An Efficient Approach for Requirement Volatility Identification

Md. Faisal
Dept of Computer Application
Integral University
Lucknow, India

Md. Rizwan Beg, Ph.D
Dept of CSE
Integral University
Lucknow, India

Halima Sadia
Dept of IT
Integral University
Lucknow, India

ABSTRACT
Software Development is a dynamic activity. Requirements of the stakeholders keep on changing throughout the lifecycle of a project development. These Volatile requirements are considered as a major risk factor during system development in the software industry. Requirement Volatility is found to be having a significant impact on schedule & cost overrun in software projects. This paper discusses an approach for identifying volatile requirement and proposes an algorithm to automate the identification of volatile requirements in a requirement document at an early stage so that preventive measures can be taken. Also, we are proposing an approach to analyze the dependencies among various requirements which can be very lucrative in requirement management especially in the presence of volatile requirements.

General Terms
Software Engineering, String Matching, Dependency Analysis

Keywords
Software Development, Requirement Volatility, Inspection Process

1. INTRODUCTION
Requirements collection is an early phase of the software development life cycle. Requirements are the foundation of the software development [1]. They provide the basis for cost estimation and for planning project schedules as well as for designing and testing specifications [2]. The success of a software project, both functionally and financially, is directly related to the quality of its requirements [3]. Constant changes to requirements during development life cycle significantly contribute to the quality of requirements specification [4] [5]. Software development is a dynamic process therefore we cannot expect requirements to be fixed. The problem is not with changing requirements; the problem is with inability to predict volatility and inadequate approaches for dealing with requirements volatility. The negative impact of volatility can be minimized if it can be identified at its earliest. It is more important to identify its causes at early stages. Once causes are detected, volatility can be eradicated and may not go forward in the further phases. Providing complete, consistent, stable, unambiguous, and correct documents throughout the life cycle improves the chances for a higher quality software system [6]. Requirements volatility must be considered as a part of project risk assessments [7].

Therefore, checking software documents for volatile requirements before proceeding to the next development phase contributes to overall system quality. Inspection is a well accepted method for the cost effective detection & correction of defects [8]. These techniques can be used to identify volatile requirements at an early age.

2. INSPECTION TECHNIQUE
Inspection is a best known way of detecting defects in SRS [8][9]. Initially created for source code, Software inspections have been extended for the intermediate products of the software life cycle such as design & requirements specification [10].Requirements are the key for any software project. Requirements keep on changing throughout the life cycle of a project [1]. These volatile requirements can be very dangerous if they are not handled properly and detected in earlier phases of life cycle.
The inspection technique consists of two vital components; (i) Participants, (ii) Requirement Inspection Process. In Volatility Identification Process (VIP) five participants will play vital role through executing their individual responsibilities to identify risks. These participants are (i) Moderator (ii) Author (iii) Reader (iv) Inspector (v) Recorder. After assigning the requirement inspection task to the participants, they are required to follow an appropriate method assigned to them for inspecting the requirements document.

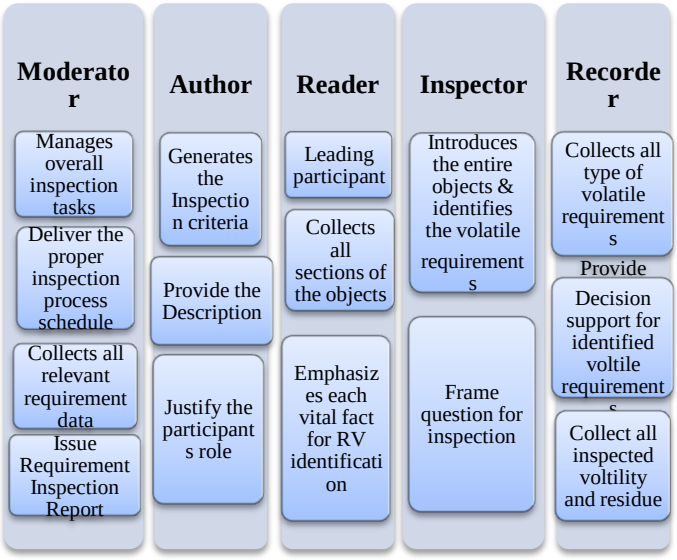

Fig1: Inspection Participants

2.1 Requirement Inspection Process (RIP):
The requirement inspection process consists of five consecutive steps:
Preparation is a key stage in which members of the inspection team prepare individually for the inspection by reviewing and finding potential volatility in the requirements.

Inspection Meeting where team members as a group review the product to find, categorizes, and record volatile

requirements. Potential RV found by individuals is discussed during the actual inspection meeting.

Third Hour is optional additional time, apart from the inspection meeting, that can be used for discussion, possible solutions.

Rework Stage is required when there are a large number of changes. The moderator and the team decide the necessity for reinspection at the end of the inspection meeting.

Reinspection is required when there are a large number of volatile requirements. Reinspection allows the changes to the product to be reviewed by the entire team instead of just the moderator. The moderator and the team decide the necessity for reinspection at the end of the inspection meeting.

Follow Up: Short meeting between the moderator and author. It determines whether RV found during the inspection meeting have been eliminated.

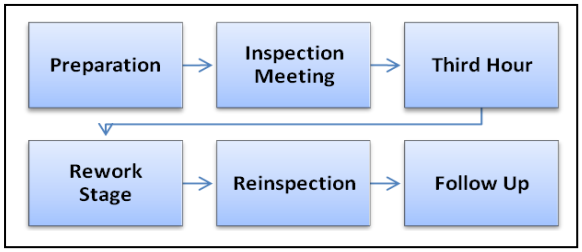

Fig 2: Inspection Process

2.2 Volatility Identification Algorithm

An algorithm for identification of volatile requirements at an early stage has been proposed. Initially requirements are kept into a requirement repository with its unique identifier. An open ended volatile data dictionary will be used for the same. Aho-Corasick Technique [11] for string matching is used to identify these types of requirements against a given volatile data dictionary.

2.2.1 Structures Used

ITECHNIQUE_VOLAREQ (PI, PAI, R, VOLA_D, VOLA_k)

Process_Inspection : Processes involved during Inspection for a set of Requirements R

PI = {P, IM, TH, RS, RI, FP}

Where P is for Preparation, IM is for Inspection Meeting, TH is for Third Hour, RS is for Rework Stage, RI is for Reinspection, FP is for Followup

Participants_Inspection : Participants involved during Inspection for a set of Requirements R

PAI = {M, A, R, I, RC}

Where M is for Moderator, A is for Author, R is for Reader, I is for Inspector, RC is for Recorder

R : A Non-Empty set of Requirements

$R = \{R_1, R_2, ..., R_n\}$ where i = 1 to n

VOLA_D : Predefined Volatility Data Dictionary

$VOLA\_D = \{V_1, V_2, ..., V_k\}$ where j = 1 to K

VOLA_$\mathcal{K}$: A Volatility Keyword Tree or TRIE for a set of patterns $\mathcal{VP}$ from the VOLA_D

$\mathcal{VP} = \{VP_1, VP_2, ..., VP_k\}$, such that

If $V_j \in$ VOLA_D and $VP_j \in \mathcal{VP}$

Then $V_j = VP_j \ \forall \ \mathcal{VP} \in$ VOLA_$\mathcal{K}$

VOLA_R : Identified Volatile Requirement Set consisting of Volatile Requirements with their respective unique Identifiers

$VR = \{VR_1, VR_2, ..., VR_m\}$

Aho_Corasick(R, VOLA_$\mathcal{K}$) : is a Volatility Matching Function which matches input Requirement R w.r.t predefined Volatility Data Dictionary Where

VOLA_R = Aho_Corasick(R, VOLA_$\mathcal{K}$)

The Algorithm uses Aho_Corasick string matching Algorithm. It is a kind of dictionary matching algorithm that locates elements of a finite set of strings ('dictionary') within an input text.

2.2.2 Algorithm: It has two phases:

Phase 1 : Construction of Volatility Keyword TRIE (VOLA_k)

Construct the Volatility Keyword TRIE for VP = {VP_1, VP_2,....., VP_k}

Begin with a root node only;

Insert each pattern VP_k , one after the other as follows :

Starting at the root, follow the path labelled by characters of each respective VP_k ;

- Build the path by adding new edges with respective labels and nodes for each VP_k .
- Add identifier k of each VP_k at the terminal node of each completed pattern in the TRIE .

Phase 2 : Aho-Corasick Automation

Actions are determined by three functions:

Step 1 : the GOTO function g(R_i , VP_k) *//give the states matched by R_i and target VP_k*

Step 2 : the Failure function fail(R_i) *//give the state entered at a mismatch*

Step 3 : the Output function out(R_i) *//give the state entered when R_i matches target VP_k*

States : nodes of the Volatility keyword TRIE

Initial State : 0 = the root

```
q := 0; // initial state (root)
for i := 1 to n do //for all requirements
for j := 1 to k do //for all patterns
while ( g(R[i], VP[j]) = null ) ; do
q := f(R[i]); // follow a fail
q := g(R[i], VP[j]); // follow a goto
if out(R[i]) ≠ null ; then print i, out(R[i]);
endfor;
```

Phase2 : Lookup of a Requirement R = {R_1, R_2, ... R_n} in Volatility Keyword TRIE(VOLA_$\mathcal{K}$)

```
Count = 0 ; // Count of Volatile Requirements
    For all R[i] ∈ R and for i = 0 to n step 1
    For all VP[j] ∈ VP and for j = 0 to k step 1
```

Starting at root follow the path labelled by characters of VP as long as possible;

- If the path leads to a node with an 'identifier', R_i is a keyword in the dictionary and hence add R_i to the identified Volatile Requirements VOLA_R

Count++ ;

VR = VR ∪ R_i ; // Add Requirement in Volatile Requirement Set

- If the path terminates before VP is reached, the requirement is not in the TRIE and hence is not volatile.

The complexity of the above algorithm is linear in the length of the pattern plus the length of the searched text plus the number of output matches. It constructs a finite state machine that resembles a trie with additional link between the internal nodes.

3. REQUIREMENT DEPENDENCY ANALYSIS

Requirement Dependency Analysis is the key process for identifying the dependencies among requirements. Once volatile requirements have been identified, dependencies among requirements are very vital to determine as it can be very useful to understand the behaviour of requirements in presence of changes. It is a step by step process which takes requirements as input, store them in Requirement Repository and generate the dependency level as output. This dependency level can be used to determine a Requirement Volatility Threshold (RVT). The value of RVT is used to decide the strategies to deal with the requirement volatility.

Step1:

Enter the collected requirements into the Requirement Repository R. Requirement Repository is simply a set of requirements (R_i, R_j, R_k,……. R_n)

Any requirement R_i is characterized by the following characteristic set:

{RId, DepLevel, ModLevel, DepSet}

Where RId is the unique identifier for the requirement

Dependency level (DepLevel): is the level up to which the requirement is depending upon other ones.

DepSet: is the set of requirements R_i is depending upon.

DepSet (R_i) = **{R_j, R_k..... R_{n-1}}**

ModLevel: is the set of possibilities of modification. A requirement can be deleted or modified or a new requirement can be added at any time. So their modification level must be recorded at the time of requirement entry in the requirement repository. The ModLevel along with DepLevel determines the effect of changing the requirements at any time instance.

ModLevel (R_i) = {Major, Moderate, Minor, Null}

Step 2:

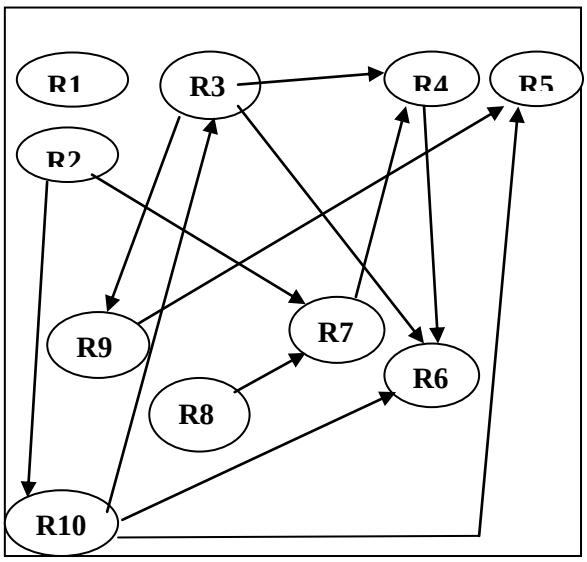

Fig 3: Requirement Dependency Graph

Plot the Dependency Graph (DG) for the requirement repository. It is a simple graph in which requirements are represented as vertices and their relationship as edges. If a requirement (R_i) is depending on requirement (R_j), corresponding edge Ri → R_j must be present in the graph.

Step 3:

Generate the Requirement Dependency Matrix (RDM). The dependency matrix is used to determine the dependencies among the requirements which can be very useful while changes are made in any requirements. It is an N*N matrix. It has three values {0, 1and -1}.It is used to calculate the DepLevel.

- **Deg ($R_{i(out)}$)** is the no of Positive entries in any row that specifies up to which extent the requirement is depending on another requirement.
- **Deg ($R_{i(in)}$)** is the Negative entry (-1) that specifies up to which extent another requirement is depending on it.
- Zero entry represents that there is no dependency on each other.
- No of negative entries in any requirements field specifies that up to how extent that requirements is being used as a **baseline requirement** for other requirements.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
R1	0	0	0	0	0	0	0	0	0	0
R2	0	0	0	0	0	0	1	0	0	1
R3	0	0	0	1	0	0	0	0	1	-1
R4	0	0	-1	0	0	1	-1	0	0	0
R5	0	0	0	0	0	0	0	0	-1	-1
R6	0	0	-1	-1	0	0	0	0	0	0
R7	0	0	0	1	0	0	0	-1	0	0
R8	0	0	0	0	0	0	1	0	0	0
R9	0	0	1	0	-1	0	0	0	0	0
R10	0	-1	1	1	-1	0	0	0	0	0

Fig 4: Requirement Dependency Matrix

Any change in this particular requirement will lead to make changes in its depending requirement.

Step 4:

Calculate the DepLevel for each element of requirement repository. DepLevel is calculated by the variable DepDeg (R_i).

DepDeg(R_i) is of two types : **Deg ($R_{i(in)}$) OR Deg($R_{i(out)}$)**

We are interested in Deg ($R_{i(out)}$) because it contains enough information to justify the DepLevel.

Deg ($R_{i(in)}$) is used to determine the no of dependants.

Step 5:

After calculating DepLevel we get a value that is generated by Dependency Level Range. Dependency Level Range is calculated by:

DepLevelRange = Deg ($R_{i(out)}$) / R_i * 100

We are providing a Dependency Level Range to determine DepLevel.

DepLevel:={Very Low, Low, Nominal, High, Very High}

Table 1: Dependency Level Range

Dependency Level Range	Dependency Level (DepLevel)
0-6%	VERY LOW
6-12%	LOW
12-24%	NOMINAL
24-48%	HIGH
>48%	VERY HIGH

Step 6:
Calculate and set a Requirement Volatility Threshold (RVT) which determines up to which extent the volatility must be allowed.
The value is RVT is calculated as
RVT (RequirementVolatiltyThreshold | DepLevel, ModLevel) = TriangularDist (RequirementVolatiltyThreshold, DepLevel, ModLevel)
TriangularDist is the Continuous probability distribution with a probability density function shaped like a triangle. It is defined by three values: the minimum value *a*, *the maximum value b, and the peak value c.*
Beyond the value of RVT the volatility management schemes must be used to minimize the impact of these volatile requirements. However the value of RVT can be adjusted according to the scenarios.

4. CONCLUSION

Requirement volatility, defined as the change in Requirements has been reported as one of the main factors causing a project to experience challenges. In this paper we have presented an algorithm to identify the volatile requirements at an early stage. Further Dependency Analysis has been made. The Dependency analysis is very important in the view of optimizing the impact of these changing requirements. The next stage of research involves introducing fuzzy based approach for the impact analysis of the changes. The direction of further study is to develop a framework for Requirement Volatility Management for early identification and depletion of requirement .

5. REFERENCES

[1] K. E. Wiegers, “Software requirements”, Microsoft press 1999.

[2] G. Stark, A. Skillicorn, and R. Ameele, "An Examination of the Effects of Requirements Changes on Software Releases," CROSSTALK, The Journal of Defence Software Engineering, December 1998.

[3] T. Hammer, L. Huffman, and L. Rosenberg, "Doing Requirements Right the First Time," CROSSTALK, The Journal of Defence Software Engineering, December 1998, pp. 20-25

[4] D. Zowghi, and Nurmuliani, "Investigating Requirements Volatility During Software Development: Research in Progress", Proceedings of the 3rd Australian Conference on Requirements Engineering (ACRE98), Geelong, Australia, 1998

[5] D. Zowghi, R. Offen, and N. Nurmuliani, "The Impact of Requirements Volatility on Software Development Lifecycle," proceedings of the International Conference on Software, Theory and Practice (ICS2000), Beijing, China, 2000

[6] L. Hyatt and L. Rosenberg, "Software Metric for Risk Assessment," proceedings of the 26th Safety and Rescue Symposium, Risk Management and Assessment Session, Beijing, China, 1996.

[7] Lane and A. L. M. Cavaye, " Requirements Volatility Enhances Software Development Productivity," proceedings of the 3rd Australian Conference on Requirements Engineering (ACRE 98), Geelong, Australia, 1998.

[8] M. Fagan, “Advances in Software Inspections,” IEEE Trans. Software Eng., July 1986, pp. 744-751.

[9] Forrest Shull Ioana Rus, Victor Basili “How Perspective-Based Reading Can Improve Requirements Inspections” 0018-9162/00/ 2000 IEEE

[10] S. Nayak, R. Khan and R.Beg “Evaluation of Requirement Defects: An Implementation of Identification Technique” ICIET2012, Journal of Procedia Technology.

[11] Aho, Alfred V.; Corasick, Margaret J. (June 1975). "Efficient string matching: An aid to bibliographic search". *Communications of the ACM* **18** (6): 333–340