International Journal of the Physical Sciences Vol. 6(8), pp. 2072-2082, 18 April, 2011
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS11.480
ISSN 1992 - 1950 ©2011 Academic Journals

Full Length Research Paper

Trivial model for mitigation of risks in software development life cycle

Basit Shahzad[1]*, Yousef Al-Ohali[2] and Azween Abdullah[3]

[1]Deanship of E-Transactions and Communication, King Saud University, Riyadh, Saudi Arabia.
[2]College of Computer and Information Sciences, Saud University, Riyadh, Saudi Arabia.
[3]University Technology PETRONAS, Perak, Malaysia.

Accepted 12 April, 2011

Software development is the art of developing the software in an appropriate manner by using the software development life cycle, regardless of the fact that which model is used for the development. The development is a dynamic activity and requires a lot of rational thinking during the analysis, design, coding, testing and maintenance phases of software development. As the development of software is becoming more systematic and tool-driven, and due to the over emphasized use of the technology the orientation of risks are increasing but the attention to risk management has been observed to be helpless to improve with the same pace to tackle the dynamically increasing software risks. In the last decade, precisely, having felt the importance of software risk management, increasing emphasis has been given in this domain. Therefore the academic and industrial community is worried to consider that how the risks can be handled to minimize the losses and to increase the profits and maintain reputation in the market This paper focuses on the aspect of suggesting the techniques to handle or manage the software risks. Taking into consideration the eighteen most prominent risk factors that affect the software quality and software process, the handling or avoidance strategy has been proposed. This paper suggests addressing the risk factors to be treated not only by the technology but by using intuition as well.

Key words: Software risk management, software risk handling, risk mitigation.

INTRODUCTION

Software risk management has been a very hot area of research since last three decades. Recently, the research community looks seriously interested to identify not only the risk factors but also the causes of the appearance of the risk factors in software development life cycle and how these risks can either be handled or avoided. A recent survey of 600 firms indicated that almost 35% of them had at least one 'runaway' software project (Rothfeder, 1988). In another study, conducted on almost 13,000 projects, it was investigated that almost 25% of the projects were either delayed or faced a failure. It has been observed that most problems in the software industry are faced just because of the poor software risk handling mechanisms or due to the absence of any such mechanism at all. In this regard it is important to note that currently strong emphasis is being given on this domain to identify more and more risk factors (Jones, 1996).

It is strongly believed that the risk identification, particularly, is an ongoing process, and apparently there is no full stop as the risk factors keep on increasing with the arrival of new technologies, people, environment, management and the circumstances. So a claim about the identification of all risk factors available in the entire software process may not be realistic. However, the researchers have keenly considered the identification of risk factors and the prioritization of risk factors is also an open area of research. This paper not only focuses on the identification of the risks but also provides a mechanism to handle them effectively.

The paper discusses the RIMAM model of software risk management by providing the step-wise execution of the risk handling methodology. The model presents the easy-to-understand, flowcharts to express the working of each mitigation/avoidance strategy against any risk factors,

*Corresponding author. E-mail:basit.shahzad@gmail.com. Tel: 00966 5698 38103.

Table 1. The risk factors with respect to their identifier.

Risk #	Risk factor	Risk #	Risk factor
1	Immature requirements	8	Staff turnover
2	Divergent estimation of cost and time	9	Staff inexperience
3	Massive user Stress	10	Backup issues
4	Less reusability	11	Natural disasters
5	Project delivery milestones	12	Excessive error detection
6	Funding un-certainty	13	Developer's faithfulness
7	Over-optimistic technology Perceives	14	Preservation of intellectuals

thus providing the development team a chance to address the risk locally. The organization may or may not opt to follow the RIMAM model in full any may opt to have practice the subset of it, which also is possible, depending on the needs of the risk management activity. The dependence diagram in this paper identifies the dependencies of the risk factors on each other. Having known that a risk on which quite a few factors are dependent, it is imperative to keep that in order to maintain the handling of risks in a software development life cycle.

PREVIOUS WORK

Software risk identification is an open area of research and considerable research has been done in recent past. (Pressman, 2000) has made an effort to identify the software risks, and has provided the ten broader risk factors. Boehm, in his work has also provided a list of top ten risk categories (Boehm, 1998). In a recent paper on risk management, the risk factors have been prioritized according to their frequency of occurrence and the impact that they possess (Shahzad, 2005), and thus a list of eighteen risk factors with respect to their total impact has been prepared. The list is presented in Table 1 which presents the list of all 14 risk factors, which presents the ordered list of software risk factors with respect to the overall impact of each risk factor (Shahzad, 2007).

The risk factor identified in this list is expected to cover a border range of the risks that may come into the software development process. Still the author feel himself restricted, not to claim that this list covers all possible risk factors widely focused area of research.

Software risk identification and mitigation has been a prime area of research since last two decades, and this area of research has received a highly overwhelming response and contribution from the researcher both: in industry and academia, world-wide. In order to identify the recent trend and practices in the domain of software risk identification a comprehensive literature survey was conducted that has helped in the more effective management of risk factors.

Danny (2006) has worked to reduce operational risks by improving the software quality. Danny focuses on the classification and quantitative evaluation of removing the software risks by effective software management, thus contributing to the classified risk mitigation. In a study that was conducted in 2005, a sample of 167 customer's data breaches were analyzed to view the distribution of risks and threats and it was identified that 3% of the total risks are caused by accidental disclosure bye-mails, 7.8% of risks are oriented due to the human weaknesses, 40.1% risks are caused by unprotected computer/backup media and 49.1% of risks are caused due to the malicious exploitation of software risks. Thus, the suggested way mitigates the risk factors more appropriately.

The SEI reports that 90% of all software risks are due to already known defects , while all of the SANA top 20 internet security problems are result of poor coding, testing and sloppy software engineering. In recent past huge emphasis has been on the management of risks in distributed software projects, which proposes a framework for handling the software projects that are not developed at geographically same location, and have advised a framework to e followed in this regard (Persson et al., 2009). Alge et al. (2003) emphases on the effective handling of risks and problems in the software development lifecycle and in team structure by the usage of knowledge building process and effective communication (Alge and Witheoff, 2003). Bradner et al. (2005) have worked to identify the correct team sizes for the different project sizes and have focused the problems that are experienced by over, low and poor staffing (Erin et al., 2005).

Burn (2001) and his team have discussed the risks that are oriented due to the in-appropriate application selection methodology, especially in the database projects (Burns and Dennis, 1985). Charatte (1989) has proposed the analysis and management of the risk factors in software development process (Robert, 1989). The surveyed literature has been identified greatly in the favor of categorical identification of the risk factors as the existence of risk factors can be extremely harmful, if not attended at the proper time by giving due consideration.

Boehm (1998) has identified the software risks in four different domains of software, including requirement, personnel, re-usable software and tools and environment. These four categories have been sub-classified by identifying more categories falling in the same categories. Barry has identified the risk factors and has given three different choices for the selection of probability. The user has to decide himself what probability of some specific threat the software is facing and has to choose accordingly. The improbable risk factors have the probability range 0.0 to 0.3, the probable from 0.4 to 0.6 and frequent from 0.7 to 1.0.

Westfall (2009) has mentioned the idea of providing a balance between the risks and opportunities and has emphasized that taking risk provides profit and huge market standing and to avoid the risks additional financial burden is to be accepted. The author has provided two conditions to limit the risk, firstly, the risk occurs and become obvious as a problem, and secondly is that the project succeeds. In order to manage the risks a model that focuses on the identification of the risk has been proposed. The author has proposed that interview, volunteer reporting, product decomposition, project decomposition, Assumption analysis, and risk taxonomies for the purpose of risk identification. The risk statement phase is described to consist of the source, risk condition, consequence and any partial consequence. In risk analysis phase the technical, cost, schedule and customer satisfaction contribute to calculate the final exposure of the risk, based on the assumptions. For the purpose of risk management, the author focuses on a detailed tracing of the risk itself. First it asks for having additional information to see if it can be avoided if not then if the risk can be transferred, if it cannot be transferred it is accepted by providing the mitigation.

Hoodat and Rashidi (2009) have focused on the classification and proposing the strategy to calculate the overall impact of these risk factors in a software development life cycle. In order to manage the risks effectively, the author has proposed to index the risk factors by calculating the impact and likelihood of each risk factor. In the risk assessment phase, risks have been classified in a numerous ways. The author has first categorized them as risks internal to the software and risks external to the software. The software risks have been also grouped into process, project and product risks. The author has further categorized the risks into performance risks, cost risks and scheduling risks. The author has further classified different risks in five different classes including requirement risks, cost risks, scheduling risks, quality risks and business risk. In these broader categories many risk factors have been mentioned with quite a few being redundant and over observant. The author has introduced the use of logic gates to present the flow and dependence of risk factors, and has calculated the dependence of risk factor on one-another in the categories already mentioned.

Armestrong and Adens (2008) have emphasized that risk is part of economic enterprise and profit. Risk management is done to minimize the negative effects of any risk rather than investing on projects to handle risks that are hardly expected to be active. The authors have elaborated that the first step to manage the risks effectively is to be aware of the negative effects of the risk and how to safeguard them. The authors have also emphasized the need for collection of risks through the survey by asking management, developers, development teams, customers and any other project stake holders. The authors have also identified 8 area of exposure to which they expect the risk factors to generally belong to. These areas include: clarity, reliability, availability, experience, stability, suitability, dependency. The risks can be categorized in 6 major areas including: requirement changes, unreliable and un-realistic estimates, high staff turnover, lack of project standards and process, lack of design and inadequate documentation and inadequate testing and quality procedures. The authors mentioned that the risks should be prioritized but keeping the business context in mind. The authors urge to take into account the business priorities like increased customers service and satisfaction, improved delivery time, reduced dependency on constraints, improved staff skill level, reduced costs, improved cost estimation and planning and benefit from re-usable components while prioritizing the risks. In terms of reducing the risk that author focuses on the establishment of incremental development process. From the above discussion, it can be identified that the impact and probability of risk factors is generally calculated by doing the interviews, surveys and learning from personal experiences.

The literature is fertile to have quite a few techniques for identification and consequently for the mitigation of risk factors.

(a) Risk taxonomy: The risk taxonomy follows the SDLC and provides a framework for the handling of information. This method is a kind of instrument with which the system level risks can be obtained (Higuera and Haimes, 1996).

(b) Risk clinic: Risk clinic is a type of workshop that takes the SEI's continuous risk management (CRM) and team risk management (TRM) and links it with the communication, project management and risk management channel of the client (Higuera and Haimes, 1996).

(c) Continuous risk management (CRM): CRM is a principal based way of handling with the risks and opportunities during the SDLC, it provides a control on the management of risks regardless of the tool and technique used. As the handling continuous, the frequency of handling the risks is very high yet the approach is expensive as huge emphasis remains on the risk identification and mitigation of the identified risks

(Higuera and Haimes, 1996).

(d) Team risk management (TRM): TRM extends risk management with team oriented activities, involving both customer and developer (Higuera and Haimes,1996).

(e) Survey, questionnaires and interviews: Provide the way of direct communication with the customer and can produce the results in very short time and can be highly effective.

(f) Intuition: The experience of the experienced team leaders can be used as an asset to guide the development team about the identification and mitigation techniques for the identified risks in the software projects.

In the presence of the identification techniques above, the literature is infertile to produce an example in which risks could be identified by the usage of any single technique. It has been observed that mostly more than one techniques are used at the same time to identify the maximum amount of risk factors, but which techniques to combine, purely depends on the nature, scope, budget, need and staffing of the project. But intuition and survey (questionnaire, interviews) are more likely to be among all combinations that can be proposed for effective risk identification.

MODEL FOR SEQUENCE OF ACTIVITIES

Figure 1, represents the sequence of activities that are performed to avoid, mitigate and handle risk factors that have been discussed in “Handling and avoidance mechanism”. The risk identification, management and avoidance model (RIMAM) is presented in Figure 1. RIMAM briefly presents the moves that are expected for the purpose of identification, management and avoidance of each risk factor presented in Table 1. The model proposes that for the handling of risk factor ‘Immature Requirements’ the inputs like interview results, questionnaire results and results of direct client communication should be present in order to form the initial requirements. Then the FAST and JAD sessions can be applied on the initial requirements to help in identifying the final requirements and then the software can be developed based on those requirements.

In an effort to identify why requirements are not properly understood and consequently why incorrect estimates about time and cost of software ware made, it was identified in a survey by Standish group, in 2004 that only 29% of the projects succeed and, 53% are challenged and 18% fail completely. Oxford University, in a survey tried to measure the project failures, and concluded that only 16% project succeeds, 74% were challenged and 10% met complete failure. It was further investigated that the ‘divergent estimation of cost and time’ is the major issue that lead the project failure. Similar findings have been proposed by the British Computer Society and national Institute of Standards and Technology. It has been further concluded that the ‘divergent estimation of cost and time t’ are mainly caused by following prominent factors. The factors are: ‘Incorrect requirements’, ‘lack of training on tools and inexperience’ and lack of intuition. Session (2009) has also investigated the same in his white paper. The dependency diagram is shown in Figure 2 (a).

The risk factor ‘less reusability’ is operational after the requirements have been finalized. The process of identifying the reusable code is initiated. If the reusable code is identified as per the initial expectations things proceed fine but in case the reusable code is not found as expected, the development team has to develop and test the code that causes the increment in the development time and cost as discussed in the study. The reusable code can be used in three ways: Software libraries, design patterns or framework (Session, 2009). It has been identified that the availability of less reusable code as compared to what had been scheduled in the beginning is a major problem which forces the development team to develop, test and integrate the piece of code. The ‘less reusability’ problem is expected to have been influenced by the three factors: ‘incorrect requirements’, ‘lack of training and in-experience’ and ‘lack of intuition’. The dependency diagram is shown in Figure 2 (b).

Next, the model discusses the risk of tightening delivery deadline and funding un-certainty. The model proposes to identify the expected time and cost with respect to the actual development, if found adequate enough and the requirements are expected to change only after a fixed amount of time the team may opt for the incremental model to follow for the software development and in the other case if the requirement change is frequent the team may first opt to finalize requirements, apply FAST and then develop. The cost of finalizing requirements and then developing will be discussed. The dependency diagram is shown in Figure 2 (c).

The risk item ‘over-optimistic technology perceives’ discusses the risk of the technology that does not meet the requirements, the avoidance strategy proposed in this regard takes into the account the initial requirements, and after the discussion with the customer the final selection about the technology is made. The continuous change remains in practice with consultation with the customer. The RIMAM focuses on this continuous consultation with the customers in order to ensure that things are done in order and the risk factor is avoided effectively. This risk factor is expected if the development and estimation of the project is done in a casual manner: without measuring the impacts of risks and individual capabilities of the workers. This risk factor is dependent on two other risk factors, namely ‘staff in-experience’ and ‘lack on

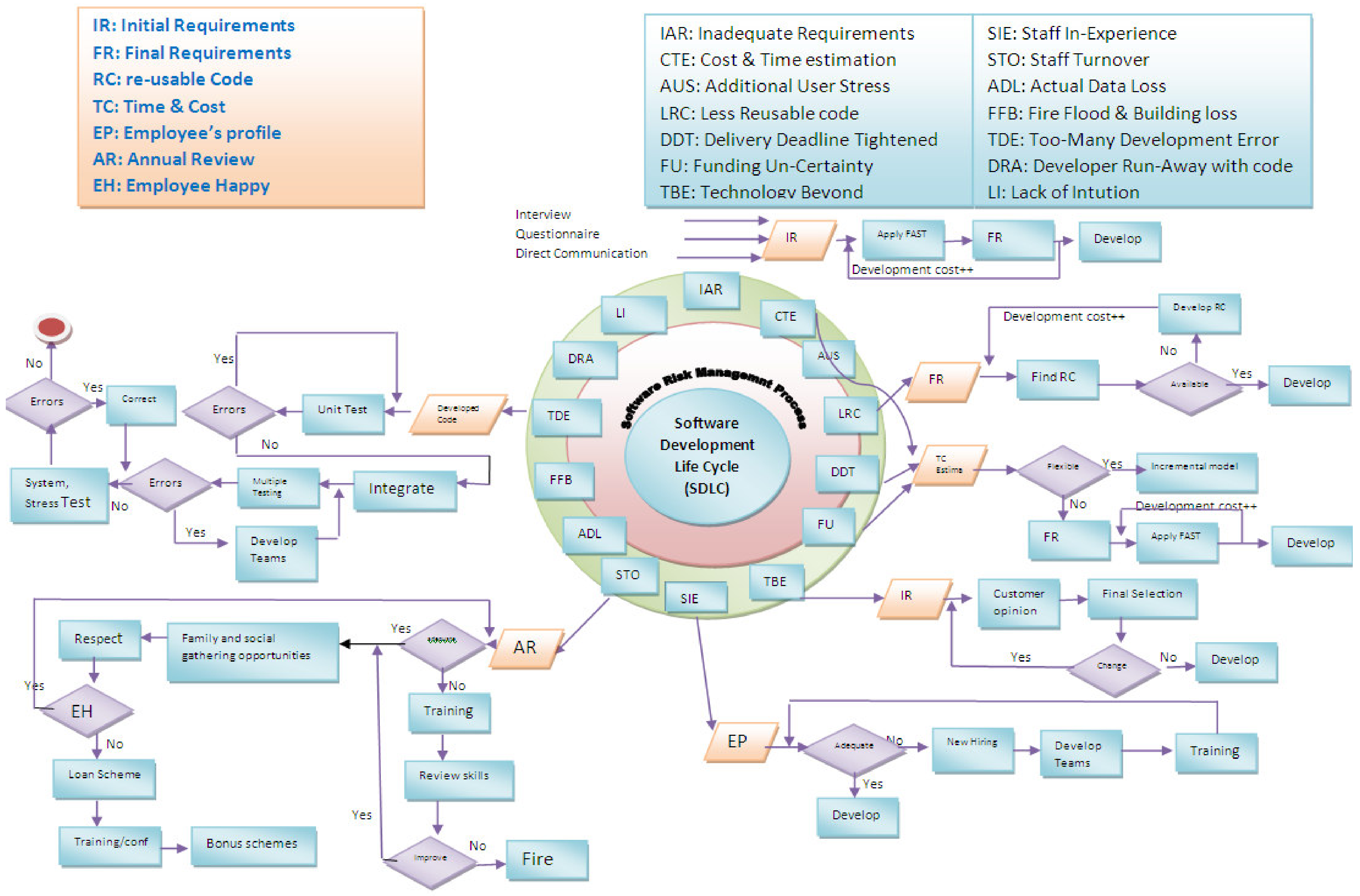

Figure 1. Risk identification, management and avoidance model (RIMAM).

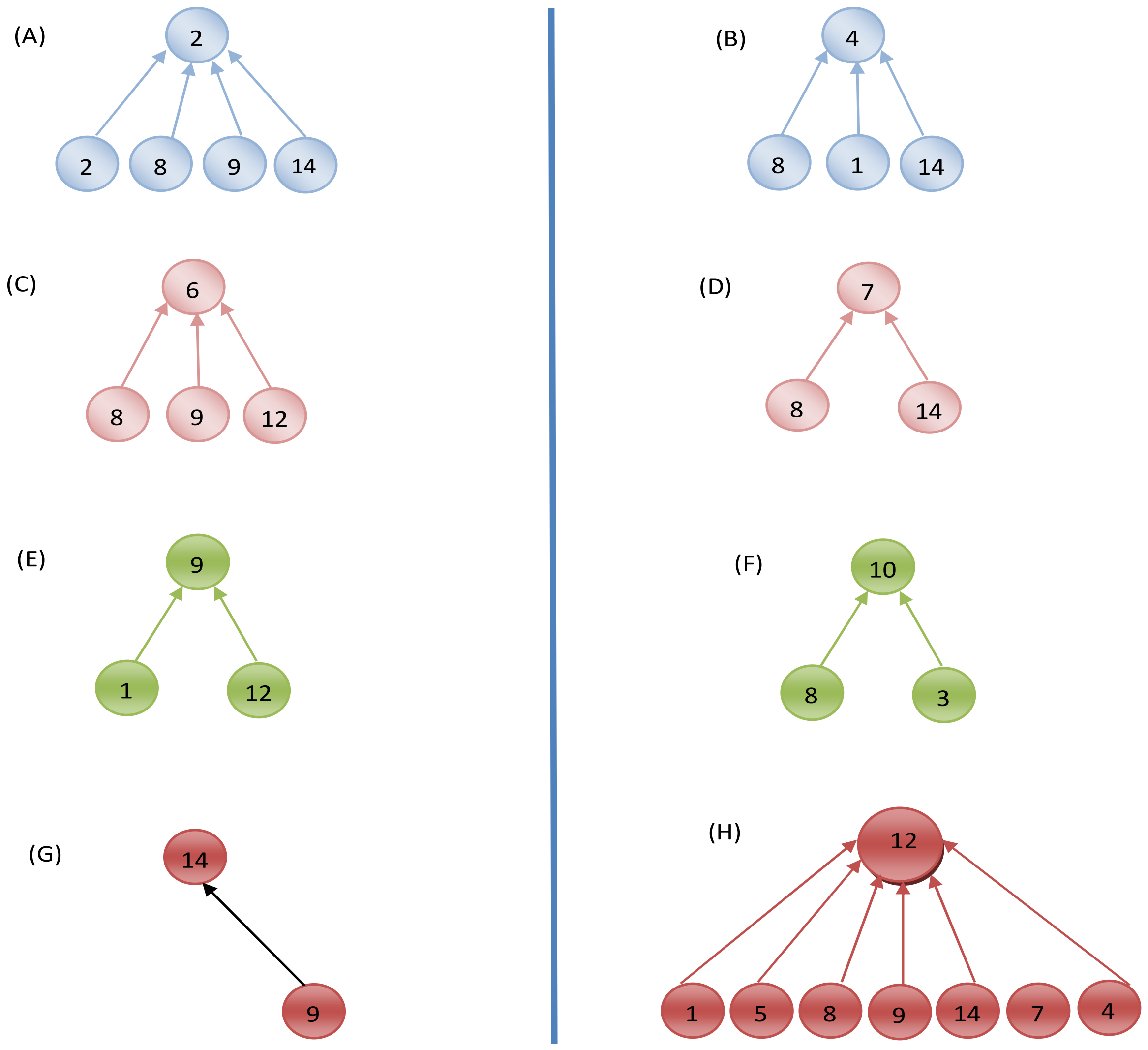

Figure 2. Inter Dependence of software risk factors.

intuition'. The dependency diagram is shown in Figure 2 (d).

The risk factor 'staff inexperience' discusses the avoidance strategy of the staff inexperience risk by taking into account the employee profile and having to see that if its adequate or not, if the employee profile is found adequate he may be deployed to develop the software other-wise if the profile is not adequate the team structure may be developed and training may also be provided to enhance the capability of the individual employees and if the employee profile is not found adequate after several such efforts, the employee may be fired and new hiring may be done in order to train and work in the software industry. The dependency diagram is shown in Figure 2 (e).

The 'staff turnover' of the RIMAM proposes the avoidance strategy regarding the employee leaving the organization frequently. Since the employees are an asset to the organization, it is important that good employees are retained within the organization by providing the attractive perks and privileges. The RIMAM proposes the annual review of the employees profile and if found suitable he may be provided with the access and participation in the meetings, conferences and social gathering events by the firm and should be given respect accordingly. The firm must adopt a secret or visible framework to see that if the employee is happy. If he is not happy the firm may offer the employees with loan schemes, trainings and bonus etc. in order to keep them committed and motivated. It is also essential that the pays are rightly at par with that are provided in the market by other competitors. It is important to note that the staff turnover is not directly dependent to most of the risk factors being discussed in this paper, yet it can be indirectly influenced by the managerial decision, which, as a plenty can stress the team member and the team member leave the organization in response (Hall et al., 2008). The management behaves this way if they identify that the worker is adding problem in development, and cannot perform his duties adequately. Other reasons of staff turnover are out of the scope of this paper. The dependency diagram is shown in Figure 2 (f).

The 'Excessive Error Detection' of the RIMAM focuses on the risk of presence of excessive amount of errors that may be identified with the passage of time. The RIMAM proposes that every developer that develops the code must unit test its piece of code and any error identified should be corrected immediately. After the developer is done with the initial unit test the codes developed by the group of developers are to be integrated and multiple test are to be conducted in order to ensure that the pieces of code work fine as a unit as well. If the software is semantically and syntactically correct after integration it is assumed ready for stress testing and system level testing after which the software can be declared as successful. In an effort to identify that this specific risk factors is influenced which risk factors that are discussed in this paper it is identified that there are seven risk factors that directly or indirectly contribute in increasing the amount of errors. The factors are: 'Incorrect requirements', 'less reusable code', 'tightened delivery deadline', 'technical and human in-experience' and 'staff turnover'. The dependency diagram is shown in Figure 2 (g).

The risk factor 'preservation of intellectuals is directly dependent to the staff turnover as the intuition itself is dependent on experience, and retention of experienced people is a must to maintain the intuition level in an organization.

The detailed step-wise avoidance/management strategies are also proposed for the remaining variables of RIMAM. Some variables, for example, 'backup issues', 'developer's faithfulness' etc. do not have any flow-chart against them. In such cases, where there are less descriptive strategies available, the flow-chart has not been developed. It has been observed that risks factors, somehow, are dependent on each other. This dependence can be strong or weak. A factor is considered strong if it directly affects on other risk factors and a risk factor is considered weak if it affects some risk factors indirectly. The dependence diagram is shown in Figure 2 (h).

HANDLING AND AVOIDANCE MECHANISM

"Handling and avoidance mechanism", discusses the handling and avoidance strategies against each risk factors, presented in Table 1.

Immature requirements

(i) Multiple requirement acquisition approaches must be used; this includes the questionnaires, interviews and direct communication.
(ii) Facilitated application specification techniques (FAST) (Pressman, 2000) should be used to ensure the elaborated understanding of the requirements at both ends, that is, the customer and developer. The customer must also allow the development team to have a flexible schedule if the requirements are expected to change dynamically.
(iii) The development team must be familiar with the Enhanced Information Deployment (Pressman, 2000) technique, to take care of the default requirements that are not explicitly mentioned by the customer (Bell and Thayer, 1976).

Divergent estimation of cost and time

(i) The development team while bidding for the project must have a clear idea of the requirements that are explicitly stated and also of those that are expected by default.
(ii) If the funding and time are not flexible, the incremental model (Boehm, 1998) of development may be a solution.
(iii) The development team must try to find the maximum amount of reusable code, the availability of reusable code will have three dimensional positive effects. Firstly, it will decrease the time required for the software development by making available the code that was to be developed if the reusable code were not available. Secondly, it will decrease the cost of development as less development is required in the presence of reusable code, the higher the usage of re-usable code the lower the cost of software development comes. Thirdly, the re-usable code is already tested component and hence does not require re-testing, therefore, saving time of testing the component.
(iv) The team of experienced developers and management may decide, in consultation with the customer, that if there are any scrub able requirements that may not harm the overall working of the software. Such requirements may be eliminated to save time and cost (Baskeles et al., 2007).
(v) Clean room engineering may not be implemented in the projects that have tight time and cost schedule.

Massive user stress

(i) The developer, if possible, must design and develop the system to tolerate with the extra burden as well.
(ii) The developers must also do the extensive stress testing to ensure that the software is capable of handling the load and stress of the users.

Less reusability

(i) While estimating for the project's cost and resource requirement, the developers must know that what amount

of software is available for re-use, this should be an rational decision as, if the reusable code is not available the effort to develop such code will be duplicated. If the component is to be developed, it is necessary that a clean room engineering approach is applied is the development so that the time required for testing the component is minimized if not completely eliminated (Matsumura and Yamashiro, 2008).
(ii) The best developer, among the available lot, should be deployed to develop the components so that the expected time on development and testing is minimized.

Project delivery milestones

(i) The managers somehow try changing the circumstances because of the deadline pressure or because of the orientation of new requirements. The development team and management of the development firm must have the foreseeing capability, and should try adhering to the dynamic circumstances without disturbing the firm itself.
(ii) The FAST approach may be used to speed up the requirement acquisition, thus decreasing the negative impact of tightened deadlines.

Funding un-certainty

(i) In order to ensure that funding issues remain in order, the development team must first ensure that the software is developed within time, developing within time will not only help to improve the revenues and profits but would also ensure that the funding remains available throughout the software development lifecycle. It is also important that friendly relationship is maintained with the funding agency.
(ii) Along with the cordial relationship with the funding agency, it is also important that the funding agency is kept updated regarding the progress of the software development process, and also any problem during the process.

Over-optimistic technology perceives

(i) The decision about the choice of technology should be taken only after a very through consideration of the available tools and technologies and only by the experienced practitioners after discussion with customer (MacManus, 2000).
(ii) The tool chosen should not only be acceptable to the customer but the customer should have necessary training on the tool. It is also important for the customer to argue with the development firm about the future acceptability of the product being developed by using that specific tool.

Staff inexperience

(i) The development firm can keep its employees updated by offering them training on the emerging tools (Lui and Chan, 2004).
(ii) The firm may hire the new graduates from the leading universities, having some knowledge of the current tools. This approach has been observed to be extremely helpful in not only fulfilling the industry-academia gap but in also producing the quality products for the industry by using the knowledge imparted by the academia (Lui and Chan, 2004).
(iii) It is important that the teams are made for each project. Developing the team structure will help in not only promoting the efficiency of the work but will also help in providing experience to new members.

Staff turnover

(i) The employer should keep the estimations of the salaries available in the market for experienced people (Pressman, 2000).
(ii) The employer may offer the services like, free family medical; children school fee, car allowance, house rent, etc in order to keep the employee attracted. The employer should provide other social gathering and meeting opportunities to the employees, in order to help establish a family culture at the organization.
(iii) The employer must try to keep the employees updated and should provide the employees with chances to refresh their knowledge about the emerging tools and technologies (Pressman, 2000). The employer may also introduce a loan scheme to help the needy individuals and the return may be in easy installments, without or at a minimal interest rate.
(iv) It is necessary that the employer try maintaining the respect and honor of the employees, and it is never compromised in any situation. The employer may also introduce a bonus scheme to make the employees a part of the profit that the firm gains. This would give a sense of ownership to the employee and the employee will try to deliver according to the best of his capabilities.

Backup issues

(i) Backup must be taken at multiple sites, so that in case of any physical or technical damage the backup itself remains intact. The management may try to introduce the paperless environment in the firm; this would help in maintaining the efficient, secure and traceable working environment.
(ii) The backup sites may be frequently updated and the updates should be inspected regularly to reduce the chances of any data not being updated on the server.

The team structures should be observed in the development environment; this will improves the working environment and will decrease the dependency on individuals (Moore et al., 2005).

Natural disasters

(i) Proper smoke detectors and fire alarms must be installed in the building to detect the fire and also the emergency exit should be provided in case of any emergency.
(ii) The organization must also ensure that the building codes have been followed and the structure is according to the prescribed standards. With the orientation of more earthquakes recently in the world, it is also important that the building structure is developed in a way that it can absolve the earthquake shocks of an adequate level.

Excessive error detection

(i) Although testing techniques can help in identifying errors yet it is more appropriate to try enforcing the clean room engineering approach (Share and Amold, 1996). Along with the availability of the inspections, the developer must unit test the piece of software that he is developing and must ensure that the code is free of errors and also that it is according to the prescribed requirements (IEEE Standards, 1999). It is important as individual components may work fine but the integrated application may still not work, because of the run-time and integration errors (Moore et al., 2005).
(ii) The organization must adopt the team structure in the software development. The teams can help each other to test the code and also to ensure that the test cases are correctly designed and are efficiently handled (Shahzad and Afzal, 2005).
(iii) It is suggested that the jump to a new technology should not be made without adequate thinking and must be supported by the discussion and should be a result of a decision governed by the logical thinking.
(iii) Sometimes there are so many errors identified in a piece of code that only a re-development is the solution. A re-development must logically be completed in much higher speed as compared to the initial development (Shaktivel, 2002).
(iv) It is also important that the testing process works fine, that is, identification of too many errors can still be less harmful as compared to the ignoring errors or un-identified errors.

Developer’s faithfulness

(i) The Human Resource (HR) department must ensure that the person they are hiring is adequately trustable and owes a good employment history. The organization may also opt to take the employees from the accredited universities and resource providers so that only, already verified, individuals can find a place in the organization.
(ii) The organization may decide to hire the employees based upon the references or recommendation of their existing employees or someone may provide the guarantee for the employee for the purpose of reliability and trust.
(iii) Backup must be taken at multiple sites, so that in case of any physical or technical damage the backup itself remains intact. The backup sites may be frequently updated and the updates should be inspected.

Preservation of intellectuals

(i) It has been observed that the experienced individuals can help in estimating the cost, budget and manpower of any project by just using their intuition (Erin and Gloria, 2005). The guess provided by them is generally accurate, and thus causes a huge benefit for the organization. The organization must do adequate effort to retain such people and should continue befitting from their experience.
(ii) Talented individual must be attached to work with the experienced individuals so that they can learn that how the estimations can be made by using the previous knowledge and intuition (Naur, 1985).

COMPARISON WITH EXISTING APPROACHES

Mitigation and avoidance of software risk factors has been in active consideration since some decades and many researchers have worked in this domain.

Danny (2006) has worked to reduce operational risks by improving the software quality (We name his work as Approach α). Danny focuses on the classification and quantitative evaluation of removing the software risks by effective software management, thus contributing to the classified risk mitigation. Danny focuses that instead of spending (rather wasting) resources on the handling and mitigation of risk factors, take preemptive actions so that risk can’t occur and are not introduced into the software system. Danny’s emphasis on the effective management of risks includes the technology, personnel, environment and infrastructure management. Danny has strongly extended the need for effective management of personnel resources, specially. Along with the personnel management Danny has strongly emphasized the need for effective technology utilization to support the software development process and to ensure that it is free from most common errors. This can be effectively done by following the CMM (capability maturity model) that helps in the effective management of software development life cycle to develop the software. As the process matures, not only the likelihood of orientation of errors decreases but also the probability that the risks (even if they arrive) will be very actively handled, and thus ensuring that a huge amount of resources can be saved. Danny’s

Table 2. Time and budget requirements for different risk management approaches.

Approach	Project scale			Budget availability			Time	
	S	M	L	A	T	IA	A	C	S
α	√		√	√			√		
β			√	√			√		
Π	√			√			√		
€	√	√	√		√			√	√

approach may be very suitable for small projects or for the large projects that have adequate funding, as the improvement of software process quality, in itself, is as resource consuming as the software development itself is.

Alge et al. (2003) have emphases on the effective handling of risks and problems in the software development lifecycle (we name this approach as π) and in team structure by the usage of knowledge building process and effective communication. The sharing of knowledge among team members ensures that the individuals do not become an integral part of any software development team, and teams can work smoothly even without them, thus reducing the person dependence and maturing the system, which reduces the chances of any in-appropriate moves from the staff working on the project. This research only focuses on the issues that are caused by the staffing proportions, thus, this approach can only be used in a multi-team environment specifically, with no description of the funding and resources to be utilized. As this approach is not sufficient to handle the risks adequately, it is not recommended for the handling and management of SDLC.

This paper proposes the approach (we name as €) of continuous observation and embarks the managerial and role based activation, thus ensuring that every role assigned to each individual is performed with accuracy and perfection. This approach provides the handling and mitigation of risk factors through a process that is very sensitive to any risk factor and thus easing the process of risk identification. Identification, being the matter of utmost importance, is the key activity and after the identification has been done, this approach focuses to minimize / avoid the risk factors. Risks are handled and rectifications are done in the affected area if a risk factor has damaged the process to some extent. In, under handing project scale (S=small, M=medium, L=large), under heading budget availability (A=adequate, T=tight, IA=inadequate) and under time heading (A=ample, C=critical and S=short) (Table 2).

Conclusion

Software development process is complex and requires efficient handling of the available resources. Poor planning invites risk factors that are very difficult to deal with. The paper unleashes the possible strategies to avoid or overcome risk, once they have been identified in a software process. Although a complete list of software risk factors is impossible to produce, as the risk factors keep on growing with the new tools and technologies, yet a comprehensive list has been considered for providing knowledge about the handling and avoidance mechanism. In the last three decades ample stress has been given on the identification, management, avoidance and handling of risk factors. This paper after having identified the risk factors, proposes the avoidance and mitigation strategies for each risk factor based on the frequency of their occurrence. The software houses that are developing the small and medium software can especially benefit by following the avoidance strategy.

ACKNOWLEDGMENT

This work is supported by the Research Center of College of Computer and Information Sciences in King Saud University. The authors are grateful for this support.

REFERENCES

Alge BJ, Witheoff C, Klein HJ (2003). When does the medium matters? Knowledge building experiences and opportunities in decision making teams. Organ, 91(1): 26-37.
Armestrong R, Adens G (2008). Managing software project risks. TASSC Technical Paper. USA.
Baskeles B, Turhan B, Bener A (2007). Software effort estimation using machine learning method. 22nd Int. Symp. Comput. Inf. Sci., pp. 1–6.
Bell TE, Thayer TA (1976). Software Requirements - Are they really a problem? Proceedings of the 2nd International Conference on Software Engineering , San Francisco, California, United States, pp. 61–68.
Boehm BW (1998). Software risk management: Principles and practices, p. 13.
Burns RN, Dennis AR (1985). Selecting the appropriate application development methodology. SIGMIS Database, 17(1): 19-23.
Danny L (2006). Enterprise Software Risk Reduction. Review, 12: 1-15.
Erin B, Gloria M, Tammie DH (2005). Team size and technology fit: Participation, awareness, and rapport in distributed teams. IEEE Trans. Prof. Comm., 48(104): 68-77.
Hall T, Beecham S, Verner J, Wilson D (2008). Impact of staff turnover on software projects: The importance of understanding what makes software practitioner's tick. Proc. ACM SIGMIS CPR Conf. Comput. Personnel Doctoral Consortium Res., Charlottesville, VA, USA, 2008.

Higuera RP, Haimes YY (1996). Software Risk Management. Technical Report CMU/SEI-96-TR-012 ESC-TR-96-012.

Hoodat H, Rashidi H (2009). Classification and Analysis of Risks in Software Engineering. WASET, pp. 446-452.

IEEE Standards Board (1999). IEEE Standards: Software Engineering, Two: Process Standards.

Jones C (1996). Patterns of software success and failure. J. Defense Softw. Eng., July 1998, pp. 13-18.

Lui KM, Chan KCC (2004). Test Driven Development and Software Process Improvement in China. Lect. Notes Comput. Sci., Springer, pp. 219-222.

MacManus J (2000). Risk Management in Software Projects. Comput. Wkly Prof. Ser. Elsevier, pp. 4-18.

Moore RW, Jaja JF, Chadduck R (2005). Mitigating risk of data loss in preservation environments. 22nd IEEE / 13th NASA Goddard Conf. Mass Storage Syst. Technol., pp. 39–48.

Naur P (1985). Intuition in software development. Proc. Int. Joint Conf. Theory Pract. Softw. Dev. (TAPSOFT) Formal Methods Softw., Berlin, Germany, pp. 60–79.

Persson JS, Mathiassen L, Madsen TS, Steinson F (2009). Managing risks in distributed software projects: An integrative framework. IEEE Trans. Eng Manage., 56(3).

Pressman RS (2000). Software engineering: A practitioner's approach. 5th ed. McGraw-hill, pp. 151-159.

Robert NC (1989). Software engineering risk analysis and management. McGraw-Hill, Inc., New York, NY, USA, p. 325.

Rothfeder J (1988). It's Late, Costly, and incomplete-But Try Firing a Computer System. Bus. Week, pp. 64-65.

Sakthivel S (2002). A decision model to choose between software maintenance and software redevelopment. Department of Accounting and MIS, Bowling Green State University, USA.

Session R (2009). The IT complexity Crisis: Danger and Opportunities. A white paper by Roger Session.

Shahzad B, Afzal T (2005). Enhanced risk analysis and relative impact factorization. 1st International Conference on Information and Communication Technology. IBA Karachi, pp. 290-296.

Shahzad B, Iqbal J (2007). Software Risk Management – Prioritization of frequently occurring Risk in Software Development Phases using Relative Impact Risk Model. 2nd International Conference on Information and Communication Technology (ICICT2007), IBA Karachi, pp. 82-87.

Share SW, Kouchakdjian A, Arnold PG (1996). Experience Using Cleanroom Software Engineering. IEEE Softw., 13(3): 69–76.

Westfall L (2009). Software is a risky business. USA, 2009.