategic business unit of Nous Infosystems, Bangalore, India.