International Journal of Innovative Computing, Information and Control
Volume **8**, Number **1(A)**, January **2012**

ICIC International ©2012 ISSN 1349-4198
pp. **489-500**

ARCHITECTURE BASED SOFTWARE QUALITY MAINTENANCE RIPPLE EFFECT ANALYSIS

Sajid Anwar[1], Awais Adnan[1], Masoom Alam[1], Tamleek Ali Tanveer[1]
Muhammad Ali[1], Muhammad Ramzan[2], Arfan Jaffer[3]
Arshad Ali Shahid[3] and Abdul Rauf[3]

[1]Department of Computer Science
Institute of Management Sciences Peshawar
7/B3, Phase 5, Hayatabad, Peshawar
{ sajid.anwar; awais.adnan; masoom.alam; tamleek; muhammad.ali }@imsciences.edu.pk

[2]University of Arid Agriculture
Shamsabad, Muree Road, Rawalpindi, Pakistan
ramzan@uaar.edu.pk

[3]Department of Computer Science
FAST National University of Computer and Emerging Sciences
A. K. Brohi Road, H-11/4, Islamabad, Pakistan
{ arshad.ali; arfan.jaffer; a.rau }@nu.edu.pk

Received July 2010; revised November 2010

Abstract. *One of the major challenges for software developer is to fulfill the quality requirements of the software systems. This emphasis on software quality has some serious implications in terms of customer satisfaction and system acceptance. Due to its significance, it is also considered as one of the major challenges to be met by software developer since s/he is responsible for fulfilling the quality requirements of the software systems. One way to address this challenge is to adopt architecture based software development. Software architecture as an artifact can be used to deal with software quality attributes (QA) such as maintainability, performance and reliability. Recently, the focus of business is transforming rapidly from manual to computer based automations. Resultantly, software intensive systems are becoming large and complex. This is enhancing our emphasis on system quality which ultimately gives rise to the need for software quality maintenance. In order to optimize the quality of the software maintenance in a rapidly changing environment, study of its ripple effect is very significant. In this paper, a methodology for architecture based quality maintenance ripple effect determination and analysis is proposed. A QA's Property-component connectivity matrix and component connectivity matrix for ripple effect analysis is also described. The methodology is illustrated and evaluated by using web content extraction application architecture. The advantages of the proposed methodology are also presented.*

Keywords: Software maintenance, Ripple effect analysis, Quality attributes

1. Introduction.

The phenomenon of ripple effect in software development is not new. The term ripple effect was first introduced by Haney [1] using a technique called "Module connection analysis". This technique was based on the idea that changes in one module necessitate a change in any other module/modules. Several other software maintenance models such as SADT [2] and Methodology for Software Maintenance [3] have considered ripple effect measure as a necessary factor. Software maintenance consumes enormous number of organization's overall resources and has some serious implications in terms of cost and effort [4]. Typically, at least 60% of software life cycle cost is associated with software maintenance activities [5]. So, ripple effect analysis based maintenance technique would help maintenance practitioners to understand the scope of effect of their changes and would also help them in identifying what parts of the system must be checked for consistency [6].

With the passage of time, software intensive systems are becoming large and more complex. Software intensive systems evolve over time which represents crucial aspect of software maintenance. These obligatory changes often tend to be most important changes on part of the customers and end-users as they better understand their needs. Wrong interpretation and implementation of these obligatory changes are a nightmarish scenario and a real but high risk which often causes the system to deviate from its original design. On the other hand, these changes also have significant time pressure associated with these due to which the developers cannot fully evaluate the impact of these crucial changes on system architecture leading to lower quality and making future changes even more unmanageable. So, it is important to focus on software architecture while dealing with changes [7,27]. Software architecture plays a significant role in understanding the system as it reflects different components of the system and relations between these components. Software architecture provides highest level of abstraction and thus makes a system understandable and maintainable. These characteristics of software architecture make it suitable to deal with software qualities [8,25].

We have to be mindful of the fact that any change introduced in the system will not only affect one specific component, but also inevitably affect certain other components directly or indirectly. Thus, ripple effect is inevitable whenever any change will be introduced while software is in the phase of development or maintenance. Any change introduced in the system involves making changes at different levels, i.e., sources code, software design and software architecture, etc. Therefore, ripple effect analysis can be used to help development or maintenance team in identifying modules which may cause problems during software maintenance. It can be used to predict the impact of modifications on the rest of the program or the system. The effect of the modification may not be local

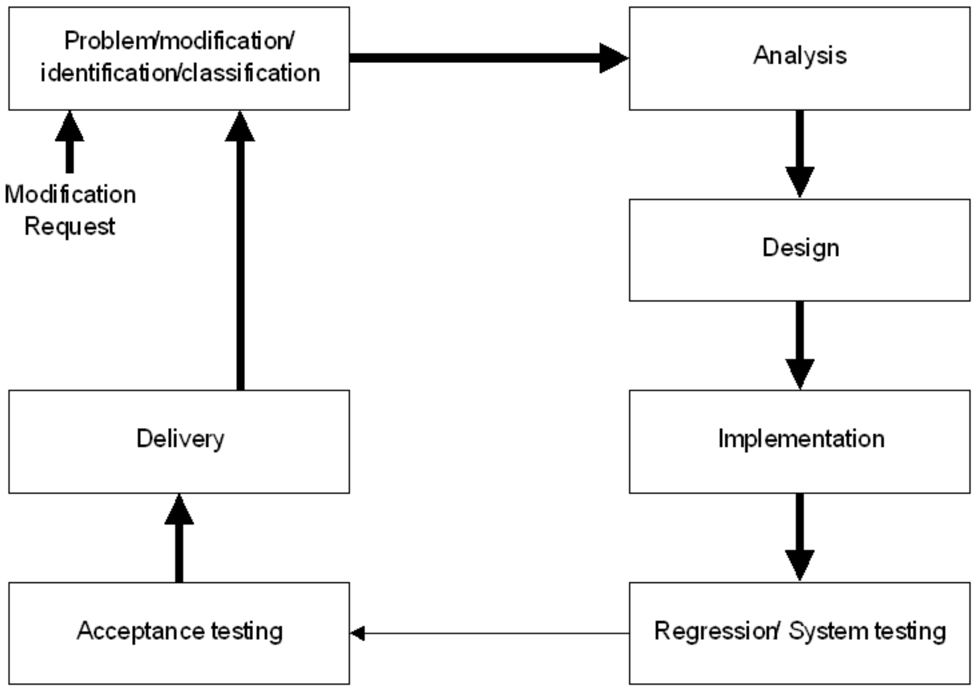
Problem/modification/ identification/classification
Analysis
Modification Request
Design
Delivery
Implementation
Acceptance testing
Regression/ System testing

Figure 1. The IEEE maintenance process [10]

to the module. Depending upon the stability of the program or system, the impact of modification may be large (e.g., poor stability) or constrained otherwise [9].

Software maintenance is one of the focused areas in software engineering. Several software maintenance models have been proposed in past. Some of these models implicitly deal with ripple affect analysis while others deal with ripple effect analysis explicitly. The IEEE standard organizes the maintenance process in seven phases as described in Figure 1 [10].

The analysis step of this model is conducted at two levels: feasibility analysis and detailed analysis where feasibility analysis implicitly deals with impact analysis. The task oriented software maintenance model [11] explicitly deals with impact analysis (cf. Figure 2). The purpose of this step is to find out how the intended modifications would cause side effects in the system and where these would occur.

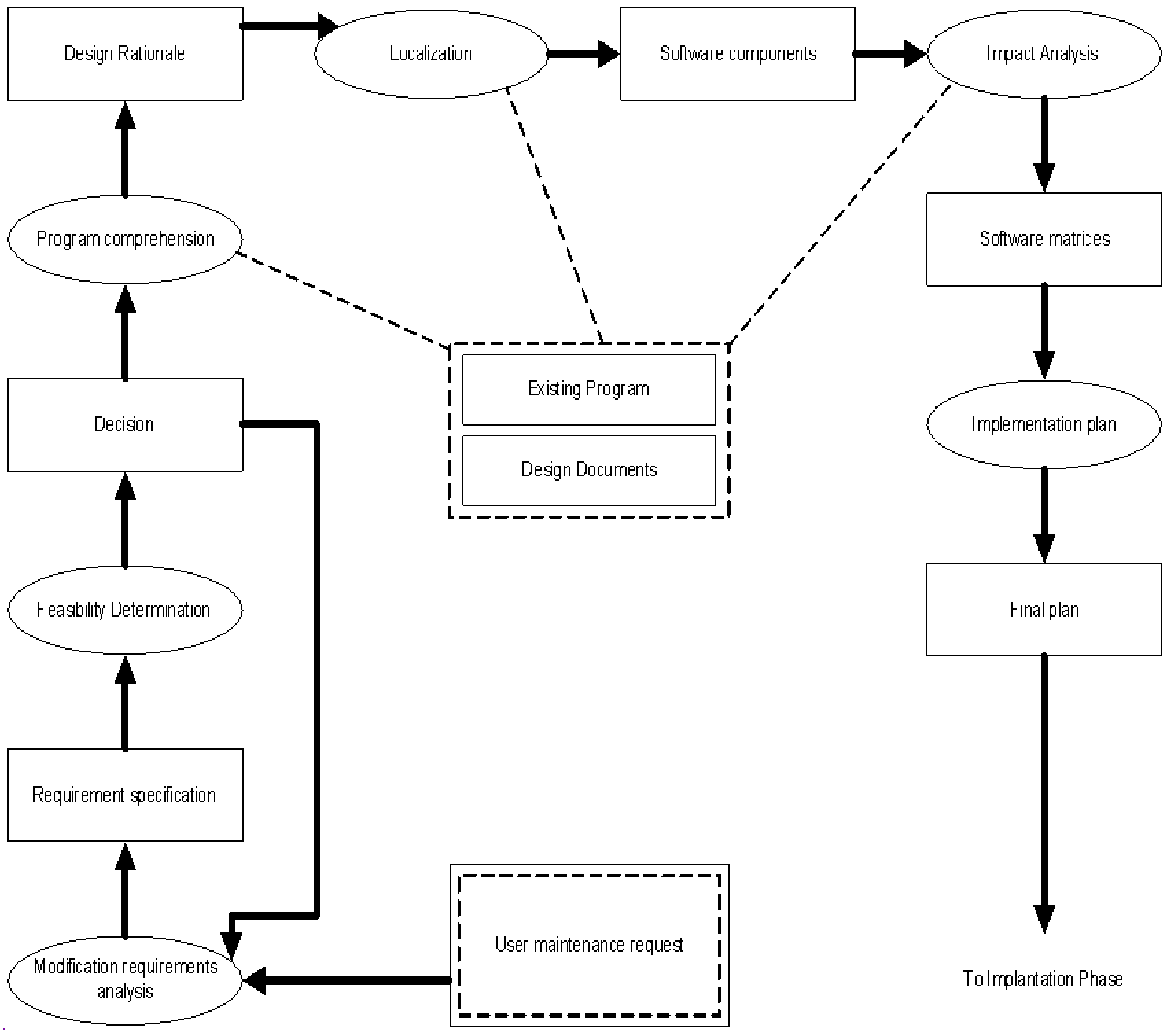

FIGURE 2. A task oriented software maintenance model [11]

A methodology for software maintenance developed by Yau [12] also explicitly deals with ripple effect as shown in Figure 3.

The fourth step in this methodology is accounting for ripple effect that is mainly focused on identifying the scope of effect as a consequence of modifications.

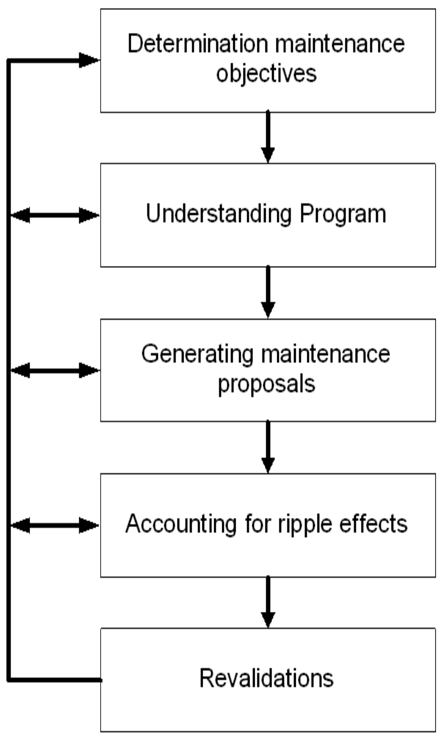
Determination maintenance objectives
Understanding Program
Generating maintenance proposals
Accounting for ripple effects
Revalidations

FIGURE 3. A methodology for software maintenance [12]

Pfleeger and Bohner model (cf. Figure 4) for software maintenance [2] also explicitly deals with ripple effect. This model mainly focuses on the usage of metrics for maintenance management.

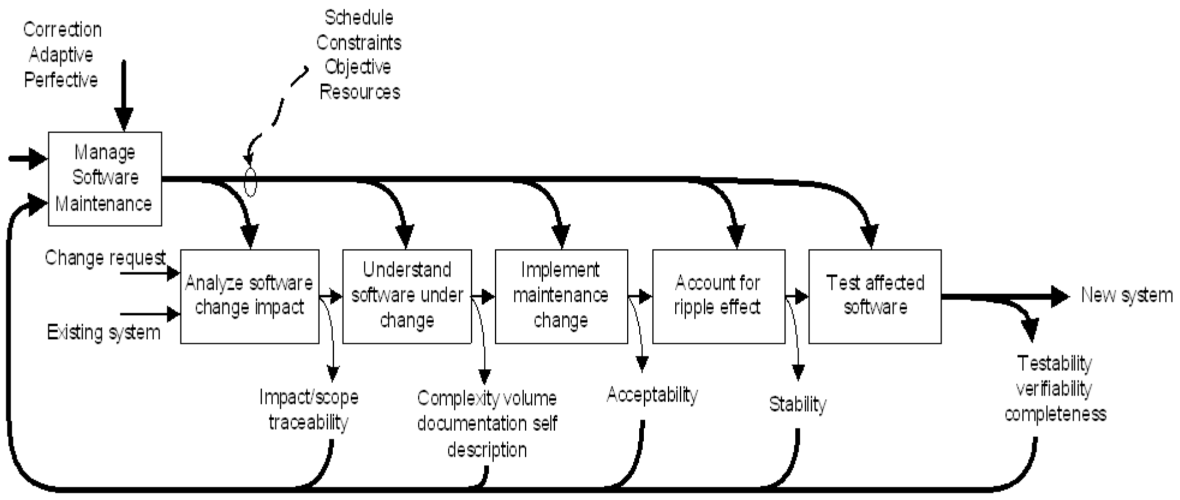
Correction
Adaptive
Perfective
Schedule
Constraints
Objective
Resources
Manage Software Maintenance
Change request
Existing system
Analyze software change impact
Understand software under change
Implement maintenance change
Account for ripple effect
Test affected software
New system
Impact/scope traceability
Complexity volume documentation self description
Acceptability
Stability
Testability verifiability completeness

FIGURE 4. SADT diagram of software maintenance activities

This model consists of six steps and each and every step has explicit feedback path. The feedback paths indicate certain factors which are evaluated by the management to decide for the next activity. The first activity, i.e., managing software maintenance uses the feedback to control the sequence of the activities and select the next appropriate action. The second activity, i.e., analyzing change impact analysis explicitly evaluates the effects of a proposed change. All these approaches deal with ripple effect analysis but at the later stages of the software development. However, none of these deals ripple effect analysis at software architecture level. So, we decide to target this area and we have developed a methodology that caters to ripple effect analysis at early stage (i.e., software architecture).

The contribution of this paper is to propose a methodology for software maintenance ripple effect analysis at software architecture level. The method requires software architecture as input. Once software architecture is at hand, it can be evaluated with respect to some quality attribute and can be tuned in case of undesirable result with respect to some specific quality attribute. This tuning will not be without ripple effect and this methodology is used to identify different components that will be affected directly or indirectly as a result of ripple effect.

The remainder of this paper is organized as follows: Section 2 presents related work; architecture based software maintenance ripple effect analysis is presented in Section 3; strength and limitations of the proposed technique are discussed in Section 4; the paper is concluded in Section 5.

2. Related Work.

The software engineering researchers have developed various techniques/methods to evaluate/measure software maintenance ripple effect. F. M. Haney [1] described a technique called "Module Connection Analysis". The technique was based on applied probabilities and elementary matrix algebra to estimate the total number of changes required to stabilize a system. S. S. Yau et al. [6] developed a technique to analyze ripple effect from both functional and performance perspectives S. Black [9] reformulated Yau and Collofello's ripple effect algorithm and its validity within the software maintenance process.

Myers [13] used matrices as a tool to quantify matrix independence. A complete dependence matrix was formulated to describe dependencies between modules within a system. N. L. Soong [14] used connectivity matrix and random markovian process to present high level quantitative measure of the information structure. Canfora et al. [15] proposed a method that identifies the side effects that can be introduced to modules from system modifications. S. S. Yau and J. S. Collofello [16] developed an algorithm to compute design stability measure. J. K. Joiner and W. T. Tsai [17] developed Data-centered Program Understanding Tool Environment (DPUTE) using ripple effect analysis with dependence analysis and program slicing. Ajila has presented a knowledge based model for a tool that can be used for impact analysis and propagation of change. [18] C. Mao et al. have used component dependence matrix for change impact analysis in component-based software system. [19]. Bohner has used change impact analysis concept for distributed applications [20].

3. Architecture Based Software Quality Maintenance Ripple Effect Analysis.

The architecture based software quality maintenance ripple effect analysis process can be visualized in Figure 5.

The proposed technique uses a "mapping" between various components of the system as designed in software architecture and software requirements. Requirements are classified as either functional or non-functional requirements. Software architecture needs to optimally meet both these kind of requirements in order to be considered satisfactory. Non-functional requirements are constraints that are placed upon the system. Non-functional requirements are also called quality of service (QoS) requirements. These represent various attributes of the system, i.e., reliability, maintainability, security, performance, availability, etc. With the completion of software architecture, the software architect/architecture team configures these components based upon system qualities.

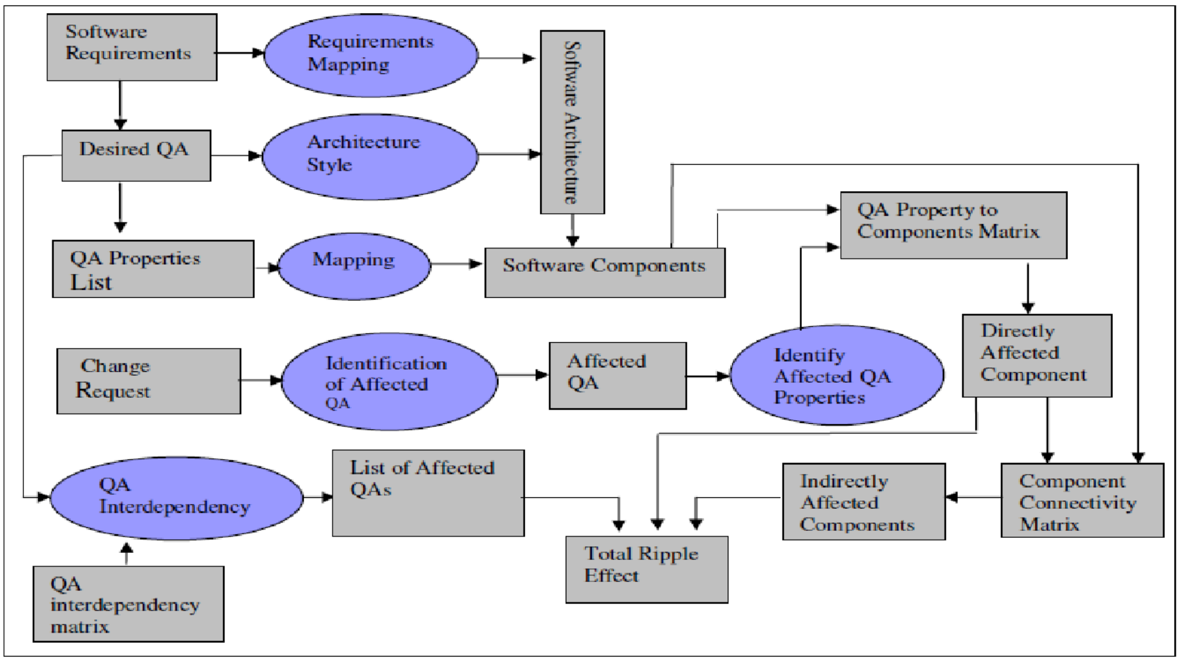

FIGURE 5. Architecture based software quality maintenance ripple effect analysis process

TABLE 1. Performance properties

QA	Properties
Performance	1) The ability of the module to execute in parallel with another module.
	2) For each resource in contention, the relative time that the module releases the resource.
	3) For each resource in contention, the relative time that the module seizes the resource.
	4) The relative time that the module begins execution.
	5) The relative time that the module transmit message to another module.
	6) The execution time of the module.
	7) For each resource utilized in the module, the resource utilization by the module.
	8) For each dependent iterative structure in the module, the number of iterations.
	9) For each data structure, the storage and retrieval times for entries in the data structure.
	10) The rate of input to the module.
	11) For each data structure, the number of entries in the data structure.
	12) For each data structure, the service time of an entry in the data structure, i.e., the relative time that an entry remains in the data structure before being serviced.
	13) For each dependent iterative structure in the module containing overhead incurring references, the number of iteration.
And two more Properties that also describe performance are	
	14) Response time.
	15) Throughput.

During development or at the time of maintenance, emergence of change requests is a routine matter. Ripple effect analysis process starts with the approval of change request. In order to calculate ripple effect, we first need to study the proposed change carefully. This change statement points towards the specific quality attribute and its related property which needs to be optimized in order to make this change happen. Detailed description of various quality attributes (QAs) and their associated properties are given in on of the subsections below. These properties are related to any number of components within the software architecture. Any type of modification to any property will have effects on component/components associated with it as well as components indirectly associated with it. In other words, when architecture tuning is required for some quality maintenance, it means that corresponding QA properties and components associated with those properties will undergo that change.

Once it is decided that the architecture should be tuned for a specific quality attribute, the next step is to identify properties of the corresponding QA. Each and every QA can be described using certain properties that describe key aspects of the corresponding QAs. These quality attributes are organized in the form of quality models, like McCall [21], Boehm [22], FURPS/FURPS+[21] and ISO/IEC 9126 [21,26]. It is clear from the study of these quality models and literature that there is no universal list of quality attributes. So, we took a sample list of QA and used it for attributes definition. The sample list includes performance and reliability. For performance the following 13 attributes were identified in Table 1 [6].

For reliability, we have listed the following 9 Properties.

Table 2. Reliability properties

QA	**Properties**
Reliability	1) Input (Validates/Invalidated) to the module
	2) Failure rate of the module
	3) Module/Service failure
	4) Mean time to failure
	5) Mean time to repair
	6) Failure impact of the module (i.e., system wide vs. local)
	7) Failure extent (complete vs. partial)
	8) Extent of recovery (complete vs. partial)
	9) Recovery mechanism (redundancy, check pointing)

Once each and every QA is described by its corresponding properties, the next step is to identify dependency of any property on various components of the system. This is achieved by creating a property-component associativity matrix. We have used Adjacency matrix AMnxm to represent the association between QA properties and associated components. We have named it "property to component associativity matrix". A representative "property to component associativity matrix" is shown in Table 3.

Table 3. Property to component associatively matrix

AMnx =

	C1	**C2**	**C3**
P1	1	0	0
P2	0	0	1
P3	1	1	0
P4	1	1	1

Software architecture consists of components and relations as mentioned above. Any type of modification to component/components may have ripple effect in other components. For this purpose, we can use component connectivity matrix CCMnxn to represent the component interaction. A representative connectivity matrix is shown in Table 4.

Software architecture is designed based upon certain quality attributes. These quality attributes are interdependent and every maintenance/design decision has potential impact on other quality attributes [23]. Therefore, in the third stage, when architecture tuning is required for some quality maintenance, associated quality attributes will also be affected.

Table 4. Components connectivity matrix

CCMnxn =

	C1	**C2**	**C3**	**C4**	**C5**
C1	0	0	1	1	1
C2	0	0	1	0	0
C3	1	1	0	1	0
C4	0	1	1	0	0
C5	1	0	1	0	0

We have used quality attributes dependency matrix QADnxn to represent the QA inter-dependency. In QAD matrix, QA are represented by rows and columns. If a QAi directly affects another AQj then QAD$[i, j] = 1$. If a QAi inversly affects another AQj then QAD$[i, j] = -1$. If a QAi does not affect another AQj then QAD$[i, j] = 0$. Supose QA1, QA2, QA3, QA4 and QA5 are quality attributes, then the corresponding QAD matrix may be represented as follows:

Table 5. Quality attributes inter-dependency matrix

QADnx =

	QA1	**QA2**	**QA3**	**QA4**	**QA5**
QA1	0	0	1	−1	1
QA2	0	0	1	0	0
QA3	1	1	0	−1	0
QA4	−1	0	−1	0	0
QA5	1	0	0	0	0

Summarizing this whole process, when the software architecture tuning is required with respect to some QA, its corresponding property/properties and associated component/s will undergo that change. Table 3 will identify the directly affected components and Table 4 will identify indirectly affected components. Table 5 can be to explore whether the QA has direct or inverse or No affect on other QAs.

4. **Web Content Extraction Application Architecture: Case Study.** In this section, we shall illustrate the application of our proposed technique on a real life application as case study. This case study will demonstrate the relevance and effectiveness of our proposed approach.

A. Application Introduction:

The system we used to illustrate the technique is web content extraction system “S. Anwar et al. (2009)”. Web is a huge growing repository of updated information available for public use. The aim of Content Extraction (CE) is to extract relevant content from the web. Due to rapid development in web technologies and absence of de-jure standards, the task of content extraction requires intricate delicacy. Emergence of Web 2.0 technologies has complicated the task of content extraction a bit more. Content Extraction application contains many components, and these components frequently get changed or replaced with the passage of time. The frequency of change in Content Extraction applications is usually greater than other applications, due to the fact that web is undergoing frequent changes and new tools and technologies are emerging from research to enhance the user experience. This makes the issue of maintenance more important in the context of Content Extraction Applications. Figure 6 represents architecture for a content extraction application.

The first layer is the interface layer which is used to interact with the user and provides web service interface to others applications. Web UIs are commonly used in modern applications due to their ease of use. However, this may not be the case for all Contents

Exaction Systems, some may use Windows based UI. In some cases, Contents Exaction System may not have a UI, for example, if used as part of crawler or another automatic application. The controller module on this layer coordinates the communication with layer below and above. The second layer is most important layer of application. It is used to extract relevant contents. It takes a request from UI and encodes and passes it on to the HTTP requester. The form request emulator is used to get the content behind a form by emulating a form request HTTP response gets the raw data and handover it to the DOM parser; the DOM parser uses the JavaScript Engine to execute any client side code. This step insures the retrieval of Web 2.0 content.

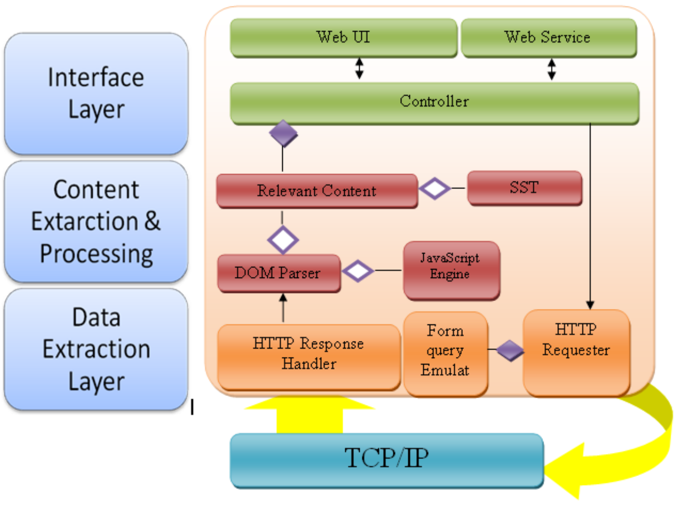

FIGURE 6. Web content extraction application architecture

The resultant DOM is passed on the Relevant Content Extractor which uses different algorithms to extract relevant content. The resultant relevant content is passed to UI for rendering on screen. The data extraction layer is used to handle HTTP related issues and errors. HTTP requester is used for making an HTTP request for the resource, while the HTTP response handler handles the HTTP Response. It waits for incoming HTTP response from a Web Server and removes the HTTP header from the message and passes on HTML to the DOM parser.

B. Problem Statement:

One of the observed problems with the above mentioned system is that it involves an overhead incurring reference/page faults that ultimately leads to low productivity. On further analysis, it was found that this specific issue related to the performance quality attribute. More specifically, it is related to the performance's property "For each dependent iterative structure in the module containing overhead incurring references, the number of iteration".

C. Ripple Effect Analysis

In order to minimize the incurring references, it was decided to tune the architecture. However, tuning the architecture with respect to incurring references will not be without ripple effect. From Table 6, we were able to extract those components which were directly associated with the specific property given above. As a result, the extracted components included "HTTP request handler" and "HTTP response handler."

In the next phase, we were able to get further details of ripple effect that may be resulted by tuning of this property. Using Table 7, we successfully identified all the components associated with directly affected components. The application of our technique shows that DOM parser, Form Query Emulator and Controller are the indirectly affected components.

Using QA interdependency matrix as presented in Table 8, we are able to identify that any change in performance QA will have serious and possibly negative impact on several other QA including flexibility, portability and reusability, etc.

Table 6. Properties to components associativity matrix for performance

Components / Performance Features	Web UI	Web Service	Controller	HTTP Requestor	Form Query Emulator	HTTP Response Handler	DOM Parser	Java Script Engine	Relevant Content Extractor	SST
Parallel Execution	0	0	0	0	0	0	0	0	0	0
The relative time that the module releases the resource	0	0	0	1	0	1	0	0	0	0
The relative time that the module seizes the resource	0	0	0	1	0	1	0	0	0	0
The relative time that the module begins execution	1	1	1	1	1	1	1	1	1	1
The relative time that the module transmit message to another module	1	1	1	1	0	1	1	0	1	0
The execution time of the module	1	1	1	1	1	1	1	1	1	1
The resource utilization by the module	0	0	0	1	0	1	0	0	0	0
For each dependent iterative structure in the module, the number of iterations	0	0	0	0	0	0	1	1	0	1
The storage and retrieval times for entries in the data structure	0	0	0	1	0	1	1	1	0	0
The rate of input to the module	1	1	1	0	0	1	1	1	1	1
The number of entries in the data structure	0	0	0	1	0	1	1	1	0	0
The service time of an entry in the data structure	0	0	0	1	0	1	1	1	0	0
For each dependent iterative structure in the module containing overhead incurring references, the number of iteration	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**
Response time	1	1	1	1	1	1	1	1	1	1
Throughput	1	1	1	1	1	1	1	1	1	1

Table 7. Component connectivity matrix

Components	**Web UI**	**Web Service Emulator**	**Controller Handler**	**HTTP Requester**	**Form Query Engine**	**HTTP Response Extractor**	**DOM Parser**	**Java Script**	**Relevant Content**	**SST**
Web UI		0	1	0	0	0	0	0	0	0
Web Service	0		1	0	0	0	0	0	0	0
Controller	1	1		**1**	0	0	0	0	1	0
HTTP Requester	0	0	0		1	0	0	0	0	0
Form Query Emulator	0	0	1	**1**		0	0	0	0	0
HTTP Response Handler	0	0	0	0	0		1	0	0	0
DOM Parser	0	0	0	0	0	**1**		1	1	0
Java Script Engine	0	0	0	0	0	0	1		0	0
Relevant Content Extractor	0	0	1	0	0	0	1	0		1
SST	0	0	0	0	0	0	0	0	1	

TABLE 8. QA inter-dependency matrix [24]

	C	**A**	**Ch**	**S**	**Ts**	**F**	**Pt**	**R**	**I**	**T**	**E**	**Ct**	**Pd**	**Rl**	**Tl**	**Ip**
Analyzability	0															
Changeability	0	1														
Stability	0	1	1													
Testability	0	0	0	0												
Flexibility	0	1	1	1	0											
Portability	0	1	1	1	0	1										
Reusability	0	0	0	0	0	0	1									
Interoperability	0	0	0	0	0	1	1	1								
Time (Lead Time)	–1	–1	–1	–1	–1	–1	–1	–1	–1							
Effort (Cost)	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1						
Content	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1					
Productivity	–1	–1	–1	–1	–1	–1	–1	–1	–1	0	0	0				
Reliability	1	0	0	0	0	0	–1	0	0	–1	–1	–1	–1			
Tailor ability	0	–1	–1	–1	–1	1	–1	0	1	–1	–1	–1	–1	–1		
Interactive Performance	0	0	0	0	0	–1	–1	–1	0	–1	–1	–1	–1	0	–1	
Usability	0	0	0	0	0	0	0	0	0	–1	–1	–1	–1	0	0	0

C = Correctness, A = Analyzability, Ch = Changeability, S = Stability, Ts = Testability, F = Flexibility, Pt = Portability, R = Reusability, I = Interoperability, T = Time (Lead Time), E = Effort (Cost), Ct = Content, Pd = Productivity, Rl = Reliability, Tl = Tailor ability, Ip = Interactive Performance.

5. **Conclusions.** With the passage of time, software intensive systems are becoming large and more complex which affects quality of the system. This ultimately emanates the need for Software quality maintenance and it is also a well known fact that software maintenance is not without ripple effect. On the other hand, software architecture has also considerable effects on quality factors such as reliability, performance and maintainability, etc. Using software architecture for software quality maintenance ripple effect analysis would help to identify the scope of the ripple effect. We have developed architecture based quality maintenance ripple effect analysis method. We have developed a technique that can identify directly and indirectly affected components. This technique can be automated that will help maintenance programmers to identify the side effects as a result of quality maintenance.

A direction we intend to pursue is to provide a quantitative measure of the maintenance programmer effort as a result of quality maintenance.

REFERENCES

[1] F. M. Haney, Module connection analysis – A tool for scheduling of software debugging activities, *Proc. of the AFIPS Fall Joint Computer Conference*, Reston, VA, pp.173-179, 1972.

[2] S. L. Pfleeger and S. A. Bohner, A framework for software maintenance metrics, *Proc. of Conference on Software Maintenance*, Los Alamitos, CA, pp.320-327, 1990.

[3] S. S. Yau and J. S. Collofello, Some stability measures for software maintenance, *IEEE Transactions on Software Engineering*, vol.6, no.6, pp.545-552, 1980.

[4] S. Anwar, M. Ramzan, A. Rauf, A. Jaffer and A. A. Shahid, A novel approach for architecture based software maintenance prediction. *International Journal of Innovative Computing, Information and Control*, vol.7, no.6, pp.3172-3193, 2011.

[5] G. Alkhatib, The maintenance problem of application software: An empirical analysis, *Journal of Software Maintenance – Research and Practice*, vol.4, no.2, pp.83-104, 1992.

[6] S. S. Yau, J. S. Collofello and T. MacGregor, Ripple effect analysis of software maintenance, *Proc. of Computer Software and Applications Conference*, Piscataway, NJ, pp.60-65, 1978.

[7] B. Williams and J. Carver, Characterizing changes to assess architectural impact, *Technical Report MSU-070115*, Mississippi State University, 2007.

[8] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practise*, 2nd Edition, Addison Wesley, 2003.

[9] S. Black, Computing ripple effect for software maintenance, *Journal of Software Maintenance and Evolution: Research and Practice*, vol.13, pp.263-279, 2001.

[10] IEEE Std. 1219-1998, *Standard for Software Maintenance*, IEEE Computer Society Press, CA, 1998.

[11] M. K. Khan, M. A. Rashid and W. N. Lo Bruce, A task-oriented software maintenance model, *Malaysian Journal of Computer Science*, vol.9, no.2, pp.36-42, 1996.

[12] S. S. Yau, R. A. Nicholl, J. J.-P Tsai and S. Liu, An integrated life cycle model for software maintenance, *IEEE Transactions on Software Engineering*, vol.14, no.8, pp.1128-1144, 1988.

[13] G. J. Myers, *A Model of Program Stability. Composite/Structured Design*, Van Nostrand Reinhold, NY, 1975.

[14] N. L. Soong, A program stability measure, *Proc. of the 1977 Annual ACM Conference*, NY, pp.163-173, 1977.

[15] G. Canfora, G. A. D. Lucca and M. Tortorella, Controlling side-effects in maintenance, *Proc. of the 3rd International Conference on Achieving Quality in Software*, Champanand Hall, London, UK, pp.89-102, 1996.

[16] S. S. Yau and J. S. Collofello, Design stability measures for software maintenance, *IEEE Transactions on Software Engineeing*, vol.11, no.9, pp.849-856, 1985.

[17] J. K. Joiner and W. T. Tsai, Ripple effect analysis, program slicing and dependence analysis for software maintenance, *Technical Report TR 93-84*, University of Minnesota, 1993.

[18] S. Ajila, Software maintenance: An approach to impact analysis of object change, *Software: Practice and Experience*, vol.25, no.10, pp.1155-1181, 1995.

[19] C. Mao, J. Zhang and Y. Lu, Using dependence matrix to support change impact analysis for CBS, *The 5th International Conference on Computational Science and Applications*, pp.192-197, 2007.

[20] S. A. Bohner, Extending software change impact analysis into COTS components, *Proc. of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop*, pp.175-182, 2003.

[21] K. K. Wnukiewicz, *The Role of Quality Requirements in Software Architecture Design*, Master Thesis, School of Engineering Blekinge Institute of Technology, 2006.

[22] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod and M. J. Merritt, *Characteristics of Software Quality (TRW Series of Software Technology)*, North-Holland Pub. Co., 1978.

[23] S. T. Alben, *The Art of Software Architecture: Design Methods and Techniques*, Wiley Publishing, Inc., Indianapolis, Indiana, 2003.

[24] *http://www.bth.se/tek/besq.nsf/(WebFiles)/B339B908DC4E9D2BC12570690032868B/$FILE/chapter_5.pdf.*

[25] J. Ren, C. Hu, K. Wang and R. Ma, A method for discovering security bugs of software based on AOE network, *ICIC Express Letters*, vol.3, no.4(A), pp.1081-1086, 2009.

[26] L. Lin and H.-M. Lee, A new algorithm of software quality evaluation for user satisfaction, *International Journal of Innovative Computing, Information and Control*, vol.4, no.10, pp.2639-2647, 2008.

[27] Y. Liu, C. Wei and S. Gao, Research on software reliability modeling based on stochastic petri nets and module level software rejuvenation, *ICIC Express Letters*, vol.3, no.3(B), pp.607-613, 2009.