A Simulation Approach for Impact Analysis of Requirement Volatility Considering Dependency Change

Junjie Wang[1,2], Juan Li[1], Qing Wang[1], He Zhang[3], Haitao Wang[1,4]

[1]Laboratory for Internet Software Technologies, Institute of Software
Chinese Academy of Sciences, Beijing 100190, China
[2]Graduate University of Chinese Academy of Sciences, Beijing 100039, China
[3]National ICT Australia, University of New South Wales, Sydney, Australia
[4]nfschina Inc, Beijing
{wangjunjie, lijuan, wq, wanghaitao}@itechs.iscas.ac.cn, he.zhang@nicta.com.au

Abstract. Requirement volatility is a common and inevitable project risk which has severe consequences on software projects. When requirement change occurs, a project manager wants to analyze its impact so as to better cope with it. As the modification to one requirement can cause changes in its dependent requirements and its dependency relationship, the impact analysis can be very complex. This paper proposes a simulation approach DepRVSim (Requirement Volatility Simulation considering Dependency relationship) to assessing this sort of impact. We abstract the general patterns of the influence mechanism, which may trigger modification in its dependency relationship and bring changes in other requirements through dependency. DepRVSim can generate such information as the probability distribution of effort deviation and schedule deviation. As a proof-of-concept, the applicability of DepRVSim is demonstrated with an illustrative case study of a real software project. Results indicate that DepRVSim is able to provide experimental evidence for decision making when requirement changes.

Keywords: Requirement Volatility; Requirement Dependency; Software Process Simulation;

1 Introduction

It is widely reported that requirements often change during the software/system development process. These changes are caused by several factors, such as evolving customer needs, errors in original requirements, technological changes, and changes in the business environment or organization policy. Requirements volatility often results in cost and schedule overruns, unmet functions and, at times, cancelled projects [1, 2]. Houston et al. [3] described an approach to modeling risk factors and simulating their effects. The effects of six common and significant software development risk factors were studied, including inaccurate cost estimation, staffing attrition and turnover, etc. Simulation results reflected that requirements volatility is the most significant risk factor modeled.

Most requirements cannot be treated independently, since they are related to and affect each other in complex manners [4, 5]. When a certain requirement changes, other requirements would be influenced through dependency relationship in ways not intended or not even anticipated. Apart from that, the requirement dependency relationship would not remain the same when requirement changes happen. Hence, during the impact analysis of requirement changes, dependency relationship is one of the important factors need to be carefully considered.

Several simulation approaches have emerged to assessing the impact of requirement volatility on project performance. Pfahl et al. [6] built a system dynamic simulation model for Siemens Corporate Technology to demonstrate the impact of requirement volatility on project duration and effort. His work modeled the relationship between unstable definition of requirements and rework cycles, rework cycles and development productivity, development productivity and project duration, and so on. This model captured a specific real-world development process in sufficient detail, but was not easily adaptable to new application contexts. Ferreira et al. [7] utilized empirical survey results and built an executable system dynamics model to demonstrate the impact of requirement volatility on cost, schedule and quality. These studies are conducted applying system dynamics simulation approach. This type of research focuses on phenomenological observations of external behaviors of process, such as job size, overall project effort, requirement defects and so on [8].

Compared with system dynamics, discrete-event simulation allows more detailed descriptions of activity, resource and work product and more suitable for building fine-grained software process simulation models [8]. Liu et al. proposed a simulation approach to predict the impact of requirement volatility on software project plans. This discrete-event simulation model can capture internal behaviors of software process, such as traceability and dependency relationship [9]. But his approach did not consider dependency relationship in sufficient detail and did not model the changes in dependency relationship.

In this paper, we propose a simulation approach named DepRVSim (Requirements Volatility Simulation considering Dependency relationship) to analyze the impact of requirement volatility on project plan. In DepRVSim, we model the dependency relationship and traceability relationship, as well as the changes in dependency relationship. We abstract the general patterns of the influence mechanism, which may trigger modification in its dependency relationship and bring changes in other requirements through dependency. DepRVSim can generate such information as the probability distribution of schedule deviation.

Only part of the simulation approaches are validated in industrial setting. Among these case studies, many of them only apply industrial context as simulation inputs. We not only base our validation on real industrial context, but also compare model outputs with actual process data and obtain statistical results. Simulation results indicate that for 10 man hours offset from real effort deviation and 10 hours offset from real schedule deviation, DepRVSim can reach a correct rate of approximately 45% and 70% respectively. DepRVSim can assist project managers in decision making process and help understand the impact of requirement volatility in depth.

The remainder of the paper is structured as follows. Section 2 describes mechanism of DepRVSim in detail. Section 3 illustrates the applicability and usefulness of DepRVSim with the help of a case study. Section 4 discusses threats to validity.

Section 5 discusses related work. Finally, Section 6 concludes the paper and gives directions of our future work.

2 The DepRVSim Approach

DepRVSim is a discrete-event simulation approach, which adopts the framework of RVSim [9]. There are four components in DepRVSim as shown in Figure 1.

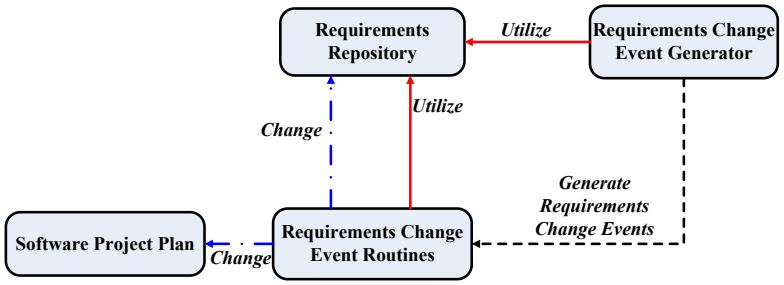

Figure 1. DepRVSim structure

Requirements Repository stores description for requirements attributes, including requirements' traceability information and dependency information. Since the realization of each requirement requires a sequence of individual tasks, the traceability information refers to the relationship between requirement and its related tasks. As most requirements are related to and affect each other, the dependency information refers to the relationship between requirements. One change on a certain requirement not only influences its related tasks through traceability, but also probably impacts other requirements through dependency, furthermore the dependency relationship can go through changes. The information in this component is fully utilized by Requirements Change Event Routines to accurately assess the impact of requirements volatility on Software Project Plan. The detailed description about this component is shown in Section 2.1.

The purpose of DepRVSim is to analyze the impact of requirements change on software project plan. So, **Requirements Change Event Generator** generates events which represent requirements changes in simulation. There are three kinds of events in DepRVSim: Requirements Addition, Requirements Deletion and Requirements Modification. The detailed description about this component is shown in Section 2.2.

When requirements change events arrive at **Requirements Change Event Routines**, the corresponding routines are started to deal with these events utilizing the information in Requirements Repository. The detailed description about this component is shown in Section 2.3.

Software Project Plan is the plan of the software project which is analyzed by DepRVSim. Software Project Plan is changed during simulation, so users can easily see how requirements volatility impacts on the project plan.

2.1 Requirements Repository

We assume a set of requirements Req_1, Req_2, … , Req_N will be developed, which stores in Requirements Repository. Req_i is defined as a tuple: (ReqId, DependencySet, RelatedTaskSet).

DependencySet denotes the set of requirement's dependency relationship. Each item in DependencySet is represented as follows: (ReqId, DepDirection, DepStrength). DepDirection specifies the dependency direction, which is IN or OUT. The IN direction denotes that other requirements depend on this one, while the OUT direction denotes that this requirement depends on others. DepStrength specifies the degree of the dependency relationship, which is STRONG or WEAK.

RelatedTaskSet denotes the set of requirement's traceability relationship. Each item in RelatedTaskSet is one of the corresponding tasks for realizing the requirement and is represented as follows: (TaskId, Type, Effort). Typical task types are design, code and test. Effort denotes the estimated effort needed to fulfill a task. Note that certain dependency relationships between tasks are applied, e.g., test cannot be started before some or all of the code has been finished.

2.2 Requirements Change Event Generator

Because the purpose of DepRVSim is to analyze the impact of requirements change on project plan, Requirements Change Event Generator generates requirements change events during simulation. Change event is described as a tuple: (ReqId, RChangeType, RChangeTime, ModifyLevel).

ReqId corresponds to the requirement which is added, modified or deleted. RChangeType defines the type of requirements change event, which are Requirements Addition, Requirements Modification and Requirements Deletion. RChangeTime is the time when requirements change event happens. ModifyLevel specifies the degree to which one requirement is modified for the change type Requirements Modification. Possible values of ModifyLevel are MAJOR, MODERATE and MINOR. ModifyLevel >0 indicates that the requirement modification is adding content, while ModifyLevel <0 indicates deleting content. They are numeric values between -1 and 1 and satisfy |MAJOR| > |MODERATE| > |MINOR|, which are calibrated based on historical project data and expert judgement.

DepRVSim allows users to specify how Requirements Change Event is generated. There are two modes for generating events:

(1) Definite events inputted by users. This mode is suitable for the situation that one requirement change request has arrived, and users want to know the impact of this change on project plan.

(2) Supposed events generated automatically according to user-defined rules. This mode is suitable for the situation that the users intend to predict the impact according to the trajectory of requirement volatility. The rules can be obtained by analyzing historical project data (like [10, 11]) or by their experience. Users can also do "what-if" analysis by setting up different rules.

2.3 Requirements Change Event Routines

Requirements Change Event Routines includes three general routines for the three types of requirements change events in simulation, which is represented as follows. Assume the changed requirement is R_i, the requirement that R_i depends on is R_{out}, the requirement that depends on R_i is R_{in}.

Requirements Addition Event Routine

This routine has three steps as follows:

- **Step1:** Add R_i to Requirements Repository with related tasks
- **Step2:** Generate R_i.DependencySet

Assume the total number of requirements is N, the parameter dper (dependency percent) of R_i is defined as follows: $dper = (N_d / N) * 100$.

N_d can be calculated easily by N and dper. dper is generated based on the uniform distribution of the type UNIFORM (min, max). The "min" and "max" represent the minimum and maximum values, respectively. We name the two parameters as dperMin and dperMax. Choose N_d requirements as ones with which R_i has dependency relationship. Randomly generate DepDirection and DepStrength.

- **Step3:** Rearrange tasks properly in Software Project Plan.

In DepRVSim, overlapping of the phases for one requirement is not allowed. Design tasks have precedence relationship the same as the dependency of requirements related to them. For example, if design tasks T_1 and T_2 realize requirements R_1 and R_2 respectively, and R_2 depend on R_1, then T_2 must be arranged to start after T_1 is finished. In code and test phases, tasks do not have such precedence relationship, so tasks in the same phase can be parallel. In addition, there is no idle time between tasks.

Requirements Deletion Event Routine

This routine has three different steps from addition routine, which is shown as follows:

- **Step1:** Delete R_i from Requirements Repository
- **Step2:** Modify the influenced requirements

When deleting R_i from current project plan, the requirements with which R_i has dependency relationship might be influenced. The ModifyLevel of these requirements is shown in Table 1, where "none" indicates that the requirement is not influenced.

Table 1. Rule for ModifyLevel of R_{out} and R_{in} in deletion routine

R_i's ModifyLevel	DepStrength	R_{in}'s ModifyLevel	R_{out}'s ModifyLevel
delete	STRONG	delete	none
delete	WEAK	major	none

- **Step3**: Adjust the Software Project Plan.

Requirements deletion may cause idle time between tasks, so the Software Project Plan needs to be adjusted.

Requirements Modification Event Routine

There are four steps in the routines:

- **Step1**: Modify corresponding tasks' effort of R_i

Set up a parameter emp (effort modified percent). DepRVSim distinguish the variant effort for the situation that a task has not been started and the situation that a task has been finished, which is signified by RChangeTime. Suppose the original task effort is Eff_i. If the task has not been started, the effort after modification is $Eff_i*(1+emp)$. If the task has been finished, apply the parameter reworkRate to signify this difference. The rework effort is $Eff_i*emp*reworkRate$. If the task has been started but not finished, divide the task into two parts and calculate new effort respectively.

The parameter emp is generated based on the uniform distribution of the type UNIFORM (min, max). When ModifyLevel = major, the distribution is UNIFORM(moderate, major). When ModifyLevel = moderate, the distribution is UNIFORM(minor, moderate). When ModifyLevel = minor, the distribution is UNIFORM(0, minor). The reworkRate is an input parameter calibrate based on particular project.

- **Step2:** Modify the dependency relationship of R_i

As experiences from software development shows that requirement dependency relationship would not remain unchanged when the certain requirement is modified, DepRVSim model this situation. When analyzing the changes in dependency relationship, we distinguish adding content and deleting content of certain requirements, as well as the direction of the dependency relationship. Detailed rules are described as follows:

Rule1: When the modification to R_i is adding its content, R_i might newly depend on other requirements.

Set up a parameter dperAdd to represent the dependency percent of newly added dependency relationship. We generate dperAdd based on the same uniform distribution as emp. We also apply an input parameter f_{Add} to revise the generated dperAdd. The parameter f_{Add} is different among software projects and can be decided based on expert judgement.

The number of newly added dependency relationship can be calculated using dperAdd, f_{Add} and N, which is similar with dper. Randomly choose requirements with which new dependency emerges. Generate the dependency relationship for R_i where DepDirection is OUT and DepStrength is randomly generated.

Rule2: When the modification to R_i is adding its content, for the dependency relationship that R_i depends on others, current dependency might be strengthened.

We apply a parameter dpermp to represent the modified percent of dper. Generate dpermp based on the same uniform distribution as emp. The number of changed dependency relationship can be calculated by N * dper * dpermp.

Randomly choose the influenced relationship. If current DepStrength is WEAK, change it to STRONG. If current DepStrength is STRONG, keep it unchanged.

Rule3: When the modification to R_i is deleting its content, for the two kinds of dependency relationship, which are R_i depends on others and other requirements depend on R_i, the current dependency relationship is weakened or disappears.

Apply the parameter dpermp to decide the number of changed relationship as Rule2. Randomly choose the influenced dependency relationship. If current DepStrength is STRONG, change it to WEAK. If current DepStrength is WEAK, delete the corresponding dependency relationship.

- **Step3:** Modify the influenced requirements

When modifying R_i from current project plan, the requirements with which R_i has dependency relationship might be influenced. The ModifyLevel of these requirements is shown in Table 2.

Table 2. Rule for ModifyLevel of R_{out} and R_{in} in modification routine

R_i's ModifyLevel	DepStrength	R_{in}'s ModifyLevel	R_{out}'s ModifyLevel
major	STRONG	major	none
major	WEAK	moderate	none
moderate	STRONG	moderate	none
moderate	WEAK	minor	none
minor	STRONG	minor	none
minor	WEAK	none	none

- **Step4:** Adjust the Software Project Plan

Requirement modification may change duration and precedence relationship of related project tasks, or cause idle time between tasks, so the Software Project Plan needs to be adjusted.

3 Case Study

The method in this paper is mainly applied to the matured software organizations, such as the ones which have achieved CMMI (Capability Maturity Model Integration) maturity level 4 or higher. Such organizations have stable development and maintenance processes. After a long-period accumulation of process execution data, they can analyze and determine the dependency strength, the modification level and other parameters with sufficient data.

We conducted our case study in such a software organization. We utilized a real software project in this organization to demonstrate the applicability of the proposed approach and the results of the study. This software project is Software Process Management Platform-Qone [12]. With more than 600 thousand source lines of code, this product has been developed and maintained for more than 7 years. More than 300 Chinese software organizations are using this tool to manage their projects.

We applied the real development data of Qone 5.1. During the requirement analysis phase, the project was planned. During the development phase, change request were forwarded to project manager. For example, changes in business environment might require a certain requirement to be enhanced. These changes made the schedule prolonged and one or several weeks' delay is the common case.

3.1 Project Introduction

There are 24 requirements (R_1~R_{24}) generated through the requirement phase. Table 3 shows the requirement-related information, including ReqId, requirement name and the estimated task-specific efforts per requirement.

Table 4 presents the estimated task-specific productivities per developer. Productivity represents the amount of work done per hours. For example, the productivity of Dev_1 for Design task is 2 as Table 4 shows, and the Design effort of R_1 is 48 hours as Table 3 shows, then Dev_1 can perform the $Design_1$ in 48/2 hours. Productivity 0 for a task type implies that a developer is not able to perform that type of task.

Figure 2 shows the requirements' dependency information. For example, the dependency relationship between R_1 and R_2 is that R_2 strongly depends on R_1. The dependency relationship is obtained by analyzing historical project data and by expert judgement.

Table 3. Requirements information of Qone 5.1

ReqId	Requirement name	Design (man hour)	Code (man hour)	Test (man hour)	Total (man hour)
R_1	Generate new PIIDS table	48	104	90	242
R_2	Search PIIDS related	48	104	90	242
R_3	Maintain PIIDS table	48	104	90	242
R_4	Export PIIDS table	48	104	90	242
R_5	Import evaluation tools	48	104	90	243
R_6	Approve change request	44	56	73	173
R_7	Timing task notification	44	56	73	173
R_8	Table handling notification	44	56	73	173
R_9	Table selection conflict	44	56	73	173
R_{10}	Project problem submission notification	44	56	73	173
R_{11}	Identity authenticate	23	18	72	113
R_{12}	Access control	20	21	72	113
R_{13}	Data security	16	18	72	106
R_{14}	Import and export file	16	40	122	178
R_{15}	Import and export project	20	37	122	179
R_{16}	Project data matching	18	37	122	177
R_{17}	Import and export failure handling	18	43	122	183
R_{18}	Import and export information modification	18	38	122	178
R_{19}	Related project handling	16	40	110	166
R_{20}	Department report import and export	16	40	110	166
R_{21}	Add configuration files	4	3	1	8
R_{22}	Bug comment	4	3	1	8
R_{23}	Size restriction of change	4	3	1	8
R_{24}	Add links for project	4	3	1	8

Table 4. Estimated productivity of developers for different task types

Developers	Design (dimensionless)	Code (dimensionless)	Test (dimensionless)
Dev_1	2	1	1
Dev_2	1	0	2
Dev_3	1.2	2	1.4
Dev_4	1	1.5	2

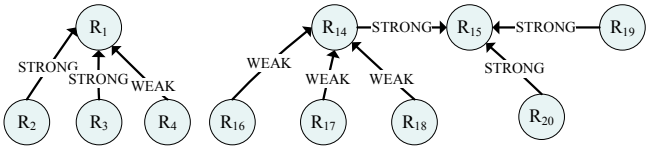

Figure 2. Dependency relationship between requirements of Qone 5.1

Software project plan specifies the planned start time and end time for each task, as well as the allocated developer for the task. Due to the limited space, we do not present the whole plan here. Part of it is shown in Figure 3.

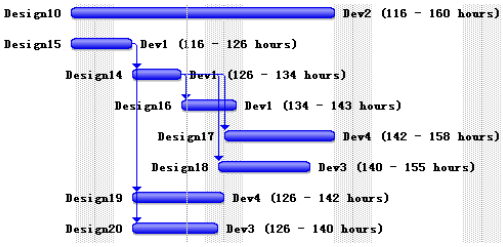

Figure 3. Part of the initial software project plan

We collected the change data of Qone 5.1, as summarized in Table 5. It has 10 requirement changes. Effort deviation and schedule deviation information is also recorded in change database. Effort deviation denotes the difference between the new total effort under requirement changes and the planned total effort. Schedule deviation is the difference between the new project duration after changes and the planned project duration. The ModifyLevel is obtained based on the actual change degree and expert judgement.

Table 5. Change data of Qone 5.1

ReqId	ModifyLevel	Effort deviation (man hour)	Schedule deviation (hour)
R_{14}	MAJOR	176	49
R_{15}	MAJOR	176	49
R_{16}	MAJOR	176	49
R_{17}	MAJOR	176	49
R_{18}	MAJOR	176	49
R_{19}	MAJOR	176	49
R_{20}	MAJOR	176	49
R_{11}	MODERATE	115	38
R_{12}	MODERATE	115	38
R_{13}	MODERATE	115	38

The parameters defined in Section 2 are set as follows: dperMin =0, dperMax = 0.4; major = 0.45, moderate = 0.3, minor = 0.15; reworkRate = 0.5; f_{Add} = 0.15. These parameters are determined by the project manager of Qone 5.1. Take reworkRate as an example, this parameter works in Step 1 of modification routine. Together with the

parameter emp, this parameter decides the rework effort for the finished tasks. Project manager can refer to similar circumstances of historical projects to obtain such information as the added workload of rework task. This parameter can then be determined through statistical techniques utilizing these project data.

3.2 Simulation Scenario and Impact Analysis

Due to limit space, we only demonstrate how Requirements Modification Event Routine works. This scenario is based on actual change data in Table 5. During project development, customers request the requirement "import and export project" to be enhanced and refined. Hence, the modification to R_{15} is adding its content. The change time is 130 hours and ModifyLevel for R_{15} is MAJOR, which is obtained in the change databases.

Note that, many of the parameters below are just random values generated based on certain distribution during this certain simulation scenario. We applied these parameters to illustrate how DepRVSim works. The ultimate simulation outcome is based 10000 simulation scenario of this kind, in which these parameters might differ among simulation scenarios. According to Requirement Modification Event Routine, there are four steps to handle this change event.

- **Step1:** Modify corresponding tasks' effort of R_{15}

R_{15} has three tasks, respectively $Desing_{15}$, $Code_{15}$ and $Test_{15}$. When this change event happens at 130 hours, $Design_{15}$ has been finished, as Figure3 shows, and the other two tasks have not been started. The original effort for $Design_{15}$ is 20 hours, as Table 3 shows. The rework effort for $Design_{15}$ is 20*emp*reworkRate. Suppose the randomly generated emp is 0.38 in this simulation scenario based on UNIFORM(0.3, 0.45). The reworkRate is 0.5, so the rework effort for $Design_{15}$ is 4 hours. The new effort for $Code_{15}$ and $Test_{15}$ can be calculated in the similar way, which is not shown due to space limit.

- **Step2:** Modify the dependency relationship of R_{15}

Current dependency relationship of R_{15} is {(R_{14}, IN, WEAK), (R_{19}, IN, WEAK), (R_{20}, IN, STRONG)} as Figure 2 shows. DepRVSim would utilize Rule1 and Rule2 to handle dependency change of R_{15}.

According to Rule1, R_{15} might newly depend on other requirements. Suppose the generated dperAdd is 0.32 in this simulation scenario based on UNIFORM(0.3, 0.45). The input parameter f_{Add} is 0.15. So the number of newly added dependency is $24*0.32*0.15 \approx 1$. Suppose the newly added dependency is (R_{10}, OUT, WEAK) in this simulation scenario.

According to Rule2, the current dependency relationship of DepDirection = IN is strengthened. dper for R_{15} is 3/24 = 0.125, suppose the generated dpermp is 0.36 in this simulation scenario, the number of changed dependency is $24*0.125*0.36 \approx 1$. Suppose the randomly chosen dependency is (R_{14}, IN, WEAK), change it to (R_{14}, IN, STRONG). The dependency relationship of R_{15} after change happens is {(R_{10}, OUT, WEAK), (R_{14}, IN, STRONG), (R_{19}, IN, WEAK), (R_{20}, IN, STRONG)}.

- **Step3:** Modify the influenced requirements

There are requirement changes in these requirements that depend on R_{15}, which are R_{14}, R_{19} and R_{20}. These requirement changes are reflected through the changes in corresponding tasks' effort. When this change event happens at 130 hours, $Design_{14}$, $Design_{19}$ and $Design_{20}$ are all on-going tasks, as Figure 3 shows. The effort after modification can be calculated similar with Step1.

- **Step4:** Adjust the Software Project Plan

The adjusted project plan of Figure 3 is shown in Figure 4. The red box denotes the rework for finished tasks, while the green box denotes the modification for unfinished tasks. The purple box denotes the tasks which are indirectly influenced. We can see from Figure 4 that due to the postponement of $Design_{14}$ and rework of $Design_{15}$, Dev_1 is late for conducting $Design_{16}$. And the follow-up tasks would be influenced.

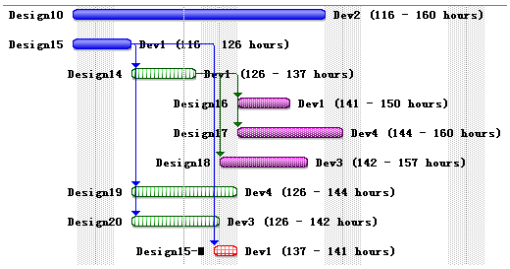

Figure 4. Part of the adjusted software project plan

We simulated 10000 times for this change event and the simulation outcomes of effort deviation and schedule deviation are shown in Figure 5 and Figure 6. The reason for the difference between effort deviation and schedule deviation is that the added effort may be performed by several developers in parallel.

The real development data in Table 5 showed that the effort deviation and schedule deviation for this requirement change are respectively 176 man hours and 49 hours. From Figure 5 and Figure 6, the probability that the simulated effort deviation has 10 man hours offset with real project data is 41.7%, while the probability for 10 hours offset of schedule deviation is 65.6%.

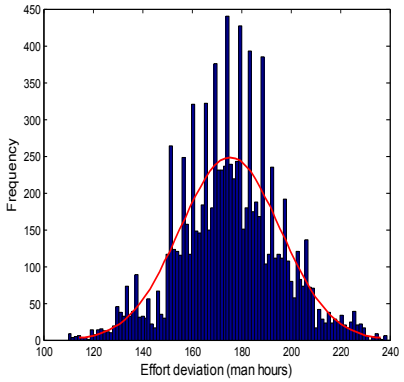

Figure 5. Simulation results of effort deviation

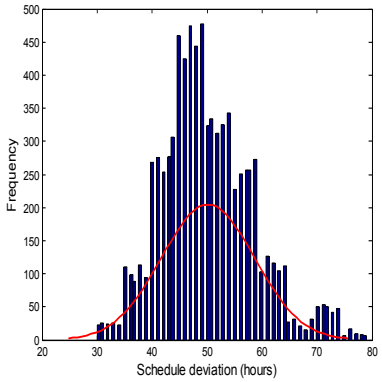

Figure 6. Simulation results of schedule deviation

3.3 Evaluation of DepRVSim

We utilize the change data in Table 5 to carry out the evaluation of DepRVSim. We simulate these requirement change events and generate the effort deviation and schedule deviation information. Our work obtains the minimum, maximum and average value, as well as the probability of offset with real project data. These results are listed in Table 6 and Table 7.

Table 6. Effort deviation information of DepRVSim

ReqId	Minimum effort deviation	Maximum effort deviation	Average effort deviation	Effort deviation±10	Effort deviation±20
R_{14}	110	238	172	42.6%	67.5%
R_{15}	110	235	174	41.7%	66.5%
R_{16}	110	232	172	42.9%	67.2%
R_{17}	105	212	166	41.3%	66.0%
R_{18}	105	218	168	42.1%	68.2%
R_{19}	102	215	172	43.3%	69.2%
R_{20}	102	214	172	43.4%	69.2%
R_{11}	67	155	110	45.0%	63.6%
R_{12}	68	156	110	46.7%	65.5%
R_{13}	57	145	99	49.2%	64.7%

Effort deviation±K signify the probability that simulation results have K man hours offset from real effort deviation. Take R_{14} as an example, Table 5 shows that the real effort deviation is 176 man hours, so effort deviation±10 means the probability that the simulated effort deviation falls into the interval from 166 man hours to 186 man hours. The results in Table 6 show that for 10 and 20 man hours offset from real effort deviation, DepRVSim can predict correctly in the probability of around 45% and approximately 70%.

Table 7. Schedule deviation information of DepRVSim

ReqId	Minimum schedule deviation	Maximum schedule deviation	Average schedule deviation	Schedule deviation ±5	Schedule deviation ±10
R_{14}	34	75	52	40.2%	68.5%
R_{15}	34	77	52	37.9%	65.6%
R_{16}	34	76	54	37.8%	64.0%
R_{17}	35	73	54	39.7%	63.9%
R_{18}	36	76	54	40.9%	66.4%
R_{19}	33	74	50	41.1%	68.9%
R_{20}	33	73	51	41.3%	68.7%
R_{11}	32	58	42	45.2%	69.2%
R_{12}	29	58	43	48.8%	67.4%
R_{13}	32	55	40	46.3%	64.4%

Similar with effort deviation information, the results in Table 7 show that for 5 and 10 hours offset from real schedule deviation, DepRVSim can reach a correct rate of 49% and 70%.

We can notice that the simulated schedule deviation is often bigger than the actual project data. Through interviews with the project manger of this project, we found that there is rescheduling process to better utilize the human resources during requirement changes in real software project. However, in our work, the added task effort caused by changes is assigned to the original developer. Even so, the simulation results accord well with the real effort deviation and schedule deviation. Project manager can refer to these simulation results to decide whether to accept a particular change request or not.

4 Threats to Validity

From running a series of simulation scenarios we have gained additional insight into the nature of requirement volatility. The results from our case study provide an indication that there is a good chance to support project managers in decision making about requirement change request. In order to better judge the meaningfulness and applicability of the results, we have to carefully check their validity status.

Construct validity: a central construct in our work is the mechanism for impact of requirement volatility. Since no generally accepted mechanism for requirement change, we had to base our routines on empirical study and real software development process. We assume that this impact can be model through dependency relationship and traceability relationship. Another construct in our work is the mechanism for changes in dependency relationship. We assume that deleting requirement content might weaken its current dependency, while adding requirement content might strengthen its dependency generally. We also distinguish the direction of these dependency relationships. It is shown that the applied routines work well in general. However, as is the case for routines in general, we cannot precisely evaluate the quality of the solution for other particular project process. This might also impact the comparability between the different projects slightly.

Internal validity concerns the extent to which observed differences can be attributed to an experimental manipulation. Since our work heavily relies on a computerized simulation model, in principle, this should be one of the easiest types of validity to maximize. The simulated environment offers the experimenter a sterile setting in which entities adhere strictly to whatever routines they are assigned and within selected parameter bounds.

External validity is the degree to which the findings in a local setting, containing a single set of sampling units, are applicable to the population of sampling units as well as other setting. In our particular case, external validity is enhanced in many ways. First of all, we base our study on real software project and apply real project change data to do the evaluation. Apart from that, we provide customizable parameters in our model and users can assign their own value according to their specific software projects. These all increase the external validity of our results. However, to further prove external validity, we need to conduct our evaluation on more software projects.

While stressing the limitations of the applicability of the results, we also want to emphasize that the overall methodology is applicable more broadly in the context of simulation-based analysis. The only difference would be the adjustment of the simulation model and the inherent heuristics.

5 Related Work

The idea of using software process simulation for predicting project performance or evaluating processes is not new. Beginning with pioneers like Abdel-Hamid [13], Bandinelli [14], Gruhn [15], Kellner [16], Scacchi [17], dozens of process simulation models have been developed for various purposes. The primary purposes of simulation models are summarized as: strategic management, planning, control and operational management, process improvement and technology adoption, as well as training and learning [18].

Planning involves the prediction of project effort, cost, schedule, quality, and so on. The impact analysis of requirement volatility is among this purpose. Pfahl et al. [6] built a simulation model for Siemens Corporate Technology to demonstrate the impact of requirement volatility on project cost and effort. Ferreira et al. [7] derived related factors from empirical survey and built a system dynamic simulation model to demonstrate the impact of requirement volatility on cost, schedule and quality.

Control and operational management involves project tracking and oversight. Project can be monitored and compared against planned values computed by simulation, to help determine when corrective action may be needed. The management of software development risks is within this purpose. Houston et al. [5] described an approach to modeling risk factors and simulating their effects as a means of supporting certain software development risk management activities. His approach considered requirements volatility as one of the six risk factors and simulated its influence on project cost and duration.

Apart from software process simulation, empirical study is often applied in the impact analysis of requirement volatility on development productivity [19], project cost [20], defect density [21], project effort [20], project schedule [22], change effort [23] and software release planning [24]. Zowghi et al. [19] conducted a survey of 430 software development companies in Australia, and the results showed that over 80% projects were late because of requirement volatility. Stark et al. [22] developed a regression analysis model to predict the schedule change percent due to requirements volatility. These empirical studies can serve as the basis for parameter calibration and general mechanism of simulation model.

The simulation method presented above focus on phenomenological observations of external behaviors of software process. Our model focused on the study of the internal details and working of process. We modeled the changes in dependency relationship when requirement changes occur. This is common in software development and a key factor for impact analysis of requirement volatility, but is not well explored yet. We abstracted the general patterns of dependency changes and provide customizable parameters for users' own process models.

6 Conclusions and Future Work

In this paper, we presented a simulation approach DepRVSim which can predict the impact of requirement volatility on software project plans. DepRVSim adopts discrete-event simulation which is able to provide many kinds of project data for users besides the project effort and schedule in the case study.

Our primary contribution is modeling the dependency relationship to assist the impact analysis of requirement volatility. Besides, we evaluate the effectiveness and applicability of DepRVSim applying the real software development data.

One significant feature of DepRVSim is that it supports fine-grained requirement change and detail change impact analysis. This feature not only provides users with such information as probability distribution of effort deviation and schedule deviation, but also assists project managers to understand the impact of requirements volatility deeply.

It should be pointed out, however, that the presented material is just the starting point of the work in progress. Future work will focus on calibration of model parameters applying data mining techniques. Another enhancement aims at validation of the proposed approach in more industrial environment, improvement of model usability, and – more importantly – enhancement of the DepRVSim model. Enhancement of DepRVSim will in particular aim at adding a heuristic that take manpower resources into consideration.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under grant No.60903050, No.60803023 and No.90718042, the National Basic Research Program (973 Program) of China under grant No.2007CB310802, the National Hi-Tech Research and Development Program (863 Program) of China under grant No.2007AA010303, as well as the Graduate Foundation of ISCAS under grant No.ISCAS2009-GR.

References

1. Boehm, B. W.: Software Risk Management: Principles and Practices. IEEE Software 8(1), 32-41 (1991)
2. Kotonys , G., Sommerville, I.: Requirements Engineeringn Process & Techniques. John Wiley & Sons (2002)
3. Houston, D. X., Mackulak, G. T., Collofello, J. S.: Stochastic simulation of risk factor potential effects for software development risk management. JSS 59(3), 247–257 (2001)
4. Dahlstedt, Å., Persson, A.: Requirements interdependencies - Moulding the State of Research into a Research Agenda. The Ninth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2003), Klagenfurt/Velden, Austria, 71-80 (2003)
5. Wohlin, C., Aurum, A.: What is important when deciding to include a software requirement in a project or a release? Fourth International Symposium on Empirical Software Engineering, Noosa Heads, Australia, 17− 18 November (2005)
6. Pfahl, D., Lebsanft, K.: Using Simulation to Analyze the Impact of Software Requirements Volatility on Project Performance. Information and Software Technology 42(14), 1001–1008 (2000)
7. Ferreira, S., Collofello, S. J., Shunk, D., Mackulak, G.: Understanding the Effects of Requirements Volatility in Software Engineering by Using Analytical Modeling and Software Process Simulation. The Journal of Systems and Software 82 (2009), 1568–1577
8. Zhang, H., Kitchenham B., Pfahl, D.: Software Process Simulation Modeling: An Extended Systematic Review. Proc. International Conference on Software Process (ICSP 2010), LNCS, vol. 6195, 309–320. Springer, Heidelberg (2010)

9. Liu, D., Wang, Q., Xiao, J., Li, J., Li, H.: RVSim: A Simulation Approach to Predict the Impact of Requirements Volatility on Software Project Plans. International Conference on Software Process (ICSP 2008), LNCS, vol. 5007, 307–315. Springer, Heidelberg (2008)
10. Nurmuliani, N., Zowghi, D., Powell, S.: Analysis of Requirements Volatility During Software Development Life Cycle. In: Proceedings of the 2004 Australian Software Engineering Conference (ASWEC 2004), Melbourne, Australia (2004)
11. Nurmuliani, N., Zowghi, D., Williams, S.P.: Characterising Requirements Volatility: An Empirical Analysis. In: Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE 2005), Noosa, Australia (2005)
12. http://qone.nfschina.com/qone/
13. Abdel-Hamid, T. K., Madnick, S. E.: Software Projects Dynamics – an Integrated Approach. Prentice-Hall, Englewood Cliffs (1991)
14. Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, M., Picco, G. P.: Modeling and Improving an Industrial Software Process. IEEE Trans. on Soft. Eng. 21(5), 440–453 (1995)
15. Gruhn, V., Saalmann, A.: Software Process Validation Based on FUNSOFT Nets. Proc.EWSPT 1992, 223–226 (1992)
16. Kellner, M. I., Hansen, G. A.: Software Process Modeling: A Case Study. Proc. AHICSS 1989, vol. II - Software Track, 175–188 (1989)
17. Mi, P., Scacchi, W.: A knowledge-based environment for modeling and simulating software engineering processes. IEEE Trans. on Know. and Data Eng. 2(3), 283–294 (1990)
18. Kellner, M. I., Madachy, R. J., Raffo, D. M.: Software process simulation modeling: Why? What? How? The Journal of Systems and Software 46(2/3), 91–105 (1999)
19. Zowghi, D., Offen R., Nurmuliani, N.: The Impact of Requirements Volatility on the Software Development Lifecycle. Proc. International Conference on Software Theory and Practice (IFIP World Computer Congress 2000)
20. Zowghi, D., Nurmuliani, N.: A Study of the Impact of Requirements Volatility on Software Project Performance. Proc. Asia-Pacific Software Engineering Conference (APSEC 2002), Gold Coast, Australia, 3–11(2002)
21. Malaiya, Y. K., Denton, J.: Requirements Volatility and Defect Density. Proc. International Symposium on Software Reliability Engineering (ISSRE 1999), 285–294
22. Stark, G., Skillicorn, A., Ameele, R.: An Examination of the Effects of Requirements Changes on Software Releases. CROSSTALK, The Journal of Defense Software Engineering, 11–16 (December 1998)
23. Nurmuliani, N., Zowghi, D., Williams, S.: Requirements Volatility and Its Impact on Change Effort: Evidence Based Research in Software Development Projects. Proc. Australian Workshop on Requirements Engineering (AWRE 2006), Adelaide,Australia
24. Al-Emran, A., Pfahl, D., Ruhe, G.: Decision Support for Product Release Planning based on Robustness Analysis. Proc. IEEE International Requirements Engineering Conference (RE 2010), 157-166