International Journal of Software Engineering and Its Applications
Vol.8, No.2 (2014), pp.51-58
http://dx.doi.org/10.14257/ijseia.2014.8.2.06

Identifying Causality Relation between Software Projects Risk Factors

Haneen Hijazi[1], Shihadeh Alqrainy[2], Hasan Muaidi[2] and Thair Khdour[2]

[1] *Software Engineering Department, Hashemite University, Zarqa, JORDAN*
[2] *Faculty of Information Technology, Albalqa Applied University, Salt, JORDAN*
{haneen}@hu.edu.jo, {alqrainy, alserhan, khdour}@bau.edu.jo

Abstract

Software development process is vulnerable to different risk factors. Several works were done in order to identify these factors and to introduce new ones. On the other hand, little research has been conducted around investigating the relationship between these risk factors. In this paper, a detailed set of software risk factors were examined. It was found that the relationship between these factors is a (cause- effect) relationship. As a result, a causality table and a cause-effect diagram are introduced to illustrate this causality relationship.

Keywords: *Causality, cause-effect, risk, risk factor*

1. Introduction

Software development process is vulnerable to different risks. Each phase of the Software Development Life Cycle (SDLC) is susceptible to several risk factors. Each of these factors threaten project success by causing major software projects risks such as time delay, budget overrun, and/or bad quality product either directly or indirectly.

First of all, we have to distinguish between two terms; risk and risk factor. In general, risk is the possibility of suffering loss [1]. The definition of risk involves two distinct terms: uncertainty and loss [2]. In software projects, this loss corresponds to the unwanted negative effects such as time delay, budget overrun, and bad quality software system [3]. Uncertainty includes events which may or may not happen [4]. The sources of the negative project effect are called “risk factors” [5].

Thus, risk factors are the uncertain conditions and influences that affect the cost, duration, and quality of the project negatively. If these factors are ignored or not mitigated well, they will present serious threats that hinder the software projects from completing successfully and achieving its goals and expected outcomes [6].

In short, we can conclude that a risk factor is a source of project risk which, in turn, can be either delay in time, loss in budget, poor quality (or any combination).

Recently, many researchers have become interested in identifying software projects risk factors. This is due to the fact that software projects risk factors change overtime, and thus regular software risk identification studies have to be conducted from time to time in order to identify more and more risk factors. In this regard, the literature is rich in such studies concerned with the identification of different software projects risk factors. Other researchers are interested in managing these factors by proposing risk management strategies, models, and methodologies that guarantee the avoidance of the majority of risks throughout the development process. None of all of these researchers can deny the importance of defining the relationship between software projects risk factors in the risk management process. Despite that, little researchers tackled this issue in their research.

ISSN: 1738-9984 IJSEIA

In this paper, similar works related to the risk identification process besides the trials to identify the relations between risk factors are described. Then the relationship among a detailed list of risk factors is examined, and the causality relationship is highlighted. Lastly, a causality table and a cause-effect diagram are introduced.

2. Related Work

As mentioned before, the literature is rich in works around the risk identification process. On the other hand, little work has been conducted around identifying the relationship between risk factors.

Keil *et al.*, [7], in 1998, classified risks into four categories upon the fact that there is a relation between the importance of risks and their perceived level of control. Then, mitigation strategies were devised to handle each type of risks instead of each risk individually. Addison and Vallabh [8], in 2002, listed a list of risk factors and controls and identified the importance of each risk factor, the frequency of occurrence for each risk factor and control, and the effectiveness of each control against each factor. In 2007, Arshad *et al.*, [9] identified forty four risk factors and classified them into six categories (*i.e.*, organizational, environment, project team, user, project requirements, project complexity, and project management).

Shahzad and Iqbal [10], in 2007, made a trial to find the most frequently occurring risks in each phase of the SDLC. Shahzad and Al-Mudimigh [11], in 2010, proposed a risk identification, mitigation, and avoidance model (RIMAM) for handling software development risks. Shahzad *et al.*, [12], in 2010, prioritized a previously defined list of risk factors according to the overall impact for each factor.

3. The Causality Relationship

In order to identify the relationship between software risk factors, an exhaustive list of risk factors was adapted [13]. This list is the harvest of a comprehensive literature survey investigated the SDLC phases, activities, problems and risks, predefined ready-made checklist and taxonomies, plus the experience of highly qualified project managers, developers, and academics interested in the domain. In this list the sources of risks in each of the five phases of the SDLC phases is defined. All of these risk factors are introduced in table 1. Moreover, each of these factors is indexed (*i.e.*, given an identification number from (1 to 100)).

After identifying the major risk factors threaten the SDLC, it would be of utmost important to understand the relationships between these factors. An in-depth study of these factors will uncover the hidden relationships between them. The examiner will notice that the dominant relationship is a (cause-effect) relationship, in which each risk factor is affected by other factors and affects others.

Causality (or the cause-effect relationship) is the relation between an event (the cause) and a second event (the effect), where the second event is understood as a consequence of the first [14]. Anything that affects an effect is a factor of that effect. Hence, it also can be defined as the relation between a set of factors (causes) and the effect.

In software projects, when we study the relationships between risk factors in the SDLC, we actually identify the causes of appearance of each risk factor in the SDLC. Any risk factor does not come out of nowhere; each risk factor can be considered as a consequence from other factors and other specific project characteristics.

In software projects, a software risk factor may directly affect the time schedule, cost, or the quality of the final product. Either, it may affect other phases creating another risk factor which may in turn affect time, cost, and quality directly or indirectly by influencing other factors, and so on. In sum, each risk factor will lastly affect cost, schedule, or quality either directly "Direct Factor" or indirectly "Intermediate Factor" through a sequence of dependent factors.

Identifying the causality between risk factors is very important. It helps project managers and developers prioritize risks; the risk factor that affects the highest number of other risks will often be given the highest priority. This would allow developers to implement the appropriate risk management strategies to handle the maximum number of potential risk factors early in the development process. Table 1 shows an indexed list of 100 risk factors, each factor is associated with the indices of the affected factors.

From another side, the best way to visually represent the causality relation between the identified risk factors is to use the Ishikawa diagrams (or Fishbone diagram). It is a diagram used to represent the cause-effect relation in management and engineering [15]. It was first introduced by Kaoru Ishikawa in 1968, and usually used to identify potential factors causing an overall effect. The fishbone diagram usually consists of the problem statement (*i.e.*, the overall effect), attached to a line (main bone) with several lines coming into this main bone represent the factors. Usually, factors are grouped into categories according to a specific criterion you decide upon, and then lines are labelled with these categories names.

Herein, the project failure is considered as the problem statement. Each factor is assigned to the specific SDLC phase it affects (*i.e.*, grouping criteria). Thus, five main lines come into the main bone which is attached to the project failure effect. Many other solid lines are used to represent the sub-factors (*i.e.*, herein, the causality within the phase). Dashed lines are used to represent the causality between factors among the phases. Figure 1 represents the cause-effect diagram for software projects risks and risk factors. This diagram shows the flow of risk factors, and how each risk factor directly or indirectly affects the cost, schedule, and/or the quality of the final product.

Table 1. Causality Table

Index	Risk Factor	Affected Factors
Phase 1: Requirements Analysis and Definition		
1	Inadequate estimation of project time, cost, scope and other resources	2,3,4,5, 29,73
2	Unrealistic Schedule	14
3	Unrealistic Budget	14
4	Unclear Project Scope	6,7,8,12
5	Insufficient resources	14,28,53
6	Unclear Requirements	8,13,17
7	Incomplete Requirements	17,94
8	Inaccurate Requirements	17
9	Ignoring the Non-functional requirements	7
10	Conflicting User Requirements	15
11	Unclear Description of the real environment	74,82
12	Gold Plating	-
13	Non-verifiable Requirements	17,30,79
14	Infeasible Requirements	17
15	Inconsistent Requirements	17
16	Non-traceable Requirements	21
17	Unrealistic Requirements	22,32
18	Misunderstood domain-specific terminology	20
19	Mis-expressing user requirements in natural language	20

20	Inconsistent requirements data and RD	22
21	Non-modifiable RD	92
Phase 2: Design		
22	RD is not clear for developers	23,33
23	Improper AD method choice	24
24	Improper choice of the PL	23,45
25	Too much complex system	26,27,31,75,92
26	Complicated Design	36,39,50
27	Large size components	33
28	Unavailable expertise for reusability	49
29	Less reusable components than expected	49
30	Difficulties in verifying design to requirements	40
31	Many feasible solutions	39
32	Incorrect Design	-
33	Difficulties in allocating functions to components	35
34	Extensive specification	38
35	Omitting data processing functions	37
36	Large amount of tramp data	68
37	Incomplete DD	42
38	Large DD	41
39	Unclear DD	41
40	Inconsistent DD	41,45
Phase 3: Implementation and Unit Testing		
41	Non-readable DD	43
42	Programmers cannot work independently	43
43	Developing the wrong user functions and properties	69
44	Developing the wrong user interface	-
45	PL does not support architectural design	-
46	Modules are developed by different programmers	47,48
47	Complex, ambiguous, inconsistent code	55
48	Different versions for the same component	65
49	Developing components from scratch	54
50	Large amount of repetitive code	-
51	Inexperienced Programmers	43,44,52,47
52	Too many syntax errors	-
53	Technology change	-
54	High fault rate in newly designed components	-
55	Code is not understandable by reviewers	59
56	Lack of complete automated testing tools	57,59,60,63
57	Testing is monotonous, boring and repetitive	-
58	Informal and ill-understood testing process	59
59	Not all faults are discovered in unit testing	67
60	Poor documentation of test cases	71
61	Data needed by modules other than the under testing one	62
62	Coding Drivers and Stubs	54
63	Poor Regression Testing	67
Phase 4: Integration and System Testing		
64	Difficulties in ordering components' integration	66,70
65	Integrate the wrong version of components	67,69
66	Omissions or oversights	67,69
67	A lot of bugs emerged during the integration	71

68	Data Loss across an interface	67
69	Integration may not produce the desired functionality	-
70	Difficulties in localizing errors	71
71	Difficulties in repairing errors	87
72	Unqualified testing team	73,78
73	Limiting testing resources	78
74	Inability to test in the operational environment	-
75	Impossible complete testing (Coverage Problem)	87
76	Testers rely on process myths	77
77	Testing cannot cope with requirements change	82,83
78	Wasting time in Building testing	-
79	The system being tested is not testable enough	-
Phase 5: Operation and Maintenance		
80	Problems in installation	-
81	The effect on the environment	85
82	Change in environment	80,83,84
83	New requirements emerge	-
84	Difficulties in using the system	85
85	User resistance to change	88
86	Missing capabilities	87
87	Too many software faults	89
88	Testers does not perform well	87,91
89	Suspension and Resumption problems	-
90	Insufficient data handling	87
91	The software engineer cannot reproduce the problem	-
92	Problems in maintainability	-
93	Budget not enough for maintenance activities	-
Risks Common to all SDLC phases		
94	Continually changing requirements	77,83
95	Project Funding Loss	
96	Team Turnover	
97	Data Loss	
98	Time contention	
99	Miscommunication	
100	Budget Contention	

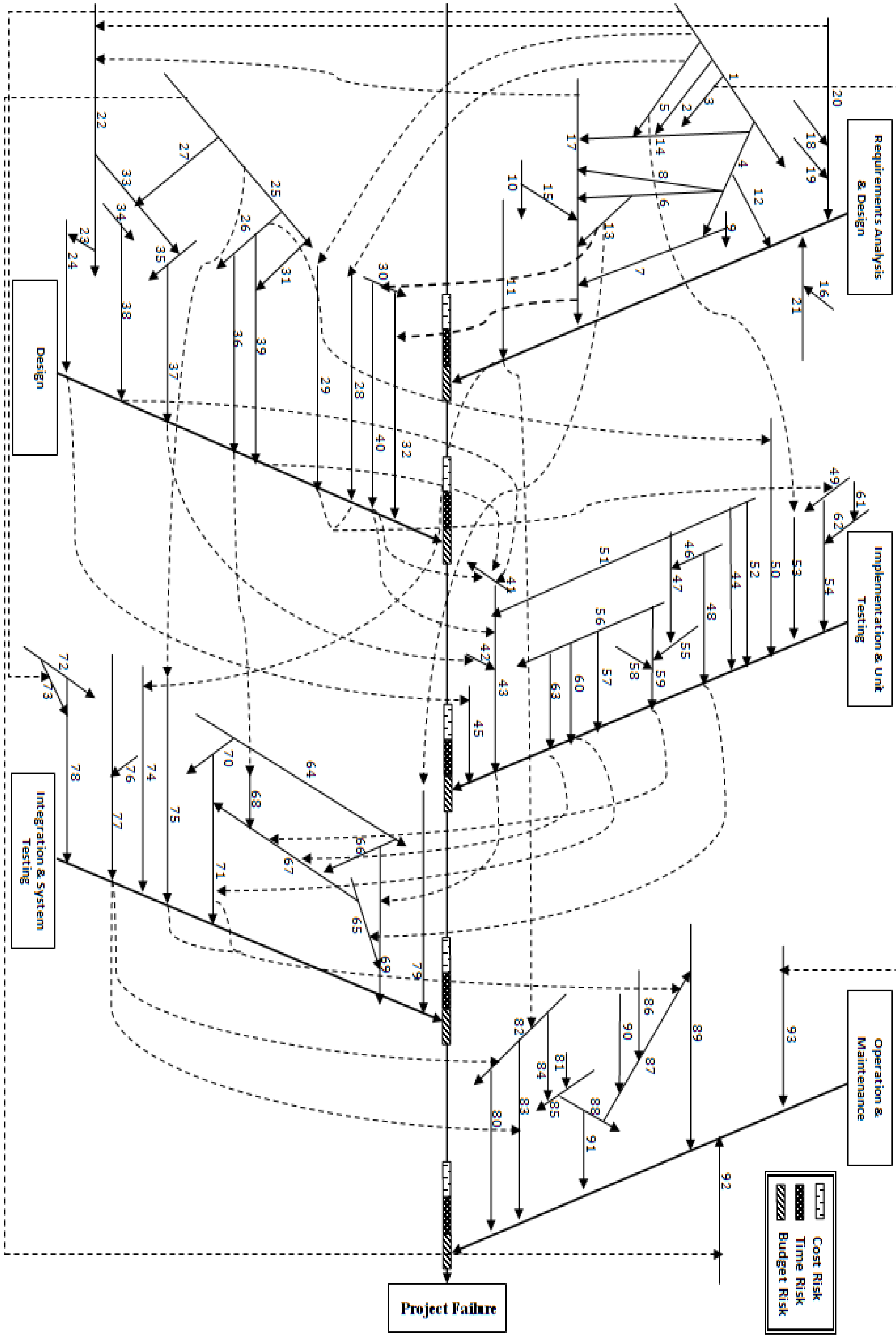

Figure 1. Risk Factors - Cause-effect Diagram

4. Conclusion and Future Work

In this paper, a detailed list of software projects risk factors was examined deeply. It was found that the relationship between software projects risk factor is a cause-effect (or causality) relation. To clarify this, a causality table and a cause-effect diagram were introduced. These can help project managers and researchers in prioritizing the most dangerous risks which can result in the largest number of affected risk factors and then designing strategies to (mitigate/avoid) them. This would reduce the maximum number of risks with the minimum number of strategies. Clearly, this would reduce time, effort, and cost needed to manage project risks.

References

[1] B. W. Boehm, "Software Risk Management Principles and Practices", IEEE Software, vol. 8, no. 1, **(1991)**, pp. 32–41.
[2] P. L. Bannerman, "Risk and Risk Management in Software Projects: A reassessment", Journal of Systems and Software, vol. 81, no. 12, **(2008)**, pp. 2118-2133.
[3] Y. Tao, "A Study of Software Development Project Risk Management", Proceedings of the International Seminar on Future Information Technology and Management Engineering, IEEE, **(2008)**.
[4] J. Nyfjord and M. Kajko-Mattsson, "Commonalities in Risk Management and Agile Process Models", Proceedings of the Second International Conference on Software Engineering Advances, Cap Esterel, French Riviera, France, **(2007)** August 25-31.
[5] Y. Hu, X. Zhang, X. Sun, J. Zhang, J. Du and J. Zhao, "A Unified Intelligent Model for Software Project Risk Analysis and Planning", Proceedings of the 3rd International Conference on Information Management, Innovation Management and Industrial Engineering, Kunming, China, **(2010)** November 26-28.
[6] S. J. Huang and W. M. Han, "Exploring the Relationship between Software Project Duration and Risk Exposure: A Cluster Analysis", Information and Management, vol. 45, no. 3, **(2008)**, pp. 175-182.
[7] M. Keil, P. E. Cule, K. Lyytinen and R. C. Schmidt, "A Framework for Identifying Software Project Risks", Communications of the ACM, vol. 4, no. 11, **(1998)**, pp. 76-83.
[8] T. Addison and S. Vallabh, "Controlling Software Project Risks: An Empirical Study of Methods used by Experienced Project Managers", Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement through technology, Elizabeth, South Africa, **(2002)** September 16-18.
[9] N. H. Arshad, A. Mohamed and Z. M. Nor, "Risk Factors in Software Development Projects", Proceedings of the 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, Corfu Island, Greece, **(2007)** February 16-19.
[10] B. Shahzad and J. Iqbal, "Software Risk Management Prioritization of Frequently Occurring Risk in Software Development Phases", Using Relative Impact Risk Model. Proceedings of the 2nd International Conference on Information and Communication Technology, IBA Karchi, **(2007)** December 16-17.
[11] B. Shahzad and A. S. Al-Mudimigh, "Risk Identification, Mitigation and Avoidance Model for Handling Software Risk", Proceeding of the Second International Conference on Computational Intelligence, Communication Systems and Networks, Liverpool, UK, **(2010)** July 28-10.
[12] B. Shahzad and S. A. Safvi, "Risk Mitigation and Management Scheme based on Risk Priority", Global Journal of Computer Science and Technology, vol. 10, no. 4, **(2010)**, pp. 108-113.
[13] H. Hijazi, S. Alqrainy, H. Muaidi and T. Khdour, "Risk Factors in Software Development Phases", European Scientific Journal, vol. 10, no. 3, **(2014)**, pp. 213-132.
[14] Random House Unabridged Dictionary.
[15] K. Ishikawa, "Guide to Quality Control", Tokyo: JUSE, **(1968)**.

Authors

Haneen Hijazi holds an M.Sc. Degree in Computer Science with a concentration in Software Engineering; she received the degree in 2012 from Albalqa Applied University. She got her B.Sc. degree in Software Engineering from the Hashemite University, Jordan, in 2006. Currently, she works as a computer lab supervisor in the Software Engineering Department at the Hashemite University. In her research, she is interested in software engineering with an emphasis on projects and risk management, e-learning, and Natural Language Processing.

Shihadeh Alqrainy received his B.Sc. and M.Sc. degrees in Computer Science from Yarmouk University, Jordan, in 1985 and 2002, respectively. During 1987-1999, he was a computer instructor and training manager of one of the biggest computer institutes in K.S.A. On completing his M.Sc. Degree in 2002, he started working as a full time lecturer at Albalqa Applied University in Jordan. He got his Ph.D. degree in Artificial Intelligence and Software Engineering from DeMontfort University, Leicester, UK, in 2008. His prime research area of interest within Artificial Intelligence is Natural Language Processing and Understanding. He is working currently as an assistant professor and the chairman of the Software Engineering Department at Albalqa Applied University, Jordan.

Hasan Muaidi is an assistant professor in the Department of Computer Science at Albalqa Applied University, Salt, Jordan. He received his B.Sc. and M.Sc. degrees in Computer Science from Yarmouk University, Jordan, in 1987 and 2002, respectively, and his Ph.D. degree in Artificial Intelligence and Software Engineering from DeMontfort University, Leicester, U.K, in 2008. His current research interests include Artificial Intelligence, Artificial Neural Networks, Arabic Natural Language Processing, Information Retrieval, and Expert Systems.

Thair Khdour received his B.Sc. degree in Information Technology from Albalqa Applied University in 2001 and his M.Sc. degrees in Information Technology (Web Engineering and Design) from Western Sydney University in 2003. He received his PhD degree in 2008 from University of Essex, UK. His Ph.D. research was primarily in the field of Artificial Intelligence. He has been working as an assistant professor since 2008. Currently, he is the chairman of the Computer Information System department at Albalqa Applied University, Jordan.