Research Journal of Applied Sciences, Engineering and Technology 8(8): 919-928, 2014
DOI:10.19026/rjaset.8.1054
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2014 Maxwell Scientific Publication Corp.
Submitted: February 22, 2014 Accepted: March 20, 2014 Published: August 25, 2014

Research Article

A Framework for Integrating Risk Management into the Software Development Process

[1]Haneen Hijazi, [2]Shihadeh Alqrainy, [2]Hasan Muaidi and [2]Thair Khdour
[1]Department of Software Engineering, Hashemite University, Zarqa, Jordan
[2]Albalqa Applied University, Salt, Jordan

Abstract: Software development projects still of high failure rates. Different risk management approaches are recommended by researchers and followed by organizations in order to control this failure rate. Current research is focused towards preventive risk management approach that improves the development process. In this study, we introduce a framework that enhances this approach. This framework describes a systematic method towards enhancing preventive risk management throughout the software development process. In this study, we devised sets of risk management strategies and controls that aim at mitigating each of the identified risks in the adapted list. These strategies besides the identified risk factors are utilized and embedded in the right corresponding Software Development Life Cycle phase to construct our preventive framework.

Keywords: Preventive risk management, risk, risk factor, strategy

INTRODUCTION

Software development process or the Software Development Life Cycle (SDLC) is a structure imposed on the development of a software system. According to this structure the software development process involves five different phases: Requirements Analysis and Definition, Design, Implementation and Unit Testing, Integration and System Testing and the Operation and Maintenance phase (Khdour and Hijazi, 2012).

Risk factors involved in each of these phases threaten project success. This raises question about new improved risk management mechanisms.

Many definitions, approaches and frameworks exist for software project risk management in the literature. Most lead to the application of the set of principles, practices, procedures, methodologies and tools aimed at identifying, analyzing and handling risk factors through the SDLC before they evolve into actual problems that negatively affect the project development process and hinder the successful completion of the project.

Risk management can be either reactive or proactive. In the reactive approaches, risks are not mitigated till their occurrence, while in the proactive we try to avoid the occurrence of risks. Clearly, it is better to avoid risks rather than repairing from their consequences (Singh and Goel, 2007).

A preventive risk management strategy means to proceed into the development process activities and the SDLC phases and risk control strategies with an eye towards the identified risks and preventing them from being materialized.

A risk management strategy is a control activity that aims at dealing with a specific risk factor(s). Not all risk factors are controllable (Zardari, 2009), some factors might be out of project manager's control. Any software risk factor can be either avoidable or non-avoidable. For the avoidable risk factors, mitigation strategies are devised and proposed to deal with risks before they mature into real problems. Else, if the risks are non-avoidable, or if the risks have matured into real problems, then contingency plans have to take place in order to repair from the occurrence of these risks. A mitigation strategy aims at either avoiding the occurrence of a risk, or reducing its effects in case of occurrence. This reduction can be achieved by reducing either the severity of the risk or its likelihood.

Either the mitigation strategies or the contingency plans must be planned in advance (Shahzad and Safvi, 2008). In other words, we must not wait till the occurrence of the risks then start to think of and design strategies. Clearly, applying a mitigation strategy is better than conducting a contingency plan, since it is cheaper and easier than repairing from risk.

A risk management strategy can control more than one risk factor. The occurrence of a risk might be as a consequence of another risk, thus, mitigating the cause may also mitigate the consequence.

In this study, we propose sets of mitigation strategies and contingency plans for a highly technical set of risk factors. A first pass review of the strategies, the reader will notice that most strategies are mitigation strategies (avoidance), rather than contingency plans.

Corresponding Author: Haneen Hijazi, Department of Software Engineering, Hashemite University, Zarqa, Jordan
This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

Indeed these avoiding strategies will form the basis for the preventive risk management framework introduced in this study. These strategies are integrated into each of the five phases of the SDLC phases.

LITERATURE REVIEW

Despite the fact that the current attitude to risk management is a preventive approach, little research has already been conducted regarding the integration between risk management and the software development process. In the following, we highlight several important studies around this issue.

Murthi (2002) proposed a preventive risk management model in software projects. Using his model, software project is divided into sub-projects and each sub-project is assigned to iteration. Singh and Goel (2007) proposed a preventive maintenance model. Their model was based on the three popular software maintenance classifications (i.e., preventive corrective, preventive adaptive and perfective maintenance) and outlined on the basis of the development cycle. Shahzad

Table 1: Software projects risk factors

Index	Risk factor	Index	Risk factor
Phase 1: requirements analysis and definition		51	Inexperienced programmers
1	Inadequate estimation of project time, cost, scope and other resources	52	Too many syntax errors
2	Unrealistic schedule	53	Technology change
3	Unrealistic budget	54	High fault rate in newly designed components
4	Unclear project scope	55	Code is not understandable by reviewers
5	Insufficient resources	56	Lack of complete automated testing tools
6	Unclear requirements	57	Testing is monotonous, boring and repetitive
7	Incomplete requirements	58	Informal and ill-understood testing process
8	Inaccurate requirements	59	Not all faults are discovered in unit testing
9	Ignoring the non-functional requirements	60	Poor documentation of test cases
10	Conflicting user requirements	61	Data needed by modules other than the under testing one
11	Unclear description of the real environment	62	Coding drivers and stubs
12	Gold plating	63	Poor regression testing
13	Non-verifiable requirements	Phase 4: integration and system testing	
14	Infeasible requirements	64	Difficulties in ordering components' integration
15	Inconsistent requirements	65	Integrate the wrong version of components
16	Non-traceable requirements	66	Omissions or oversights
17	Unrealistic requirements	67	A lot of bugs emerged during the integration
18	Misunderstood domain-specific terminology	68	Data loss across an interface
19	Mis-expressing user requirements in natural language	69	Integration may not produce the desired functionality
20	Inconsistent requirements data and RD	70	Difficulties in localizing errors
21	Non-modifiable RD	71	Difficulties in repairing errors
Phase 2: design		72	Unqualified testing team
22	RD is not clear for developers	73	Limiting testing resources
23	Improper AD method choice	74	Inability to test in the operational environment
24	Improper choice of the PL	75	Impossible complete testing (coverage problem)
25	Too much complex system	76	Testers rely on process myths
26	Complicated design	77	Testing cannot cope with requirements change
27	Large size components	78	Wasting time in building testing
28	Unavailable expertise for reusability	79	The system being tested is not testable enough
29	Less reusable components than expected	Phase 5: operation and maintenance	
30	Difficulties in verifying design to requirements	80	Problems in installation
31	Many feasible solutions	81	The effect on the environment
32	Incorrect design	82	Change in environment
33	Difficulties in allocating functions to components	83	New requirements emerge
34	Extensive specification	84	Difficulties in using the system
35	Omitting data processing functions	85	User resistance to change
36	Large amount of tramp data	86	Missing capabilities
37	Incomplete DD	87	Too many software faults
38	Large DD	88	Testers does not perform well
39	Unclear DD	89	Suspension and resumption problems
40	Inconsistent DD	90	Insufficient data handling
Phase 3: implementation and unit testing		91	The software engineer cannot reproduce the problem
41	Non-readable DD	92	Problems in maintainability
42	Programmers cannot work independently	93	Budget not enough for maintenance activities
43	Developing the wrong user functions and properties	Risks common to all SDLC phases	
44	Developing the wrong user interface	94	Continually changing requirements
45	PL does not support architectural design	95	Project funding loss
46	Modules are developed by different programmers	96	Team turnover
47	Complex, ambiguous, inconsistent code	97	Data loss
48	Different versions for the same component	98	Time contention
49	Developing components from scratch	99	Miscommunication
50	Large amount of repetitive code	100	Budget contention

and Safvi (2008) proposed sets of mitigation strategies and contingency plans for each software risk factor identified previously by Shahzad and Iqbal (2007).

Nyfjord and Kajko-Mattsson (2008) investigated the state of practice of integrating risk management with the development process in different software organizations. They found that this type of integration still in its infancy. Shahzad *et al*. (2010) utilized an already defined set of risk factors that might be faced during software development and proposed different strategies to mitigate each of the identified risks. Shahzad and Al-Mudimigh (2010) proposed a model that aims at handling risks in software development environment, it is called RIMAM. Recently, Khdour and Hijazi (2012) proposed a preventive risk management model that integrates risk management with the software development process.

PROPOSED METHODOLOGY

The framework introduced in this study is built on top of the model proposed by Khdour and Hijazi (2012). In their model, they suggest that most potential risk factors must be identified early in the development process. Regarding the risk factors, herein, we adapt a detailed set of risk factors (Hijazi *et al*., 2014). This list of risk factors is summarized in Table 1. In their work, they proposed a preventive risk management model that integrates risk management into the SDLC. This framework guarantees this integration. To achieve it, this framework suggests developers to take the potential risk factors into their consideration while developing the software system and to devise and corporate mitigation strategies throughout the entire life cycle of the software development process. This framework embeds risk management strategies into the SDLC and assigns the right strategies to the right development phase in which it should be conducted in order to avoid the occurrence of the target risks. This framework considers the most frequent avoidable risks which lie under the control of the project manager and the development team. In this framework, risk management strategies take place during the development process while developing the system. It shows the sequence of these phases, activities and strategies that manage software risks effectively.

In addition, a set of mitigation strategies was defined for each risk factor to help developers and projects managers deal with these risks. By introducing these sets of mitigation strategies we significantly raise baselines for our preventive risk management framework in which these strategies constitute the main building blocks.

Herein, we use process flowcharts to visualize the proposed framework. These Fig. 1 to 5 show the sequential execution of the software development process activities supported by different strategies that aim at handling specific risk factors. Hence, the software development process operation is represented by a set of boxes of different types connected by arrows that represent the flow of control.

The main building blocks: The main building blocks of these Fig. 1 to 5 involve the following components.

The activity: Indicates a development process step represented by a rectangle.

The strategy: Indicates a risk handling mechanism represented by a rounded rectangle.

The factor: Indicates the item that may cause risk, represented by an index contained in a square bracket and attached to the end of the risk strategy statement in order to relate the source of risk to the mitigation strategy that is proposed to handle it. For instance, to reference the factor numbered 12 in our list we attach "[F12]".

Decision to be made or condition to be tested: Represented by diamond with two coming out arrows labeled by (Yes/No or True/False).

Iterative process (loop): Wherein an arrow points to another arrow, represented by black blob.

Start/end of the process: Represented using the start/end symbol.

A document: A data file the might act as an input to or output from a specific phase or activity represented by a wavy-base rectangle.

Input/output: Each phase has its inputs and outputs; the output of a phase is usually the input of the next phase, represented using parallelograms.

The flow of control: Represented by arrows that shows either the moves between two activities, between the activity and its (input/output), or the embedded strategies into the activity.

As foreshadowed above, we can notice that the SDLC comprises five phases, each phase involves a set of activities and each activity underlies different strategies that aim at mitigating risks. Moreover, some activities can be themselves considered as mitigation strategies to specific factors if they were performed properly. Each strategy may address one or more risk factor and each of the identified risk factors can be mitigated by one or more strategy. These strategies may involve tools, techniques and behavioral aspects.

Requirements analysis and definition phase is a system engineering activity that aims at extracting what the system should do and how from the users' needs. It was found that most of the high level risks lie in the early phases of the software development process (Abdullah *et al*., 2010). Moreover, it was found that risks in this phase greatly affect the cost and schedule of the project (Abdullah *et al*., 2010).

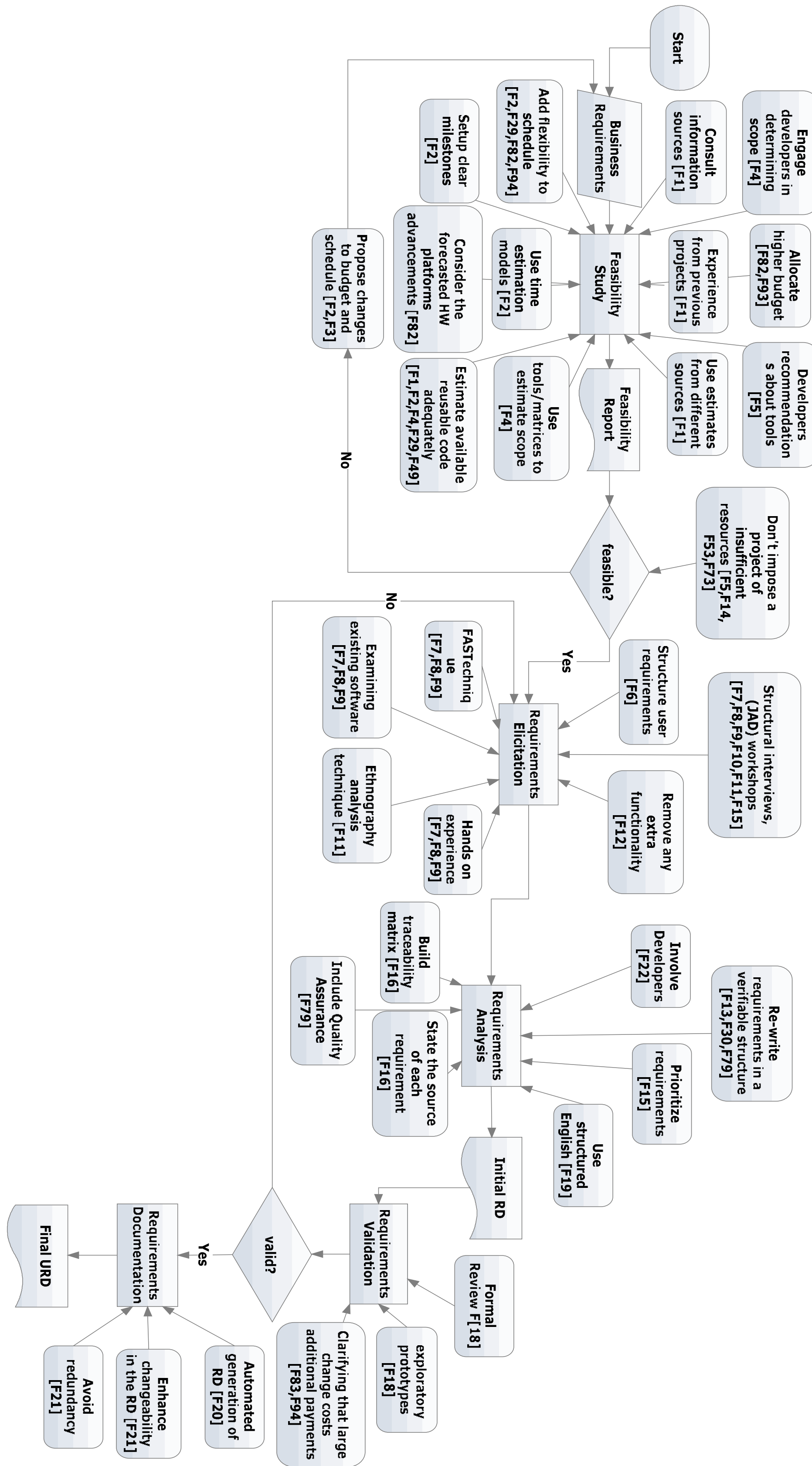

Fig. 1: Requirements analysis and definition phase

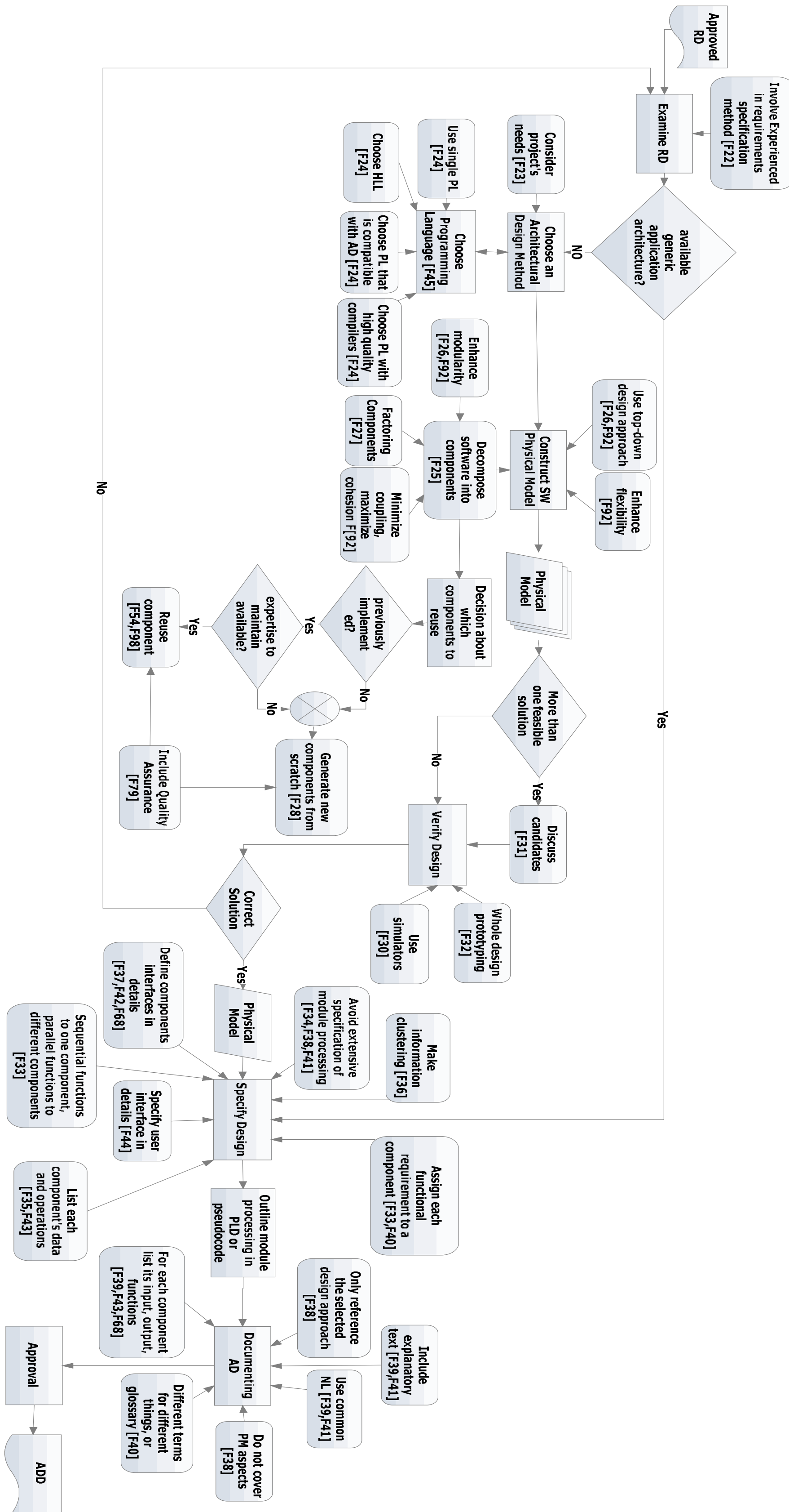

Fig. 2: Design phase

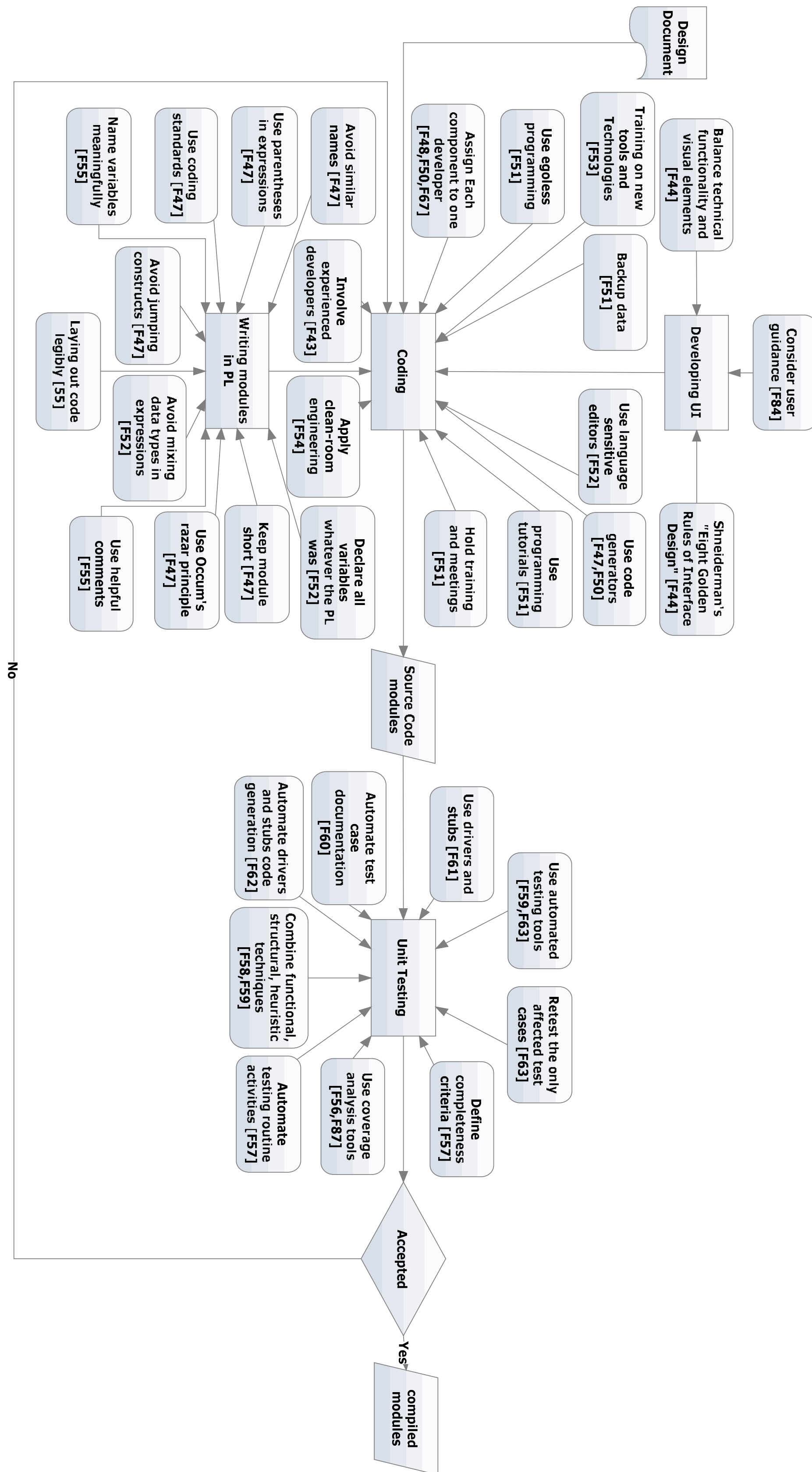

Fig. 3: Implementation and unit testing phase

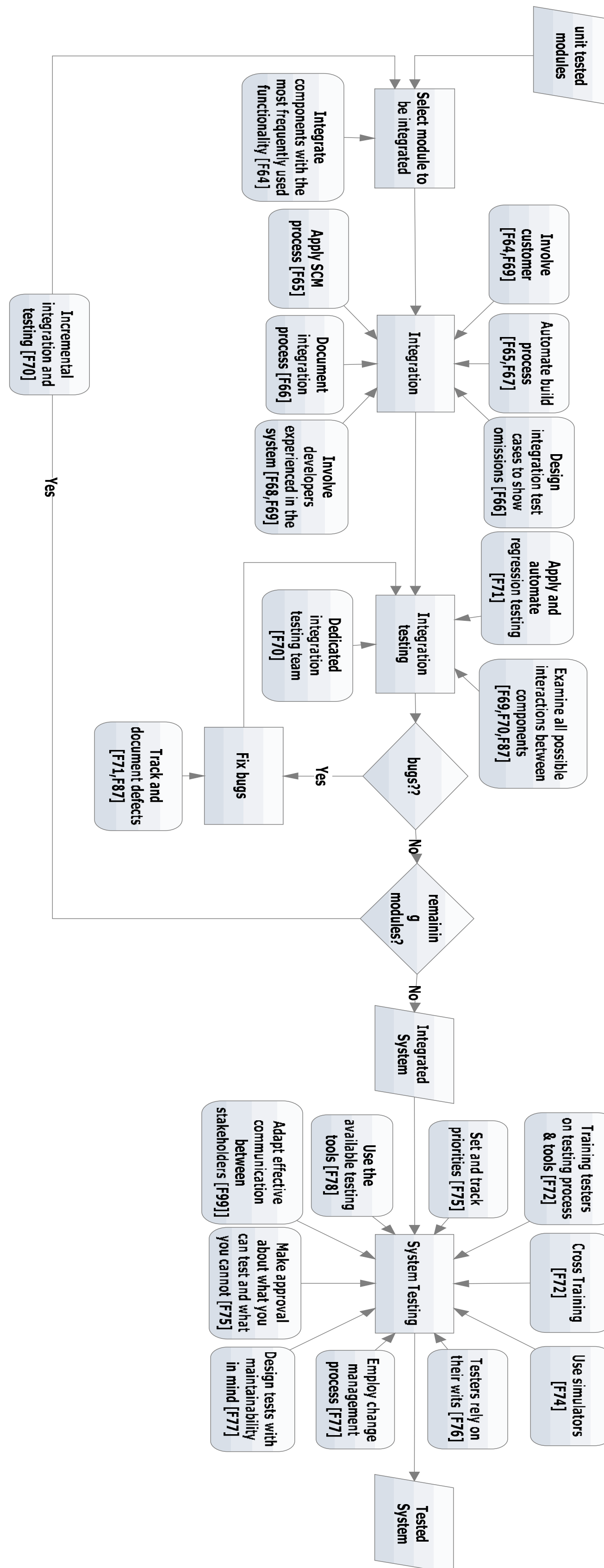

Fig. 4: Integration and system testing phase

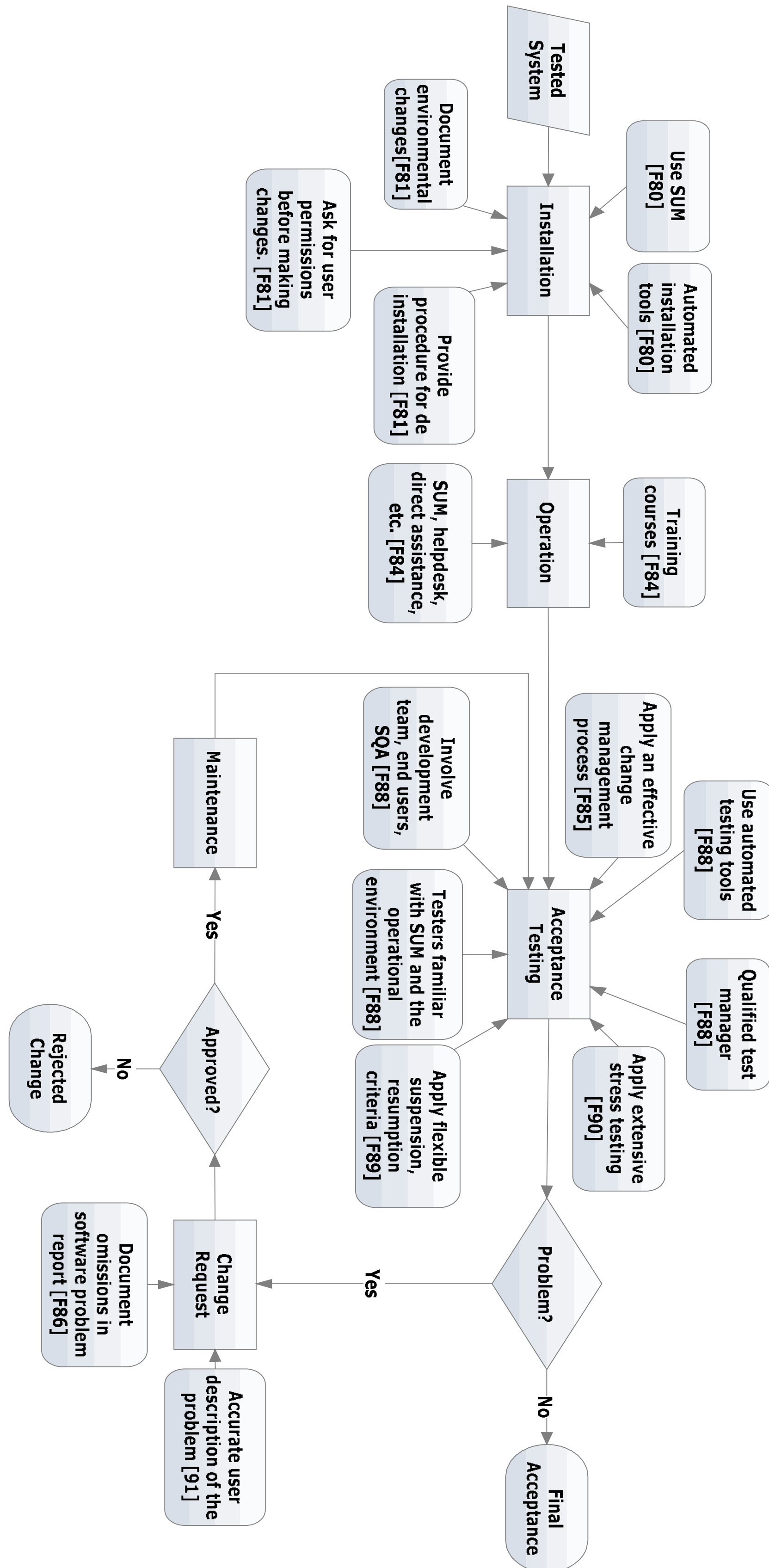

Fig. 5: Operation and maintenance phase

The detailed integration of risk management into the requirements analysis and definition phases is illustrated in Fig. 1 of the model. The main input to this phase is the Business Requirements in which the users' needs from the new system are stated from the business point of view. The main output is the Requirement Document (RD) which specifies and documents what the users expect from the software system under construction. This document is the main input to the next phase (design phase) and to the testing phase (mainly the acceptance testing). As it is obvious from the figure; the requirements analysis and definition phase consists of five main activities involving feasibility study, requirements elicitation, requirements analysis, requirements validation and requirements documentation. During each activity; different risk mitigation strategies could be developed in order to avoid specific risk factors.

Design phase is critical. The majority of risk factors in this phase are consequences from the increased design complexity, which makes it very expensive to resolve any fault in the design later on. The detailed integration of risk management into the design phase is illustrated in Fig. 2 of the model. The main input is to this phase is the "Approved Requirements Document", the main output is the "Detailed Design Document". Design phase consists of four major activities: construct the physical model, verify design, specify design and document design, beside several strategies involved in order to handle specific risk factors.

In the coding and unit testing phase, wherein the actual implementation of the system is carried on, the majority of risks are related to the programmers themselves (Khdour and Hijazi, 2012); their different customs, their natural tendencies and their relationships, etc.

The detailed integration of risk management into the coding and unit testing phase is illustrated in Fig. 3 of the model. The main input to this phase is the "Detailed Design Document", the main output is the "Compiled Modules". This figure shows that coding and unit testing phase involves three main activities: developing UI, coding modules and unit testing.

In the integration and system testing phase, potential risks will set off to show their powers here, thus, it is the most risky phase (Khdour and Hijazi, 2012). To reduce this, risks should be avoided early in the development process. Most risk factors in this phase are related to testers, the followed testing techniques and the boring nature of the testing process itself (Khdour and Hijazi, 2012).

A detailed integration of risk management into the integration and testing phase is illustrated in Fig. 4 of the model. The main input is to this phase is the "Unit Tested Modules", the main output is the "Tested System". This phase consists mainly of three activities: integration, integration testing and system testing. Integration and integration testing are done incrementally for each program module; that is why the loop exists in the same figure.

When developing the system is finished and is put into actual use, few risks could be avoided. The majority of the faults appear in the operation and maintenance phase are consequences from the non-mitigated risks appeared in earlier phases. The detailed integration of risk management into the operation and maintenance phases is illustrated in Fig. 5. The main input to this phase is the "Tested System" and then the system goes to a never ending maintenance phase till the final acceptance of the system.

We have proposed a set of mitigation strategies that aim at mitigating each of the risk factors identified in the previous chapter. These strategies help developers and projects managers deal with these risks. By introducing these sets of mitigation strategies we significantly raise baselines for our preventive risk management model in which these strategies constitute the main building blocks. Then, a preventive risk management model is proposed; this model integrates risk management into the software development process. It utilizes the set of risk factors listed in the previous sections that threaten the software development process and embeds their mitigation strategies through the different phases of the SDLC. The model proposed by this section represents the third step of our proposed strategy.

CONCLUSION AND RECOMMENDATIONS

In this study, we introduced a framework to integrate risk management into the software development process. Risk management strategies are proposed and integrated into the different phases of the SDLC as process activities in order to avoid/mitigate the one hundred risk factors identified early in this since most of its strategies are avoidance ones that aim at handling risks before they materialize.

REFERENCES

Abdullah, T., A. Mateen, A. Sattar and T. Mustafa, 2010. Risk analysis of various phases of software development models. Eur. J. Sci. Res., 40(3): 369-376.

Hijazi, H., S. Alqrainy, H. Muaidi and T. Khdour, 2014. Risk factors in software development phases. Eur. Sci. J., 10(3): 213-132.

Khdour, T. and H. Hijazi, 2012. A step towards preventive risk management in software projects. Proceeding of the 2012 4th International Conference on Software Technology and Engineering. Phuket, Thailand, September 1-2, pp: 471-478.

Murthi, S., 2002. Preventive risk management software for software projects. IT Prof., 4(5): 9-15.

Nyfjord, J. and M. Kajko-Mattsson, 2008. Outlining a model integrating risk management and agile software development. Proceeding of the 34th Euromicro Conference on Software Engineering and Advanced Applications. Parma, Italy, September 1-5, pp: 476-483.

Shahzad, B. and S. Iqbal, 2007. Software risk management-prioritization of frequently occurring risks in software development phases. Using relative impact risk model. Proceeding of the 2nd International Conference on Information and Communication Technology. Dhaka, Bangladesh, March 8-9, pp: 110-115.

Shahzad, B. and S. Safvi, 2008. Effective risk mitigation: A user prospective. Int. J. Math. Comput. Simulat., 2(1): 70-80.

Shahzad, B. and A. Al-Mudimigh, 2010. Risk identification, mitigation and avoidance model for handling software risk. Proceeding of the 2nd International Conference on Computational Intelligence, Communication Systems and Networks. Liverpool, United Kingdom, July 28-30, pp: 191-196.

Shahzad, B., A. Al-Mudimigh and Z. Ullah, 2010. Risk identification and preemptive scheduling in software development life cycle. Glob. J. Comput. Sci. Technol., 10(2): 55-63.

Singh, Y. and B. Goel, 2007. A step towards software preventive maintenance. ACM SIGSOFT, 32(4), Article No. 10.

Zardari, S., 2009. Software risk management. Proceeding of the 3rd International Symposium on Empirical Software Engineering and Measurement. FL, USA, October 15-16, pp: 375-379.