Automated Testing – A Literature Review

Sarathy Venkatakrishnan[1], Chethana G[2]
[1]*Department of Electronics and Communication Engineering*
RV College of Engineering, Bengaluru -59
[2]*Assistant Professor, Department of Electronics and Communication Engineering*
RV College of Engineering, Bengaluru -59.

Abstract

The drive for automation is at the forefront of recent revolutionary changes in industries, with more and more corporations realizing the enormous benefits associated with automating the process in their factories or equivalent production units. Though the competitive advantage obtained from reducing human effort in repetitive tasks is well appreciated, many resource managers hold the view that the initial cost of automating a division is too high a risk to take in comparison to the possible rewards from doing so. Hence work in the domain remains impeded to a large extent, with most corporate entities preferring the more straightforward option of assigning testing duties to company employees. This work aims to provide a comprehensive review of the major steps taken in recent years in the field of automation, focusing specifically on the advancements and proposals in the field of automated testing in the software domain. Multiple works in progress and standards for automation specifications are discussed at length in accordance with the scale of the contribution to the field. The pros and cons of automation in the software testing domain are elucidated, showing that the implementation of automation schemes with sufficient prior analysis really do yield large payoffs.

Keywords: *Automation, Testing, Software, Metrics, Tools.*

1.0 Introduction

Automation is the control of machines or system process by independent systems which function using technology based on software. The world is experiencing major changes in this domain, as more and more industry entities are realizing that the initial investments in setting up an automation system are well recovered with added interest from the financial and economical returns. The primary source of such benefits is the creative work that employees are able to give their attention to once the repetitive tasks have been taken care of by a self-regulating system. An especially lucrative application for automation is the IT industry, where the routine testing of software modules comprises the majority of many employees' work. With repetitive tasks like these automated, more focus would be given to development, rather than management. In particular, the development of a system to perform standard regression and sanity tasks on models under design would prove extremely useful to boosting the creativity of engineers. This paper reviews the developments in software automation and the recent advances made in efforts to automate processes in the industry.

2.0 The State of Automated Testing

Test Automation has lasting effects on the software under development's market success. In a global race where the quickest to offer clients a functional product ends up cornering the market, the effects of test automation are majorly beneficial to the cost, functionality and time to reach the marketing stage of the product. Since the initial cost and time of automation needs to be deemed as exceedable by the rapid development in the aftermath with sufficient Return On Investment, the question needs to be

answered firmly in order to put out any doubts in the minds of the resource managers. [1] sought to show that it is indeed so, by measuring the effects of test case automation on three software modules in the domains of the Railway Cloakroom management, a given Restaurant Billing system and a chosen Mini Geometric figure analyser. The results were all in agreement of the overwhelming rewards generated from the extended use of test automation on the software involved.

2.1 Low Level Analysis of Automated Testing

The reason for the popular automation choice of scripting languages is to skip the compilation times and directly run each line of code one by one, so that even in the case of an error, the tests prior to the bug would be successfully passed. In a compiled program, the error in the last line of a program would prevent the binary for the entire program from being generated. The most famous languages for scripting are Tcl/Tk, whose names are derived from Tool Command Language and Tool Kit respectively. In [2], scripts written in these languages are used as readymade modules and a model is proposed to process a high-level command, its inputs and results, and create a new command using the existing script libraries to perform exactly the functionalities as specified in the high-level description. Multiple APIs at both high and low levels were designed and implemented for such command construction.

It is all right to configure the automation of tasks in the work environment, but managing the software modules that comprise the automation setup is a crucial step that needs to be undertaken – and an easy one to overlook. Steps have been taken to build such a system in industrial automation, in particular catering to IEC 61499 functional blocks, whose architecture embraces software developmental techniques such as imperative languages, state machines and object oriented programming. The method for analysing such functional blocks exists and software metrics to characterize the same was given by [3] with reasonably described quality of software modules.

2.2 Metrics for Automated Testing

An essential step to proving the superiority of an automated testing system is the definition of reliable metrics in both the manual and automated domain to efficiently compare the benefits of each method of testing. [4] provides a comprehensive list of metrics which cover the broad range of operations that might be expected from a testing system. On the manual side, the metrics are defined as Test Cases Productivity, Defects Acceptance, Tests Execution, Defects Rejection, Inefficient Fix Defects, Tests' Execution Productivities and Tests' Execution Efficiencies metrics. The automation metrics, on the other hand, are Automated Scripting Productivities, Automated Tests' Execution Productivities and Automated Tests' Coverages. An essential point to note is that of coverage. On the manual end, it is difficult to provide complete coverage of all the possible testing cases for a system that takes a wide range of inputs. If the system is susceptible to a few corner case inputs, however, the manual tests perform better at isolating the cases which are liable to cause problems and bring those to the forefront.

If, however, the range of inputs given coverage are moderately large and uniform, automated testing are much more efficient at repeatedly brute forcing every possible situation to completely provide full test coverage to the operated system.

During software testing, a failure of one automated test is generally the cause for less alarm than a manual test on the same module, the reason being that there is a possibility that something might have gone wrong with the automation program. This is especially possible after a system update or the modification of test scripts which could break existing functionality. [5] proposed a Markov process to show a semblance of the model's correlation among the chosen software runs during the software's development. The result of each test is classified into Software Under Test failure, Successful state, or Automated Test System failure. Using a Markov probability model, the chances of each failure being due to a Software Under Test failure and Automated Test System failure are predicted using the previous state of the system along with a probability matrix of the system changing from one state to any other probability state.

Automation system software is required to provide sufficient diagnosis information for testing to faciliate early imperfection detection in addition to quality estimation. Quite often it is the case that the facets of automation, software testing and diagnosis are intermingled in the code, which creates the problem that it harder to parse, modify and evaluate. To combat situations like this, separate modules built with automation schemes which consider test input during the process are necessary. [6] illustrates the design of a prototype for the same in the domain of unit testing. The model is called Test Driven Automation, and complies with the IEC 61499 standard.

The same authors presented a framework in [7] for linking tests' case generations, executions and the essential reporting function as part of the automated test system's operations. This is based on an established approach well known in business IT software development – Test First Development. A model called the Unified Modelling Language is also discussed, which helps addressing the individual aspects of a system. A limitation of such frameworks is also mentioned, namely that automated test case generation is in need of a systematic testing procedure to identify the non-fluid structure aspects of automation systems.

2.3 Automated Testing Tools

Within software testing, there are plenty of ways one could go about configuring tests, which means it is important to develop the tools independently according to their broad classifications. [8] offered a three category classification scheme on the basis of Testing Manager related tools, Functionality Testing tools and Load-related Testing tools. In the same paper, the more popular software tool options under each category were presented. There is ample room for choice of module testing applications on this front. The primary software testing strategies for doing so as well, including units' testing, integrations' testing, systems' testing and acceptances testing. Depending on the domain of the

automation software in question, different units should also be relegated to performing white box, black box and in some cases, grey box testing – which, from the name is somewhere in between the first two methodologies.

The testing utilities available in the current market are plenty and varied. To select one tool for the purposes of automation is not task to be taken lightly. The parameters to judge every option vary depending on the domain and type of end goal reserved for such automation. [9] in particular has presented not only an extensive list of testing tools available in the market, but analyzed a case in point after selecting the open source tool Selenium based on such analysis of the automation need. It is also necessary to appreciate the benefits of automation testing rather than manual testing, which is obvious when comparing the types of tests that can be performed in either case. It ends up coming down to the execution speed, sequence, and resources required to perform each type, namely Regression testing and Usability testing, which leans very favourably towards automation testing.

In the domain of evaluating testing tools, it is important to establish some common criteria to rank multiple candidates for the job against one another to make a selection. There are many tools in the market to perform such an evaluation task, but the lack of a common metric makes cross- verification a complicated action. [10] has been one such attempt to develop a metric suite to facilitate the comparison and the selection of a desired tool for testing, including the evaluation of such testing tools such as RFT, Ranorex, Janova, QuickTest, SilkTest and Panorama. The evaluation of the tools can be done into parameters such as ease of installation, basis of operation, required coding knowledge to operate, requirement of access to code and listing of testing features. Each tool had its pros and cons but Ranorex was evaluated to be the most well rounded software for the chosen type of tasks.

Some testing software seem more prone to adaptation to automating systems than others, as might be perceived from the general vein of the preceding analysis. One such frequently chosen tool for automation purposes is the software named Panorama by Univera Computer Systems. The platform essentially offers solutions in sales, logistics and service areas. [11] is a case in point, where this software was used to further proves the enormous benefits linked to the use of an automated testing system compared to a system based on manual workforce testing. A test automation project based on the same met with considerable success.

Multiple studies have delved into the domain of automation testing tools and their frameworks, and sought to find the perfect integration between them for maximum efficiency. The extended classification of automation frameworks are generally accepted to be Linear, Modular, Library-related Architecture based, Data-driven, Keyword-base-driven, and a novel Hybrid-methodology-driven testing frameworks. [12] went a step further and analysed multiple automation tools in depth with correlations to the above and their own set of parameters to evaluate the performance of the tool by. These included, but were not limited to, test development platform, range of possible applications under test, use of scripting languages, the programming skills required, learning curve for the same, script

generation/creation time, object resilience and related maintenance, image-centered testing prowess, license types and cost. The tests were conducted on various widely used tools, namely Katalon Studio, Selenium, UFT and TestComplete.

A few specific applications of automated software testing have been carried out. One such application is [13], where a design for automated testing of Service Oriented Architecture based software was presented. It explained the entire process of tests' planning, testing design, test cases' generation, tests' involved execution, test results' analysis and repeated testing. Service Oriented Architecture is a model for creating coupling software systems and open business methods for obtaining access to worldwide-offered software services. These functionalities are made use of by other applications through publishing platforms. The model takes software testing/test specifications as center and considers plenty of parallel options in the overall development and testing processes in question.

The various tools analyzed in recent years for their automation compatibility yield a substantial superset of equally valid options which, as previously mentioned, need to be filtered as per the requirements of the domain using the automation metrics as so heavily enumerated previously. A brief list of the same, amassed for the benefit of would-be resource managers is presented in Table 1.

Table 1

Automation Compatible Testing Tools

Testing Tools	Type of Testing	Use Cases/Applicability
Apache JMeter	Load Testing Functional Testing Regression Testing	Judging quality of networking performance of server/servers, Testing behaviour of components under stress conditions
Selenium	Load Testing	Specializes in Record and Playback tests, Mainly web based applications
TestComplete	Unit Testing GUI Testing Regression Testing	Largely used for mobile application testing purposes, Framework testing, User Experience Testing
WebLoad	Load Testing	Useful in simulating real world stress environments, open source and web-oriented, flexibility with Ajax, Adobe Flex technologies
Junit	Functional Testing Regression Testing Unit Testing	Built towards establishing coordination between developers and testers, Suited for seamless testing between product updates, open source

UFT	Regression Testing	More User-Friendly, Focuses largely on report generation and report validation, Useful for inexperienced units
Load Runner	Load Testing Functional Testing	Focused on simulation of user transactions, Simulates real time unpredictable environments with substantial test case coverage

2.4 Shifting From Manual to Automated Systems

A major cause for concern to resource managers in the industry when faced with the option to begin automating certain processes or procedures in their departments is the transitional cost of the shift from manual to automated testing of the designs under test. Before it is possible to evaluate the major changes which will be required in order to effect a full-scale transition in the system, it is necessary to construct a working software testing lifecycle to evaluate which parameters vary with the type of testing being performed on the design under test. [14] provides a comprehensive lifecycle model which is broadly split into the following stages: Requirements Analysis, Tests Cases' Designs and Developments, Tests Execution Manners, Test Closures' Details and Test Processes' Analyses. In addition to these, a list of candidate parameters to evaluate a test case by are also presented, which is a comprehensive listing of the most critical parameters of every test case sent to the module for evaluation. Multiple tools were inspected to ascertain the friendliness of each to adapting to automation from manual operation, including Apache JMeter, Load Runner, UFT, Selenium and TestComplete. Each of these had a slight edge in different evaluation parameters they were graded against, but there was no one tool which stood out over the rest.

Extending the concept of transitioning from manual to automated testing of software modules, the question of how to smoothly shift from one mode to another in an industry setting was taken up by [15], where the manual testing platform in place was replaced with a GUI-based automated testing software. The major result obtained from observation of the transition process was that the limit of test generation was analysis effort, i.e. the capacity of analysing test results. The manual evaluation of test results varies with the complexity of tests and the use of human experience to judge the results from a task. However, the automation testing is much quicker while parsing through test results but evaluation is murky and the boundaries of whether to classify the results of a test as pass or fail are ill-defined at best. However, if the system generates fairly easily evaluable test results, automated test systems win by a long margin.

As mentioned previously, the transition from manual testing to automated testing systems requires a large amount of work in the beginning. One of these is the automation of old manually executed scripts by new automation tools. This compatibility issue faced in the beginning and similar issues faces when any script needs to be updated due to any reason, e.g. Firmware upgrades, requires the manual updation of scripts for the automated test system to function properly again. The approach suggested in [16] not only offers a model to do so, but also defines the rules for new scripts to detect which old script will be useful for fulfilling the high-level task supplied to it as input. This also requires a Natural Language Processing engine as mentioned previously to work correctly. The model uses heuristics like file similarity and command similarity to create the functions required for new automation scripts.

Another tool designed for a similar purpose is presented in [17], this time for the automated updation of test scripts of legacy versions which get outdated with every new version of the application. The target of the scripts, i.e. the modules with which they interact, are GUI based applications. Thus, new scripts are required to cope with the new type of interactions needed to deal with an upgraded GUI based application. Since these types of operations entail the configuring of interactions with hundreds of little GUI objects, a tool called REST(Reducing Effort in Script based Testing) is proposed to help developers update the scripts. This is an example of a suggested tool to semi- automate the task of updating scripts, since the developers still need to be involved, but this semi- automation is indirectly helping automation of the actual testing of the application as well.

A similar case in Web-based applications where new updates breaks the interactions of automation scripts is presented in [18]. Some testing tools like RFT(Rational Functional Tester) follow a strict hierarchy in the file containing the interacting objects and looks for the same order during the test replay. This means that a separate database must be maintained for the same which causes a lot of overhead. Using descriptive programming, however, the script can be made as an independent entity since the object that is required to be interacted with it searched for inside the browser itself, which is permanently available, so there is no need to store it separately. This allows the standalone automation tool to truly function independently of environment configurations.

The maintenance problem in scripts that are meant to deal with a certain firmware level and no higher is a perennial problem and requires no end of workarounds to prevent the automation tools from breaking. [19] tries a novel approach to deal with this nuisance by having each software module be created with a set of scripts as a part of its package, so that any automation script need only contain the most basic scripts and the rest of the test scripts can be ported from the product package itself. This way, the scripts are already tailored to the settings of the software under test. The Model Driven Architecture proposed lists the primary types of models the system must be compatible with, namely Platform Independent Model and Platform Specific Model, as well as the Toolset and Mapping Facility which needs to be modified for each new software build on the script side.

2.5 Domains of Automated Testing

There exist certain domains where the Return On Investment on automated testing systems are so lucrative that it immediately captures the attention of automation developers. One such area is of test execution programs for software maintenance purposes, as analysed in [20]. Quite often, the tests that are performed in software maintenance systems are of a widely random nature, seeking to simulate the real time stress on the system along with a wide range of unpredictable inputs, ensuring that the system is able to cope with such varied inputs during product launch and will not fail to meet the live tests. The testing for these corner cases from an automation perspective is enormously complex, since the simulating of pseudo-random sequences poses substantial problems to the automated testing system, which is, after all, dealing with a list of testing inputs that does not change over time. Hence, any automation model that successfully tests corner cases and detects loopholes in any software maintenance system would meet with widespread success. This type of testing is called as "Capture/Playback" testing, since the test inputs are designed to simulate the actions of previous erratic information supplied to the system repeatedly until the model is able to cope with those tests – this, however, uses only previous states of the tests and generates nothing random. Many alternative approaches were examined to overcome this problem – one such method was a "Keyword-based approach" which involves a convoluted approach akin to Natural Language Processing.

There are multiple stages of a software development process and efforts have been made to integrate automation into every one of them, namely Requirement Analysis, System Validation/Verification and Code Generation. [21], in fact, has suggested a model for the implementation of an automated test system for all three. The success of such a model hinges on the efficiency of the execution of the first step. A well defined set of requirements translated well into systematic design parameters depends on the analysis of a Natural Language Processing model which makes its judgement of a high level statement of system requirements.

2.6 High Level Analysis of Automated Testing

Quite often, if a resource manager is willing to invest in automating testing in a certain platform, they would rather prefer that the analysis of failures of the tests to be automated as well, to get rid of all the sunk costs in one fell swoop. A module that analysed the reasons for a product failing would be able to suggest more clearly to the engineers involved the possible corrective actions that could be undertaken. Such a hybrid framework which does more than just automate the testing on modules is presented in [22], where a model to determine whether a failure is a false positive, and if it is not, to determine the cause of failure during the automation process. A mechanism to implement parallel processing was also suggested as a part of the model to speed up the execution times of the test case analysing process.

The end goal of any automated system is having the capability to automatically align itself with high-level policies that are presented to it without need for further instructions. As explored in [23], CIM-SPL (Common Information Model – Simplified Programming Language) is one such language that has

been developed for expressing condition-action policies. Resources allocated to be managed by the system are appropriately distributed according to its policy engine. A key component in such an autonomous system would be the data integration module, which would need to keep operators and other closely related software up to date about the decisions taken by the system pertaining to resource allocation.

2.7 Caveats Of Automated Testing

At this stage, it would be helpful to weigh the pros and cons of automation by performing a usability analysis here. These are reasons which prevent a large number of industry agencies from completely automating some elements of their testing system. [24] discusses the same and lists most of the concerns raised by resource management entities at some point of time or the other. These include concerns that the preparatory work for such an automated testing system is very large, which out shadows the work required to test an individual module and hence is not worth repeating for every module manufacture. The financial risk associated with automating the testing of a module demands a certain level of caution in the proportion of tests approved for automation. An entry ratio of 1:4 is recommended for software products with large financial risk with a gradual transition to full automation once stabilization occurs.

2.8 Other Automation Applications

Software of the embedded type in automotives has the need to change frequently to match the modifications in specifications or hardware design. [25] takes the industry case of AUTOSAR - a global partnership of car manufacturers and suppliers from the electronics, technological and software industry. They work on the independent development of standards to provide electric and electronic architecture. This requires automotive systems to be controlled by embedded software since it features a client-server mechanism. There have been attempts to model an automated testing protocol for such a system with a HVAC(Heating Ventilation and Air Conditioning) module built on top of it and testify the propriety of the same as standard architecture.

Apart from automation of testing on models, another avenue which automation has great scope if system upgrades. This model is proposed in [26] as an alternative to Next-Gen Network's blade switch software which has a manual upgrade policy. The possible model of an expert system that uses a two step inference engine is described in detail. Expert systems are automated systems which are trained in the beginning of their lifetimes by a knowledge engineer and an expert in the domain. After this primary training, they are equipped with a logical reasoning database as well as an inference engine to interpret the rules accumulated in their storage during their training period. It was shown that the network upgrade, when performed automatically, improves the dependency of the network switching upgrades and decreases the required administrators for the network drastically.

3.0 Conclusion

A general trend of automation in the industry is already apparent with the large influx of developers hired with that particular end goal in mind. The advances in the field of automated testing in the software domain were discussed at length, along with detailed descriptions of the models and standards proposed to regulate the same. Automation of all repetitive tasks is one of the key components to achieve ideal efficiency in any system regardless of the domain. And as new tools and frameworks are developed towards that end goal, the benefits of the same will be reflected in the increased efficiency of developers and likewise by the quality of the products.

References

1. D. Kumar and K. Mishra, “The impacts of test automation on software cost, quality and time to market,” Procedia Computer Science, vol. 79, pp. 8–15, 2016.
2. H. Tohjo, I. Yoda, T. Goto and T. Yamamura, "An implementation of TMN OpS development platform using high-level APIs on Tcl/Tk language-robust OpS development platform using scripting language," GLOBECOM 97. IEEE Global Telecommunications Conference. Conference Record, Phoenix, AZ, 1997, pp. 1700-1705 vol.3.
3. G. Zhabelova and V. Vyatkin, “Towards software metrics for evaluating quality of IEC 61499 automation software,” in 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, 2015.
4. R.M.Sharma, “Quantitative analysis of automation and manual testing,” International Journal of Engineering and Innovative Technology, 2014.
5. C. Lin and C. Huang, "Modeling the Software Failure Correlations When Test Automation Is Adopted during the Software Development," 2008 19th International Symposium on Software Reliability Engineering (ISSRE), Seattle, WA, 2008, pp. 307-308.
6. Winkler, Dietmar & Hametner, Reinhard & Biffl, Stefan. (2009). "Automation component aspects for efficient unit testing." IEEE Conference on Emerging Technologies Factory Automation, 2009.
7. Winkler, Dietmar & Hametner, Reinhard & Östreicher, Thomas & Biffl, Stefan. (2010). "A framework for automated testing of automation systems.", IEEE Conference on Emerging Technologies and Factory Automation (ETFA).
8. K. Sneha and G. M. Malle, “Research on software testing techniques and software automation testing tools,” 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), IEEE, 2017.
9. M. Uday Patkar Harshal Shedge, “Automation testing in software organization,” International Journal of Computer Applications Technology and Research Volume 5 Issue 4, 198 - 201, 2016.
10. S. K. S. Tarik Sheth, “Software test automation approach on evaluating test automation tools,” International Journal of Scientific and Research Publications, Volume 5, Issue 8, 2015.

11. E. Celik, S. Eren, E. Cini, and O. Keles, “Software test automation and a sample practice for an enterprise business software,” 2017 International Conference on Computer Science and Engineering (UBMK), IEEE, 2017.
12. C. Z. Mubarak Albarka Umar, “A study of automated software testing: Automation tools and frameworks,” International Journal of Computer Science Engineering, 2019.
13. Wu Deng, Shuqin Liu and Jingjing Liu, "Automation testing process modeling method of SOA-based isomerous software," 2009 International Conference on Industrial Mechatronics and Automation, Chengdu, 2009, pp. 129-132.
14. B. N. Shalini Gautam, “Descriptive study of software testing & testing tools,” International Journal of Innovative Research in Computer and Communication Engineering, vol. 4, no. 6, pp. 10 288–10 295, 2016.
15. C. Klammer and R. Ramler, “A journey from manual testing to automated test generation in an industry project,” 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), IEEE, 2017.
16. S. Kitajima, S. Kikuchi and Y. Matsumoto, "Identification of Related Management Scripts for Efficient Automation of Cloud Management Tasks," 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, Singapore, 2014, pp. 747-750.
17. Qing Xie, Mark Grechanik and Chen Fu, "REST: A tool for reducing effort in script-based testing," 2008 IEEE International Conference on Software Maintenance, Beijing, 2008, pp. 468-469.
18. T. Agarwal, "A Descriptive Programming Based Approach for Test Automation," 2008 International Conference on Computational Intelligence for Modelling Control & Automation, Vienna, 2008, pp. 152-156.
19. A. H. Ahmed, A. A. A. SidAhmed and R. B. Eltoum, "Automation of test scripts in software product line using Model driven architecture," 2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), Khartoum, 2015, pp. 62-66.
20. T. Wissink and C. Amaro, "Successful Test Automation for Software Maintenance," 2006 22nd IEEE International Conference on Software Maintenance, Philadelphia, PA, 2006, pp. 265-266.
21. X. Liu, "Process Oriented Analysis for Software Automation," 2009 First International Workshop on Education Technology and Computer Science, Wuhan, Hubei, 2009, pp. 1131-1133.
22. H. S. Chaini and S. K. Pradhan, "Test script execution and effective result analysis in hybrid test automation framework," 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, 2015, pp. 214-217.
23. D. Kaminsky, B. Miller, A. Salahshour, and J. Whitmore, “Policy-based automation in the autonomic data center,” 2008 International Conference on Autonomic Computing, IEEE, 2008.

24. X. Meng, "Analysis of software automation test protocol," Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology, Harbin, 2011, pp. 4138-4141.
25. H. Moon, G. Kim, Y. Kim, S. Shin, K. Kim, and S. Im, “Automation test method for automotive embedded software based on AUTOSAR,” 2009 Fourth International Conference on Software Engineering Advances, IEEE, 2009.
26. L. Tang, P. Yan, Z. Liu and J. Chen, "A Next-Gen Network Switch Software Automation Upgrade Method Based on a Two-step Inference Expert System," 2009 International Conference on Information Management and Engineering, Kuala Lumpur, 2009, pp. 161-165.