Requirements Eng (2016) 21:161–184
DOI 10.1007/s00766-014-0215-x

ORIGINAL ARTICLE

Analyzing and predicting software integration bugs using network analysis on requirements dependency network

Junjie Wang · Qing Wang

Received: 22 January 2014 / Accepted: 27 October 2014 / Published online: 7 November 2014

Abstract Complexity, cohesion and coupling have been recognized as prominent indicators of software quality. One characterization of software complexity is the existence of dependency relationships. Moreover, the degree of dependency reflects the cohesion and coupling between software elements. Dependencies in the design and implementation phase have been proven to be important predictors of software bugs. We empirically investigated how requirements dependencies correlate with and predict software integration bugs, which can provide early estimates regarding software quality and thus facilitate decision making early in the software lifecycle. We conducted network analysis on the requirements dependency networks of three commercial software projects. Significant correlation is observed between most of our network measures and the number of bugs. Furthermore, many network measures demonstrate significantly greater values for higher severity (or a higher fixing workload). Afterward, we built bug prediction models using these network measures and found that bugs can be predicted with high accuracy and sensitivity, even in cross-project and cross-company contexts. We further identified the dependency type that contributes most to bug correlation, as well as the network measures that contribute more to bug prediction. These observations show that the requirements dependency network can be used as an early indicator and predictor of software integration bugs.

J. Wang (✉) · Q. Wang
Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China
e-mail: junjie@nfs.iscas.ac.cn

Q. Wang
e-mail: wq@itechs.iscas.ac.cn

J. Wang
University of Chinese Academy of Sciences, Beijing, China

Q. Wang
State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China

Keywords Requirements dependency · Bug prediction · Network analysis

1 Introduction

Most requirements cannot be treated independently, because they are related to and affect each other in complex manners [15, 26]. The existence of dependency relationships is an important characterization of software complexity. Moreover, prior work has shown that complexity harms software quality [44]. The degree of dependency reflects the cohesion and coupling between software elements [37], which are also perceived as indicators of software quality.

Dependencies between requirements can affect various decisions and activities during development, e.g., requirement change management [48], release planning [2], requirement selection and prioritization [25] and requirement reuse [43]. This influence implies that there is a need to consider requirements dependency to make sound decisions during software development.

Research has demonstrated the effects of requirements dependency on change propagation [26, 48]. In addition, change is commonly regarded as a major cause of software bugs. Furthermore, requirements dependencies can create technical dependencies during software development [32], which also influence software quality. Meanwhile, dependencies in the design and implementation phase have been proven as important predictors of software bugs [10, 53].

The above illustration motivates this study. We empirically investigated requirements dependency from the standpoint of software quality. In particular, we focus on the bugs detected during integration testing, where the communication between modules is tested for correct functioning [1]. Integration testing focuses on the connections or interactions between software components, which is established by requirements dependency. Practical experience suggests that integration bugs may be caused by the complicated interactions and connections under the effects of requirements dependencies.

Typically, studies on bug prediction utilize information in the implementation phase to conduct prediction. However, these approaches can only be applied in later phases of the software lifecycle [49, 53]. Prior work also concentrated on requirement inspection techniques to detect requirement-related errors [23, 46]. However, these techniques can only be used to detect errors originating in the requirements phase. In contrast to these techniques, we employed information present in the requirement phase to build a prediction model that can predict software bugs associated with each requirement during testing phase. These bugs can be caused by defects introduced throughout the software lifecycle, such as errors in requirements, incorrect design or implementation. This approach can provide early estimates regarding software quality and therefore facilitate decision making early in the software lifecycle.

Previous research has utilized the contents of requirements for bug prediction [19, 22], without considering the dependency relationships among requirements. Therefore, this study can provide important new insights into the potential impact exerted by requirements dependencies on software quality.

We investigate how requirements dependencies correlate with and predict software bugs with three commercial software projects from two software companies. The results show that requirements dependencies correlate with software bugs and can serve as predictors for bug prediction. Predicting potential bugs for requirements plays an important role in project management, either by highlighting the need for increased quality monitoring during development or by using the models to plan verification and validation activities. Furthermore, our work predicts the occurrence of bugs in the early phase of software development, even before coding. Therefore, our approach can act as a useful complement to existing code-based prediction models.

This article is the extended version of the conference paper published in RE 2013 [47]. Compared to the original version, in addition to the correlation between network measures and the number of bug, we also investigate the relation between network measures and the severity of bug (Sect. 4.2), as well as the fixing workload of bug (Sect. 4.3). We further replicate the studies with another software project from another software company and apply the prediction model in cross-project (Sect. 5.3) and cross-company (Sect. 5.4) contexts to further evaluate its efficacy. In addition, this article includes numerous minor updates, additional analysis and enhancements to the conference paper.

The remainder of the paper is structured as follows: We begin with the motivation of this study and introduce our two research questions in Sect. 2. After describing the detailed study design in Sect. 3, we demonstrate the identified results to answer our two research questions in Sects. 4 and 5. Subsequently, Sect. 6 identifies threats to validity and conducts cost-benefit analysis. After discussing related work in Sect. 7, we end by discussing our conclusions and future work in Sect. 8.

2 Motivation

Dependencies in the design and implementation phase have been proven to be important indicators and predictors for software bugs. Based on this insight, we wondered whether requirements dependencies can also indicate software bugs. Therefore, we traced requirements dependencies down into the code and investigated how these dependencies are structured at the code level. The following observations can be made.

Certain requirements dependencies correspond to the same method when propagating to the code level. Here is an example. Several quality plans are involved in our experimental projects, such as the review plan, test plan and audit plan. Requirements for creating the corresponding plan have the *Similar_to* relationship. When implementing these requirements, the superclass established a common interface and foundational functionality for these plans, and specialized subclasses inherited and overrode it. The existence of defects in the superclass can trigger bugs in all these requirements with this dependency.

Certain requirements dependencies involve call and data flow dependency relationships when propagating to the code level. Consider an illustrative example. The requirement “view review report” is a *Constraint* for the requirement “download work products through review report.” The corresponding methods for implementing the dependent requirements involve passing parameters (related attributes in review report) among each other, thus forming similar call and data flow relationships. Dependencies at the code level have served as significant

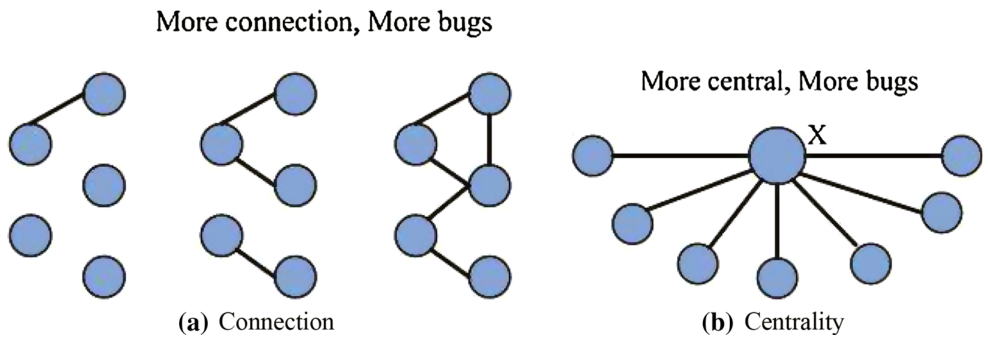


Fig. 1 Bug-prone patterns

indicators for software bugs [53], and thus, requirements dependencies are also likely to contribute to bug indication.

Because requirements dependencies would indicate software bugs, we established requirements dependency network, where nodes represent requirements and edges represent dependencies between two requirements. Based on the network, we tried to explore which requirements in the network are more bug-prone. Through analyzing the bug data and dependency networks of the experimental projects, we made the following observations.

The more connection, the more bug-prone. Because networks are defined by nodes and the connections among them, we began our investigation by examining the number of connections. We first established sub-networks with randomly chosen five requirements and their related dependencies (see Fig. 1a) and then counted the number of connections for each sub-network. Next, we computed the number of bugs for requirements for each sub-network. We observed that bugs increase with the number of connections. Put another way, requirements that are part of more complex areas (with more connections) have more bugs. This observation motivates network analysis: requirements with more connections can be treated as close to each other, which is measured by the network measure *closeness*. Connection is also reflected by other network measures such as *density* and *weak component*. We hypothesize that these network measures regarding connection would correlate with the number of bugs.

The more central, the more bug-prone. We identified several network motifs in the requirements dependency network. Network motifs are patterns that describe similar, but not necessarily isomorphic subgraphs; originally, they were introduced in biological research [34]. One of the motifs looks like a star (see Fig. 1b): It consists of a requirement X that is connected to the main component of the dependency network. Several other "satellite" requirements surround the X and depend exclusively on X. In most occurrences of the pattern, the central requirement was bug-prone, while the satellite requirements were bug-free. Network analysis identifies requirement X as *central* in the dependency network because it controls its satellite requirements. We conjecture that requirements that are identified as central by network analysis are more bug-prone than others.

In this paper, we will compute measures from network analysis on requirements dependency networks, which signify requirements' position with respect to other requirements. Based on previous observation, we focus mainly on network measures reflecting connection and centrality. We hypothesize that requirements with higher connection and higher centrality are more likely to be prone to software bugs. We aim at answering the following research questions:

1. How significantly can network measures on requirements dependency network indicate software bugs?

Before prediction, we must confirm that it is possible to use network measures on requirements dependency network to predict software bugs. More precisely, we need to find how network measures on requirements dependency network can indicate software bugs. For example, due to the interactions and connections exerted by requirements dependencies, requirements with high centrality might be more bug-prone than requirements located in surrounding areas of the network. We examine the relationship between network measures and number of bugs, as well as the severity and fixing workload of bugs.

2. Can network measures on requirements dependency network be used to predict software bugs, and what is their predictive performance?

If network measures on requirements dependency network indicate software bugs, can we use them to predict bugs and what is the predictive performance? If so, software engineers would benefit from an early estimate regarding software quality to facilitate decision making early in the software lifecycle. Furthermore, the idea behind prediction is to obtain models that can be used for prediction in future projects or other software companies. Taken in this sense, we apply the prediction model in cross-project and cross-company contexts to further evaluate its efficacy.

3 Study design

3.1 System under study

Our research questions are investigated based on three projects from two medium-sized software organizations in China. They have established stable development and maintenance processes and have achieved CMMI (Capability Maturity Model Integration) maturity level 3 or 4. We use "Project A," "Project B" and "Project C" to indicate them in the rest of this paper. Table 1 summarizes the information on their requirements and integration bugs.

Requirements are organized into a tree structure with modules, sub-modules and requirements, not distinguishing functional from non-functional requirements. Hence, our experiments did not intentionally distinguish functional requirements from non-functional requirements. We understand that most non-functional requirements are realized by functional requirements and can be defined by one or more use cases. Only a few non-functional requirements cannot be described by use case. They are usually very abstract needs with less impact on requirement dependency.

Table 1 Projects summaries

	Project A[a]	Project B[b]	Project C[c]
Brief description			
Domain	Software quality management		Web content management
Requirement summaries			
Number of requirements	308	334	75
How requirements defined	Use case (name, description, actor, event flow, precondition and post-condition)		Text description
Bug summaries			
Number of integration bugs	732	418	286
How bugs stored	Bugzilla	Bugzilla	Jira
Related bug attributes	Title, description, severity, fixing workload and sub-module		Title, description and sub-module
Bug attributes—severity			
Critical	327	188	n/a
Serious	189	137	n/a
Normal	216	93	n/a
Bug attributes—fixing workload			
High	227	142	n/a
Medium	251	126	n/a
Low	254	150	n/a

[a] http://qone.nfschina.com/qone

[b] http://qone.nfschina.com/qone

[c] http://www.trs.com.cn/en/pro/wcm/index.html

Note that Project A and Project B are two different versions of the same software product, so they share certain attributes. However, they were developed by different teams, in different geographical locations and during different periods of time. Project A is a customized version, while Project B is a third upgraded version after Project A. The overlap of requirements between these two versions is less than 30 %. Therefore, we assume that Project A and Project B can be treated as independent datapoints in the results.

When design, coding and unit testing are completed, individual program units are integrated and tested. Integration bugs are recorded and kept in project repositories. Our experiments involve two bug attributes: severity and fixing workload. There are three levels of severity, namely Critical, Serious and Normal from high to low. The bug submitter supplies the original severity information. Afterward, the test manager verifies the submission and can revise the value. Fixing workload is the time spent on resolving the bug, provided by a bug-fixer. As the fixing workload is a continuous value, we first discretized it into three levels from high to low, namely High, Medium and Low. The severity and fixing workload data are incomplete for Project C, so experiments on bug severity (Sect. 4.2) and bug fixing workload (Sect. 4.3) are conducted based only on Project A and Project B. We have examined Spearman correlation between these two attributes, and the results revealed no correlation between them (0.122 ($p = 0.215$) for Project A; 0.317 ($p = 0.190$) for Project B). Table 1 also displays the distribution of bugs among different attributes.

3.2 Requirements dependency identification

To construct the requirements dependency network, identifying requirements dependencies is the top priority. In contrast to the commonly accepted call and data flow relationship at the code level, many types of requirements dependency have been proposed over the years, and most of them have different levels of abstraction and different criteria for categorization.

We focus on three types of requirements dependency relationships, namely *Precondition*, *Similar_to* and *Constraint*, for network construction.

Precondition: Only after one function is finished or one condition is satisfied can another function be performed. Usually this relationship reflects the business rule or the sequential relationship between (sub-)processes, e.g., the requirement "copy task" is the precondition for "paste task."

Similar_to: The description of one requirement (e.g., "print task report") is similar to or overlaps with another requirement (e.g., "print progress report").

Constraint: A requirement can relate to or constrain another requirement. For instance, the requirement "option to print task report" is a constraint for the requirement "print task report."

These dependencies are chosen because of their relevance to change propagation [26]. Change is commonly regarded as the major cause of software bugs. Moreover, metrics quantifying change impact have been proven to be significant predictors for bug proneness [24]. For a detailed description of these three dependency types, refer to [26].

To establish a solid foundation for our experiment, we utilized manual identification of requirements dependencies. Four or three practitioners were involved in the network construction processes for the three experimental projects. The details are summarized in Table 2.

First, every participant individually learned relevant information about these dependencies. Second, we randomly chose 1/4 requirements (77, 84 and 19, respectively, for Project A, B and C) and asked the participants to identify specific types of dependencies for these requirements. We then analyzed the identified dependencies across the participants, and follow-up interviews were conducted to elicit the in-depth reasons behind the differences and the similarities until a consensus was reached. Third, the practitioners were required to conduct identification for all remaining requirements. After their separate associations, we still analyzed the differences and similarities among their associations and conducted follow-up interviews. A common consensus was reached on all issues, and a final edition of the dependencies was used for further analysis. Table 3 lists the descriptive statistics of the identified requirements dependencies. Table 24 presents the time taken for the identification and interview. We also summarize the details and lessons learned during the identification process, which can serve as notes and guidelines for practical use.

Table 2 Practitioners summaries of network construction

	Project A and Project B	Project C
1.	Project manager and architect	Project manager
	(8-year experience in project management and architecture)	(5-year experience in project management)
2.	Requirements analysis engineer	Developer
	(8-year experience in requirement analysis and management)	(4-year experience in software development)
3.	Developer	Developer
	(3-year experience in software development)	(2-year experience in software development)
4.	Developer	
	(2-year experience in software development)	

Table 3 Identified requirements dependencies

	Precondition	Similar_to	Constraint	Total
Project A	43	55	83	161
Project B	25	49	81	155
Project C	0	20	48	68

1. For the first 1/4 requirements, the similarity rates[1] between any pair of practitioners were between 0.74 and 0.86 for all projects. However, the similarity rates for the identification of the remaining requirements increased to above 0.90 (from 0.90 to 0.98 for all projects).

We interviewed the participants and several of them noted that they initially misunderstood the definition of certain types of requirements dependencies. After the follow-up interviews and discussion, the incorrect dependencies were revealed, and the revised version of the dependencies served as examples and standards for the identification of the remaining requirements, thus improving the accuracy. This indicated that certain concrete examples or pilot identification is effective for improving accuracy.

2. The participants' background and viewpoint could influence their identification. Requirements analysis engineers paid more attention to the text representation of the requirements, so they were good at finding structure and content dependencies, e.g., Similar_to. The developers discovered dependencies from a development point of view and cared more about how to implement the requirements. Thus, they did well in finding Constraint and Precondition relationships. Project managers, who possess the knowledge of both requirements management and development, can identify the dependencies from both business and implementation viewpoints. Therefore, their identification accuracy was the highest. This result indicated that people with rich experience and diverse backgrounds can perform better in identification.

3.3 Network construction

In our requirements dependency network, nodes represent requirements, while edges represent dependencies between two requirements. We did not distinguish the direction of edges, so our network is a symmetric, undirected network.

[1] Similarity rate between practitioner P1 and P2 is defined as $\frac{\text{dependencies identified by } P1 \cap \text{dependencies identified by } P2}{\text{dependencies identified by } P1 \cup \text{dependencies identified by } P2}$.

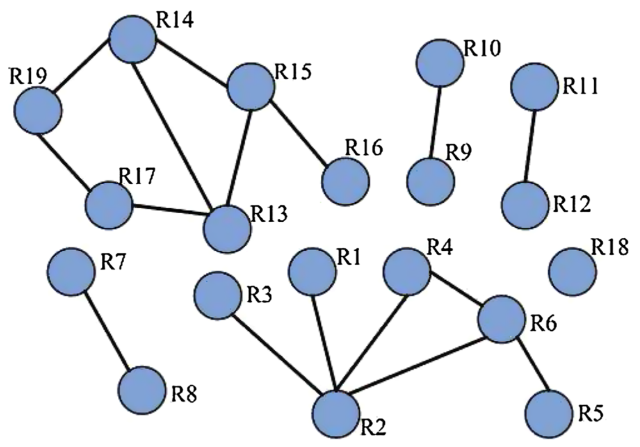


Fig. 2 Example of part of requirements dependency network

Table 4 Corresponding requirements to Fig. 2

Id	Name
R1	Add task
R2	Maintain task information
R3	View task detailed information
R4	Assign task
R5	Delete task
R6	Save task
R7	Copy task
R8	Paste task
R9	Single critical path
R10	Multiple critical path
R11	Select task
R12	Display selected task
R13	Fill in/Modify task report
R14	Fill in/Modify individual report
R15	Submit work product
R16	Maintain work product details
R17	View detailed task report
R18	Select task report by date
R19	View individual report

We emphasize that the network in this paper refers to all the requirements of a project and related dependencies. In other words, our network might contain isolated nodes (details are shown in Table 5). As our prediction would produce results for every requirement, we utilize the whole network (including isolated nodes) to compute network measures and conduct correlation and prediction analysis. Figure 2 displays an example of what a small part of requirements dependency network looks like, with Table 4 showing the corresponding requirement information.

Based on the identified requirements dependencies, we constructed three types of requirements dependency network: non-HybridNetwork, semi-HybridNetwork and HybridNetwork. Non-HybridNetwork is the requirements dependency network where the edges represent only one type of dependency. Corresponding to the three types of requirements dependencies, there are three types of non-HybridNetwork, respectively, PreNetwork, SimNetwork and ConNetwork. Semi-HybridNetwork is the requirements dependency network where edges represent two types of dependencies. There are also three types of semi-HybridNetwork: PreSimNetwork, PreConNetwork and SimConNetwork, corresponding to the three types of dependencies. Similarly, HybridNetwork is the requirements dependency network where the edges represent three types of dependencies.

As our requirements are fine-grained, we did not observe the circumstance in which two or more dependencies suggest an edge. Table 5 lists each types of dependency network in terms of the number of edges, number of nodes and number of isolated nodes for each network. Because only the Constraint and Similar_to dependency relationships are identified for Project C (see Table 3), there are only two types of non-HybridNetwork and one type of semi-HybridNetwork, which is also HybridNetwork, for Project C.

We constructed non-HybridNetwork and semi-HybridNetwork to examine the relative indication effect exerted by different types of requirements dependency (Sect. 4.1.1). If not specially noted, network and network measures refer to HybridNetwork.

3.4 Bug association

As described in Sect. 3.1, the bug tracking system only stored information about which sub-function module a bug belongs to. To investigate our research questions, we needed to establish the association between bugs and requirements, i.e., which requirement a bug is associated with.

Consider an example to illustrate the association. There is a bug in the bug tracking system described as follows: *On task report viewing page, when viewing the manual-input task report, column names were wrongly displayed*. It was recorded in the sub-function module "Task report." Aided by such information as "task report viewing page" embedded in the description, one can determine that this bug should be associated with the requirement "View detailed task report."

To guarantee completeness and accuracy, this association process involved three practitioners for the three experimental projects. The details are summarized in Table 6.

Table 5 Network construction summaries

		Project A	Project B	Project C
Non-HybridNetwork	PreNetwork	43, 308/164	25, 334/240	n/a
	SimNetwork	55, 308/173	49, 334/191	20, 75/48
	ConNetwork	83, 308/131	81, 334/166	48, 75/25
Semi-HybridNetwork	PreSimNetwork	98, 308/134	74, 334/160	n/a
	PreConNetwork	126, 308/92	106, 334/139	n/a
	SimConNetwork	138, 308/94	130, 334/108	68, 75/13
HybridNetwork		161, 308/69	155, 334/88	68, 75/13

The value denotes *edges, nodes/isolated nodes* of the dependency network

Table 6 Practitioners summaries of bug association

	Project A and Project B	Project C
1	Test manager	Tester
	(5-year experience in software testing)	(3-year experience in software testing)
2	Tester	Tester
	(5-year experience in software testing)	(2-year experience in software testing)
3	Developer	Developer
	(3-year experience in software development)	(2-year experience in software development)

Table 7 Bug association summaries

	Project A	Project B	Project C
Number of bug per req.			
Max	26	9	14
Average	2.37	1.25	3.81
Number of bug with isolated req.	19	10	23
Number of req. without bug	66 (21.4 %)	98 (29.3 %)	4 (5.3 %)
Number of isolated req.	69 (22.4 %)	88 (26.3 %)	13 (17.3 %)
Number of isolated req. without bug	55 (17.8 %)	81 (24.2 %)	2 (2.6 %)

The association process was broadly divided into three steps. Firstly, 50 randomly chosen bugs were distributed to the three practitioners. After determining their separate associations, we analyzed the similarity and differences among their associations. The similarity rates (see Sect. 3.2) between any pair of practitioners were, respectively, 0.75, 0.77 and 0.81 for Project A.[2] We then conducted follow-up interviews to elicit in-depth reasons behind the differences and the similarity until a common consensus was reached.

Second, we randomly chose another 50 bugs and repeated the first step. The pairwise similarity rates were then 0.92, 0.90 and 0.94 for Project A.[3] Follow-up interviews were also conducted, as in the first step.

Third, the practitioners were required to conduct the association for all remaining bugs. After their separate associations, we again analyzed the differences and similarity among their associations and conducted follow-up interviews until a common consensus was reached.

After these three steps, all the integration bugs were associated with requirements. We summarize the bug association results in Table 7. To provide a concrete description, Table 8 lists the number of bugs and bug descriptions after the association process for the requirements of Fig. 2 and Table 4.

3.5 Network measures

For each requirements dependency network, we computed a number of network measures using the Ucinet tool [8]. In this section, we describe these measures in detail. For a more comprehensive overview, refer to textbooks on network analysis, e.g., [21].

Within the context of network analysis, each node belongs to two types of network: an ego network and a global network. The ego network of a node consists of the node itself and every other node that depends on it or on which it depends. The global network corresponds to all nodes in the network. While an ego network allows us to measure the local importance of a node with respect to its neighbors, a global network reveals the importance of a node within the entire network. As we expect local and global importance to complement each other, we used both in our study.

Table 9 describes the ego network measures. For each node, we computed each measure for the ego network. For the global network, we computed measures of centrality and structural holes as shown in Table 10. Section 2 mentions that we mainly investigate network measures related to connection and centrality. Therefore, we used the

[2] The similarity rates of first step for Project B were, respectively, 0.80, 0.80 and 0.82, while similarity rates for Project C were, respectively, 0.76, 0.78 and 0.78.

[3] The similarity rates of second step for Project B were, respectively, 0.94, 0.94 and 0.96, while similarity rates for Project C were, respectively, 0.86, 0.88 and 0.88.

Table 8 Bug information corresponding to the requirements in Fig. 2 and Table

Req.	Bug number	Examples of detailed bug description
R1	4	When adding sub-task to certain task, the default task leader is missing
R2	9	When maintaining task information, these assigned tasks can still be split
		When setting the restriction relationship, the begin time of successor task can be earlier than the end time of precursor task
R3	2	For these ongoing task, when viewing task information, the state of the task is wrong
R4	2	When assigning task, in the pull-down menu of project members, certain project members are missing
R5	1	These ongoing tasks can be deleted
R6	6	After setting the restriction relationship, when saving tasks, there is a reminder 'time restriction violation,' which is actually not the case
		After assigning two tasks, when saving, only the later task is saved
R7	1	When copying task, the sub-tasks of certain task are not copied
R8	2	For these tasks which have such information as task progress and actual workload, pasting as sub-task can still be done
R9	0	
R10	1	When calculating multiple critical path, if the task belongs to several paths, the end time at the latest is wrong
R11	1	When selecting task, the sub-task is not selected
R12	2	When displaying selected task, the task leader of sub-tasks is wrong
R13	7	For these unscheduled tasks, when modifying task report, the time for the task is wrong
		For these finished tasks, when modifying task report, the modification can still be done
R14	6	For these unscheduled tasks, when modifying individual report, the time for the task is wrong
		For these people belonging to multiple projects, when filling in individual report, some project names are wrong
R15	2	When submitting work product, photographs can not be loaded
R16	3	When maintaining work product details after deleting work product, the deleted work product is still displayed
R17	2	For these unscheduled tasks, when viewing detailed task report, the time for the task is missing
R18	0	
R19	1	When project manager views the individual reports of project members, for these people belonging to multiple projects, the individual reports of other projects are still displayed

Table 9 Ego network measures

Measure	Description	Category
Size	The number of nodes in the ego network	Connection
Ties	The number of edges in the ego network	
Pairs	The maximal number of ties, i.e., Size * (Size-1)	
Density	The percentage of possible ties that are actually present, i.e., Ties/Pairs	
AvgDist	Average geodesic distance among reachable pairs	Other
WeakComp	The number of weak components (sets of connected nodes) in neighborhood	Connection
nWeakComp	The WeakComp normalized by size of the ego network, i.e., WeakComp/Size	
TwoStepReach	The percentage of nodes that are two steps away	
ReachEfficiency	The TwoStepReach normalized by size of the ego network, i.e., TwoStepReach/Size	
	High ReachEfficiency indicates that ego's primary contacts are influential in the network	
Brokerage	The number of pairs not directly connected	Centrality
	The higher Brokerage, the more paths go through ego, i.e., ego acts as a "broker" in its network	
nBrokerage	The Brokerage normalized by the number of pairs, i.e., Brokerage/Pairs	
EgoBetweenness	The percentage of shortest paths between neighbors that pass through ego	
nEgoBetweenness	The EgoBetweenness normalized by size of the ego network, i.e., EgoBetweenness/Size	

category column to indicate whether each network measure belonged to *Connection*, *Centrality* or *Other*. Note that, the classification into Connection and Centrality is solely for this research, and certain network measures, such as Closeness as mentioned in Sect. 2, can relate to both Connection and Centrality. In this case, we give priority to Centrality, as it is widely researched.

4 Correlation between network measures and software bugs

To answer the research question Q1, "How significantly can network measures on requirements dependency

Table 10 Global network measures

Measure	Description	Category
Centrality		
Degree	The number of nodes that are directly connected to a node	Centrality
Closeness	The total length of all shortest paths from a node to all other nodes	
Reachability	The weighted number of nodes that can be reached from a node. The weigh is 1, 1/2, 1/3 for nodes that are 1, 2 or 3 steps away	
Betweenness	The number of shortest paths between other nodes that a node occurs	
Structural holes		
EffSize	The number of nodes that are connected to a node minus the average number of ties between these nodes	Connection
Efficiency	The EffiSize normalized by size of the network	
Constraint	The extent to which the node is limited in option to reach other nodes	Other
Hierarchy	Concentration of constraint in the global network	

network indicate software bugs?", we examine the relationships between network measures and the number of bugs, as well as the severity and fixing workload of bugs. For the bug number, we conducted correlation analysis to test whether there were significant correlations between network measures for each requirement and its number of bugs. We also identified which type of requirements dependency network works best for the correlation, as well as which type of requirements dependencies contribute more to the correlation. For severity and fixing workload, we used hypothesis testing to examine whether there are significant differences in network measures among different severities (or fixing workloads).

4.1 Correlation between network measures and number of bugs

We determined the Spearman rank correlation between the network measures for each requirement (Sect. 3.5) and the corresponding number of bugs. We selected Spearman over Pearson correlation because Spearman is a more robust technique that can be applied even when the association between the measures is nonlinear [18]. Spearman obtains correlation values range between -1 and $+1$. The closer the value of correlation is to -1 or $+1$, the more strongly the two measures are correlated, positively for $+1$ and negatively for -1. A value of 0 indicates that the two measures are independent.

In statistics, it is generally considered that a value lower than -0.5 or larger than $+0.5$ denotes significant correlation between two measures, and a value lower than -0.7 or larger than $+0.7$ denotes strong correlation [20].

For each project, we display the Spearman correlation values between network measures and bug number for non-HybridNetwork, semi-HybridNetwork and HybridNetwork. The correlation values for Project A are shown in Table 11 (non-HybridNetwork) and Table 12 (semi-HybridNetwork and HybridNetwork). Similarly, the correlation values for Project B are displayed in Tables 13 and 14. Table 15 shows the correlation values for Project C. We have mentioned that there are only two types of non-HybridNetwork, and one type of semi-HybridNetwork (which is also HybridNetwork) for Project C (see Sect. 3.3 and Table 5).

Tables 11, 12, 13, 14 and 15 consist of two main parts: ego network measures and global network measures. Global network measures are grouped into centrality measures and structural holes measures. The columns distinguish different types of dependency network. The values "n/a" for Project C signify that the Spearman correlations for these network measures are incomputable.

4.1.1 *Which type of requirements dependency network works best?*

We compare the correlation values of different types of requirements dependency network and make the following observations.

1. Generally speaking, network measures on HybridNetwork demonstrate the highest correlation with bug number, while the correlations of semi-HybridNetwork are higher than correlations of non-HybridNetwork. We can infer that every type of requirements dependency contributes to the correlation. Put another way, each of the three types of requirements dependencies contributes to bug indication and prediction.

However, we could not assert that we have studied the complete set of requirements dependencies related to bug prediction. Nevertheless, only these three dependencies are identified as relevant to change propagation [26]. Taken in this sense, our experimental setup is reasonable. However, the influence of other types of requirements dependencies on bug prediction needs further investigation.

2. Narrowing our focus to non-HybridNetwork, we could observe that network measures on ConNetwork demonstrate the highest correlation with bug number. The difference between PreNetwork and SimNetwork is less obvious, with SimNetwork slightly outperforming PreNetwork. This result suggests that the *Constraint* type of requirement dependency may contribute more to bug indication.

Table 11 Spearman correlation between network measures of non-HybridNetwork and number of bug for Project A

	PreNet	SimNet	ConNet
Ego network			
^{n} Size	0.252	0.276	**0.616**
^{n} Ties	0.221	0.242	**0.537**
^{n} Pairs	0.226	0.211	0.462
^{n} Density	0.189	0.189	0.437
^{o} AvgDist	0.202	0.201	0.384
^{n} WeakComp	0.269	0.310	**0.600**
^{n} nWeakComp	0.245	0.490	0.429
^{n} TwoStepReach	0.266	0.275	**0.570**
^{n} ReachEfficiency	0.203	0.238	0.431
^{t} Brokerage	0.203	0.251	0.448
^{t} nBrokerage	0.219	0.244	0.411
^{t} EgoBetweenness	0.200	0.229	0.458
^{t} nEgoBetweenness	0.192	0.210	0.410
Global network			
Centrality measures			
^{t} Degree	0.252	0.276	**0.616**
^{t} Closeness	0.254	0.393	0.448
^{t} Reachability	0.281	0.318	**0.623**
^{t} Betweenness	0.185	0.310	0.462
Structural holes measures			
^{n} EffSize	0.236	0.230	**0.518**
^{n} Efficiency	0.217	0.220	0.454
^{o} Constraint	0.163	0.189	0.427
^{o} Hierarchy	−0.183	−0.196	−0.263

All correlations are significant at the 0.01 level $p < 0.01$. Correlations above 0.50 are printed in boldface

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category

As illustrated in Sect. 2, for *Constraint* dependency, the majority correspond to the same method when propagating to code. Most of the *Similar_to* dependencies correspond to the same method when traced down to code level. For the *Precondition* type, both patterns are observed. We assumed that the difference in correlation values for the three types of non-HybridNetwork can be attributed to different behaviors when the requirements dependencies propagate to code level. However, the evidence needs further exploration.

3. For semi-HybridNetwork, not surprisingly, the correlation values demonstrate the combined effect of the single types of requirements dependencies. The network measures on PreSimNetwork show the lowest correlation with bug number, while PreConNetwork and SimConNetwork display higher correlation values, with SimConNetwork being the highest for most network measures.

Table 12 Spearman correlation between network measures of semi-HybridNetwork (and HybridNetwork) and number of bug for Project A

	PreSimNet	PreConNet	SimConNet	HybridNet
Ego network				
^{n} Size	0.437	**0.735**	**0.805**	**0.831**
^{n} Ties	0.268	0.493	**0.511**	**0.527**
^{n} Pairs	0.401	**0.675**	**0.652**	**0.673**
^{n} Density	0.306	**0.556**	**0.571**	**0.650**
^{o} AvgDist	0.206	0.457	0.463	0.478
^{n} WeakComp	0.424	**0.680**	**0.714**	**0.511**
^{n} nWeakComp	0.354	0.462	0.485	**0.785**
^{n} TwoStepReach	0.434	**0.740**	**0.602**	**0.646**
^{n} ReachEfficiency	0.413	**0.625**	**0.602**	**0.646**
^{t} Brokerage	0.395	0.443	0.482	**0.518**
^{t} nBrokerage	0.453	**0.579**	**0.603**	**0.602**
^{t} EgoBetweenness	0.394	0.470	0.489	**0.509**
^{t} nEgoBetweenness	0.452	**0.602**	**0.627**	**0.607**
Global network				
Centrality measures				
^{t} Degree	0.437	**0.735**	**0.805**	**0.831**
^{t} Closeness	0.366	0.396	**0.584**	**0.621**
^{t} Reachability	0.437	**0.677**	**0.716**	**0.769**
^{t} Betweenness	0.392	0.445	0.481	**0.508**
Structural holes measures				
^{n} EffSize	0.235	**0.580**	**0.541**	**0.783**
^{n} Efficiency	0.027	−0.313	−0.416	**0.514**
^{o} Constraint	−0.350	−0.539	−0.457	0.438
^{o} Hierarchy	−0.095	−0.398	−0.456	−0.455

All correlations are significant at the 0.01 level $p < 0.01$. Correlations above 0.50 are printed in boldface

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category

4.1.2 Further correlation analysis for HybridNetwork

As the network measures of HybridNetwork demonstrate the highest correlation with bug number, we focus on HybridNetwork and further explore its correlation results. The following observations can be made.

1. Indicated by the high correlation values of HybridNetwork, there are significant correlations between most of the network measures and the number of bugs. Narrowing our focus to the network measures with strong correlation (strong for at least two projects), they can be mapped to the two categories we mentioned: Connection (i.e., Size, WeakComp, TwoStepReach and EffSize) and Centrality (i.e., Degree and Reachability). This result further verifies the observations in Sect. 2. Furthermore, it is consistent with observations in call and data flow dependency networks [53]. It indicates that requirements with higher

Table 13 Spearman correlation between network measures of non-HybridNetwork and number of bug for Project B

	PreNet	SimNet	ConNet
Ego network			
^{n} Size	0.151	0.360	**0.558**
^{n} Ties	0.149	0.351	**0.544**
^{n} Pairs	0.193	0.335	0.432
^{n} Density	0.188	0.287	0.329
^{o} AvgDist	0.175	0.253	0.351
^{n} WeakComp	0.221	0.342	**0.547**
^{n} nWeakComp	0.217	0.327	0.362
^{n} TwoStepReach	0.204	0.358	**0.613**
^{n} ReachEfficiency	0.206	0.343	0.438
^{t} Brokerage	0.165	0.297	0.462
^{t} nBrokerage	0.162	0.267	0.415
^{t} EgoBetweenness	0.137	0.198	0.304
^{t} nEgoBetweenness	0.380	0.178	0.299
Global network			
Centrality measures			
^{t} Degree	0.151	0.360	**0.558**
^{t} Closeness	0.158	0.290	0.397
^{t} Reachability	0.231	0.358	**0.551**
^{t} Betweenness	0.133	0.201	0.302
Structural holes measures			
^{n} EffSize	0.179	0.324	0.456
^{n} Efficiency	0.185	0.307	0.430
^{o} Constraint	0.124	0.200	0.241
^{o} Hierarchy	−0.130	−0.203	−0.190

All correlations are significant at the 0.01 level $p < 0.01$. Correlations above 0.50 are printed in boldface

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category

Table 14 Spearman correlation between network measures of semi-HybridNetwork (and HybridNetwork) and number of bug for Project B

	PreSimNet	PreConNet	SimConNet	HybridNet
Ego network				
^{n} Size	0.468	**0.637**	**0.755**	**0.766**
^{n} Ties	0.235	0.434	0.477	**0.500**
^{n} Pairs	0.364	**0.602**	**0.652**	**0.696**
^{n} Density	0.296	**0.606**	**0.652**	0.365
^{o} AvgDist	0.238	0.417	0.429	0.449
^{n} WeakComp	0.452	**0.714**	**0.680**	**0.728**
^{n} nWeakComp	0.410	0.429	0.475	0.400
^{n} TwoStepReach	0.449	**0.746**	**0.740**	**0.791**
^{n} ReachEfficiency	0.404	**0.670**	**0.602**	**0.536**
^{t} Brokerage	0.297	0.331	0.458	**0.630**
^{t} nBrokerage	0.395	0.400	**0.554**	**0.650**
^{t} EgoBetweenness	0.296	0.331	0.457	**0.618**
^{t} nEgoBetweenness	0.395	0.400	**0.554**	**0.649**
Global network				
Centrality measures				
^{t} Degree	0.468	**0.637**	**0.755**	**0.766**
^{t} Closeness	0.222	0.343	**0.538**	**0.613**
^{t} Reachability	0.443	**0.616**	**0.753**	**0.794**
^{t} Betweenness	0.295	0.332	0.455	**0.630**
Structural holes measures				
^{n} EffSize	0.296	0.330	0.457	**0.759**
^{n} Efficiency	−0.287	−0.406	−0.424	0.455
^{o} Constraint	−0.338	−0.241	−0.451	0.174
^{o} Hierarchy	−0.362	−0.427	−0.494	−0.300

All correlations are significant at the 0.01 level $p < 0.01$. Correlations above 0.50 are printed in boldface

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category

connection and higher centrality are more prone to software bugs.

2. The centrality measures Degree and Reachability exhibit a positive and strong correlation with bugs, while the centrality measures Closeness and Betweenness exhibit a positive and significant correlation. This result means that central requirements are more likely to be bug-prone than requirements located in the surrounding areas of the network. Similar evidence has been found in call and data flow dependency networks [53] and developer-module networks [41], suggesting that network centrality measures are significant indicators for bug proneness.
3. We could not observe a noteworthy difference in correlation values between ego network measures and global network measures. That is, not only the global insights, but also the local information, contribute to bug indication.
4. Not all the network measures have positive correlations with the number of bugs. Such structural holes measures as Efficiency, Constraints and Hierarchy show an obvious negative correlation for Project C (for Project A and B, Hierarchy shows a negative correlation). This result means that an increase in structural holes usually corresponds to a decrease in the number of bugs. However, structural holes measures such as Constraints and Hierarchy fall outside the Connection and Centrality categories and appear less intuitively understandable, and moreover, the correlations are less significant, so we put these measures aside for in-depth discussion.
5. Correlations of network measures demonstrate minor inconsistency among different companies and projects. For example, nWeakComp and ReachEfficiency show correlations significant at 99 % for Project A and Project B but not significant for Project C. As these two measures are obtained through normalization by the size of the ego network (see Sect. 3.5), the

Table 15 Spearman correlation between network measures of non-HybridNetwork (and semi-HybridNetwork, HybridNetwork) and number of bug for Project C

	SimNet	ConNet	SimConNet (HybridNet)
Ego network			
^{n} Size	0.331**	**0.615****	**0.764****
^{n} Ties	n/a	0.321**	**0.527****
^{n} Pairs	0.479**	**0.521****	**0.770****
^{n} Density	n/a	0.370**	0.469**
^{o} AvgDist	n/a	−0.077	0.385**
^{n} WeakComp	0.331**	**0.636****	**0.720****
^{n} nWeakComp	0.258**	0.394**	0.048
^{n} TwoStepReach	0.193	**0.627****	**0.682****
^{n} ReachEfficiency	0.124	0.423**	0.040
^{t} Brokerage	0.479**	**0.566****	**0.710****
^{t} nBrokerage	**0.554****	**0.678****	**0.578****
^{t} EgoBetweenness	0.479**	**0.566****	**0.710****
^{t} nEgoBetweenness	**0.544****	**0.674****	**0.578****
Global network			
Centrality measures			
^{t} Degree	0.331**	**0.615****	**0.764****
^{t} Closeness	−0.235	**0.588****	**0.547****
^{t} Reachability	0.227	**0.641****	**0.647****
^{t} Betweenness	0.477**	0.028	**0.666****
Structural holes measures			
^{n} EffSize	0.479**	**0.566****	**0.667****
^{n} Efficiency	n/a	−0.272	−0.406**
^{o} Constraint	−0.696**	−0.585**	−0.488**
^{o} Hierarchy	−0.686**	−0.352**	−0.452**

Correlations significant at 99 % are marked by **. Correlations above 0.50 are printed in boldface

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category

Table 16 Significance level of one-tailed Mann–Whitney *U* test for bug severity of Project A

	Critical versus serious	Serious versus normal	Critical versus normal
Ego network			
^{n} Size	0.009**	0.000**	0.000**
^{n} Ties	0.046*	0.000**	0.000**
^{n} Pairs	0.008**	0.000**	0.000**
^{n} Density	0.220	0.285	0.036*
^{o} AvgDist	0.573	0.096	0.000**
^{n} WeakComp	0.062	0.000**	0.000**
^{n} nWeakComp	0.861	0.791	0.896
^{n} TwoStepReach	0.063	0.000**	0.000**
^{n} ReachEfficiency	0.003**	0.000**	0.040*
^{t} Brokerage	0.002**	0.000**	0.002**
^{t} nBrokerage	0.434	0.478	0.215
^{t} EgoBetweenness	0.005**	0.000**	0.005**
^{t} nEgoBetweenness	0.875	0.320	0.234
Global network			
Centrality measures			
^{t} Degree	0.009**	0.000**	0.000**
^{t} Closeness	0.684	0.000**	0.000**
^{t} Reachability	0.067	0.000**	0.000**
^{t} Betweenness	0.371	0.000**	0.000**
Structural holes measures			
^{n} EffSize	0.003**	0.000**	0.004**
^{n} Efficiency	0.531	0.025*	0.002**
^{o} Constraint	0.016*	0.011*	0.253
^{o} Hierarchy	0.083	0.407	0.668

Significance level at 0.01 are marked by **, and significance level at 0.05 are marked by *

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category

correlation might be influenced by network structure and connectivity. However, the evidence needs further exploration.

4.2 Relation between network measures and severity of bugs

To investigate the relationship between network measures and bug severity, we design our experiments to check whether network measures of the corresponding requirements are significantly greater for higher severity bugs compared to lower severity bugs.

We have mentioned that there are three levels of severity, namely Critical, Serious and Normal from high to low. Therefore, we created three groups, the Critical group, the Serious group and the Normal group. Each group contains the network measures of requirements whose associated bugs have the corresponding level of severity.

We compare the distributions of the network measures pairwise for these three groups using the one-tailed Mann–Whitney *U* test. Our motivation to examine the groups pairwise is to allow the observation of differences between each pair of groups. For each network measure, we run three tests for these three groups with the following hypothesis:

H1: The network measures of the requirements for the higher severity group are significantly greater than for the lower severity group.

We have mentioned that the severity attributes of Project C are incomplete (Sect. 3.1), so we only present the statistical results for Project A and Project B, respectively, in Tables 16 and 17. We can make the following observations.

Table 17 Significance level of one-tailed Mann–Whitney *U* test for bug severity of Project B

	Critical versus serious	Serious versus normal	Critical versus normal
Ego network			
^{n} Size	0.763	0.008**	0.002**
^{n} Ties	0.301	0.002**	0.000**
^{n} Pairs	0.759	0.011*	0.002**
^{n} Density	0.096	0.059	0.012*
^{o} AvgDist	0.062	0.099	0.000**
^{n} WeakComp	0.015*	0.058	0.006**
^{n} nWeakComp	0.835	0.008**	0.004**
^{n} TwoStepReach	0.232	0.159	0.071
^{n} ReachEfficiency	0.223	0.060	0.159
^{t} Brokerage	0.166	0.368	0.199
^{t} nBrokerage	0.188	0.063	0.191
^{t} EgoBetweenness	0.165	0.367	0.194
^{t} nEgoBetweenness	0.165	0.368	0.199
Global network			
Centrality measures			
^{t} Degree	0.763	0.008**	0.002**
^{t} Closeness	0.233	0.102	0.022*
^{t} Reachability	0.256	0.011*	0.002**
^{t} Betweenness	0.930	0.162	0.170
Structural holes measures			
^{n} EffSize	0.257	0.110	0.251
^{n} Efficiency	0.000**	0.916	0.129
^{o} Constraint	0.464	0.299	0.441
^{o} Hierarchy	0.159	0.087	0.005**

Significance level at 0.01 are marked by **, and significance level at 0.05 are marked by *

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category

Table 18 Significance level of one-tailed Mann–Whitney *U* test for bug fixing workload of Project A

	High versus medium	Medium versus low	High versus low
Ego network			
^{n} Size	0.031*	0.570	0.048*
^{n} Ties	0.473	0.642	0.560
^{n} Pairs	0.032*	0.522	0.060
^{n} Density	0.119	0.437	0.032*
^{o} AvgDist	0.072	0.098	0.001**
^{n} WeakComp	0.007**	0.161	0.037*
^{n} nWeakComp	0.476	0.198	0.196
^{n} TwoStepReach	0.000**	0.319	0.000**
^{n} ReachEfficiency	0.178	0.112	0.021*
^{t} Brokerage	0.000**	0.511	0.000**
^{t} nBrokerage	0.011*	0.098	0.000**
^{t} EgoBetweenness	0.000**	0.572	0.000**
^{t} nEgoBetweenness	0.017*	0.073	0.000**
Global network			
Centrality measures			
^{t} Degree	0.031*	0.570	0.048*
^{t} Closeness	0.000**	0.587	0.000**
^{t} Reachability	0.000**	0.696	0.000**
^{t} Betweenness	0.000**	0.533	0.000**
Structural holes measures			
^{n} EffSize	0.000**	0.637	0.000**
^{n} Efficiency	0.043*	0.093	0.001**
^{o} Constraint	0.151	0.649	0.206
^{o} Hierarchy	0.054	0.806	0.019*

Significance level at 0.01 are marked by **, and significance level at 0.05 are marked by *

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category

1. As indicated by the significance level, many of the network measures for the higher severity groups are significantly greater than in lower severity groups. Generally speaking, more network measures demonstrate significant differences between the Critical group and Normal group, compared to critical versus serious or serious versus normal. This result may be because of the larger difference in severity values between the Critical group and Normal group.
2. Narrowing our focus to the network measures with significant differences (significant for at least two tests), we can map them to the two categories: Connection (i.e., Size, Ties, Pairs and WeakComp) and Centrality (i.e., Degree and Reachability). Although the detailed network measures appear to exhibit tiny differences, these two categories are consistent with the observations regarding the number of bugs in Sect. 4.1. This result indicates that the higher the connection and the higher the centrality of the requirements, the more likely it is for high severity bugs to occur.
3. The significance levels of network measures exhibit minor inconsistency among different projects. More network measures exhibit differences among different severity levels for Project A than for Project B. Moreover, such network measures as TwoStepReach, ReachEfficiency and EffSize (related to Connection) and Brokerage, EgoBetweenness, Closeness and Betweenness (related to Centrality) are significant only for Project A. This result might be because the number of bugs for Project A is greater (see Table 1), and more network measures have shown significant correlation with the number of bugs for Project A (see Table 12). However, we leave the evidence for further exploration.

4.3 Relation between network measures and fixing workload of bugs

To fix a bug requires a certain amount of work, and different bugs generate different fixing workloads. Our experiments involve checking whether network measures of the corresponding requirements are significantly greater for higher fixing workloads compared to lower workloads.

We have mentioned that we discretized the fixing workload into three levels: High, Medium and Low. Then, we created three experimental groups, respectively, the High group, the Medium group and the Low group. Each group contains the network measures of requirements whose associated bugs have the corresponding level of fixing workload.

As in the previous section, we compared the distributions of the network measures pairwise for these three groups using one-tailed Mann–Whitney *U* test. For each network measure, we ran three tests for these three groups with the following hypothesis:

H1: The network measures of requirements for the higher workload group are significantly greater than for the lower workload group.

We have mentioned that the fixing workload data for Project C are incomplete (Sect. 3.1), so we only present the statistical results for Project A and Project B, respectively, in Tables 18 and 19. We can make the following observations.

1. Similar to severity, we can observe that many of the network measures for the higher fixing workload group are significantly greater than in the lower fixing workload group. Generally speaking, more network measures demonstrate significant differences between the High group and the Low group, than high versus medium or medium versus low. Similarly, this result may be because of the larger difference in fixing workload between the High group and the Low group.
2. We still narrow our focus on the network measures with significant differences (significant for at least two tests). These network measures can also be mapped to the two categories: Connection (i.e., WeakComp and TwoStepReach) and Centrality (i.e., Closeness and Reachability). Although the detailed network measures show minor differences, the two categories are consistent regarding for number of bugs (Sect. 4.1) and severity of bugs (Sect. 4.2). This result further indicates that the greater the centrality of requirements and the more connection with other requirements, the more likely the associated bugs are to require a high fixing workload.
3. The significance levels of network measures exhibit tiny inconsistencies among different projects. Similar to severity, more network measures demonstrate differences among different fixing workloads for Project A than for Project B. Such network measures as EffSize and Efficiency (related to Connection) and Brokerage, nBrokerage, EgoBetweenness, nEgoBetweenness, Degree and Betweenness (related to Centrality) are significant only for Project A. These network measures overlap with the measures for bug severity to a large extent. Hence, we suppose the same reason for this phenomenon as mentioned for bug severity.
4. The correlation of network measures with fixing workload is weaker than the correlations with severity and number of bugs, for both projects. Moreover, fewer network measures exhibit differences in fixing workload than in severity, for both projects. We

Table 19 Significance level of one-tailed Mann–Whitney *U* test for bug fixing workload of Project B

	High versus medium	Medium versus low	High versus low
Ego network			
^{n} Size	0.549	0.076	0.047*
^{n} Ties	0.900	0.276	0.298
^{n} Pairs	0.491	0.078	0.042*
^{n} Density	0.697	0.530	0.386
^{o} AvgDist	0.058	0.955	0.106
^{n} WeakComp	0.966	0.006**	0.023*
^{n} nWeakComp	0.364	0.876	0.440
^{n} TwoStepReach	0.568	0.011*	0.008**
^{n} ReachEfficiency	0.427	0.085	0.313
^{t} Brokerage	0.711	0.087	0.058
^{t} nBrokerage	0.875	0.374	0.361
^{t} EgoBetweenness	0.737	0.083	0.060
^{t} nEgoBetweenness	0.821	0.411	0.356
Global network			
Centrality measures			
^{t} Degree	0.549	0.076	0.047*
^{t} Closeness	0.093	0.034*	0.000**
^{t} Reachability	0.149	0.007**	0.000**
^{t} Betweenness	0.468	0.063	0.020*
Structural holes measures			
^{n} EffSize	0.706	0.088	0.060
^{n} Efficiency	0.382	0.807	0.589
^{o} Constraint	0.193	0.176	0.028*
^{o} Hierarchy	0.392	0.549	0.228

Significance level at 0.01 are marked by **, and significance level at 0.05 are marked by *

The superscript denotes Connection (^{n}), Centrality (^{t}) or Other (^{o}) category assume that the fixing workload is easily influenced by the experience and proficiency of developers, which leads to the nonsignificant results. However, this assumption needs further exploration.

4.4 Summary

For research question Q1, "How significantly can network measures on requirements dependency network indicate software bugs?", we provide answers from three perspectives: number, severity and fixing workload.

First, significant correlation is observed between almost all our network measures and number of bugs, with the correlation coefficient up to 0.8. As hypothesized, central requirements are more bug-prone than requirements located in the surrounding areas of the network, and requirements with higher connection are more bug-prone than loosely connected requirements.

Furthermore, many network measures exhibit significantly greater values for higher severity (or higher fixing workload). These network measures mainly relate to centrality and connection, indicating that the higher the centrality and the higher the connection of the requirements, the more likely they are to be associated with high severity bugs and require a high fixing workload.

These observations show that network measures correlate with software bugs and can indicate software bugs. This association occurs because under the effects of complicated interactions and connections of requirements dependencies, requirements with high centrality and connection, which have more relationships with other requirements, are more likely to be prone to software bugs.

Moreover, due to the high-density dependencies with other requirements, requirements of high centrality and connection are likely to be prone to higher severity bugs. This association can be explained by the fact that bugs associated with these requirements might influence other dependent requirements, thus making these bugs more severe. The same reason applies to the bug fixing workload. Because requirements with high centrality and connection have more connections with other requirements, fixing bugs for these requirements might require more workload to consider the synergy effects of dependent requirements.

Furthermore, the correlation effect of network measures on bug number varies among different types of dependency network and different types of dependencies, with HybridNetwork working best for the correlation and the Constraint type of dependency contributing more to the correlation.

5 Bug prediction based on network measures

As network measures of requirements dependency network correlate with software bugs, especially for the number of bugs, can these measures be used to predict software bugs, and what is their predictive performance? For research question Q2, we focus on predicting the number of bugs. We built multiple linear regression models based on network measures in Tables 9 and 10. Furthermore, the idea behind prediction is to obtain models that can be used for prediction in future projects or other software companies. Therefore, we apply the prediction model in cross-project and cross-company contexts to further evaluate its efficacy. In addition, we further identify which network measures contribute more to bug prediction.

5.1 Experimental design

5.1.1 Building regression models

One difficulty associated with multiple regression analysis is multi-collinearity among the independent variables. Multi-collinearity arises from the inter-correlations among measures. For instance, Closeness and Reachability, which reflect different aspects of closeness centrality, have a correlation of 0.771 for Project A. Inter-correlations can lead to an inflated variance in the estimation of the dependent variable.

A common indicator of multi-collinearity is the Variable Inflation Factor (VIF). We started the linear regression analysis with all network measures in Tables 9 and 10 and iteratively removed the measures with the highest VIF until the statistics showed no evidence of multi-collinearity between the remaining variables (VIF <4.0) [3]. After this process, five measures for Project A, five measures for Project B and four measures for Project C remain. We reported these measures and their VIF in Table 20.

Our next step was to filter out the network measures that are not significant for predicting the number of bugs. We applied the four common measures for all projects (Table 20) to build linear regression models. For all 50 random

Table 20 Remaining measures and their VIF

Project A		Project B		Project C	
Variable	VIF	Variable	VIF	Variable	VIF
Ties	1.678	Ties	1.663	Ties	1.562
WeakComp	2.278	WeakComp	3.400	WeakComp	3.146
Brokerage	2.248	Brokerage	1.021	Closeness	3.053
Closeness	1.858	Closeness	2.803	Betweenness	2.630
Betweenness	1.964	Betweenness	2.630		

splits and all three projects, the *p* value for each model was less than 0.05, signifying that every model was significant at the 0.05 level. Furthermore, for all three projects and over 80 % of the 50 random splits, the *p* value for WeakComp, Closeness and Betweenness are less than 0.05, which indicates that these three measures are statistically significant at the 0.05 level for all three projects.

Consequently, these measures are significant predictors for bug prediction. We then built linear regression models with WeakComp, Closeness and Betweenness. These models were then evaluated by their explanative power and predictive power.

5.1.2 *Evaluating explanative power*

We measured the quality of the trained models using R-Square, which is the percentage of the variance in the dependent variable explained by the regression model. Values range from 0 to 1, where a high R-Square value denotes a model with high statistical explanative power but not predictive power. We also recorded the *p* value of each model to measure its significance level.

5.1.3 *Evaluating predictive power*

We evaluated the predictive power of linear regression models using accuracy and sensitivity. For accuracy, we computed the mean squared error (MSE). It quantifies the difference between the predicted number of bugs and the actual number of bugs. We computed MSE based on the relative difference, which is normalized by the actual number of bugs. The smaller the value, the lower the error and the less difference between the predicted values and actual values, which indicates higher predictive performance.

Regarding sensitivity, we computed the Spearman correlation coefficient between the predicted number of bugs and the actual number of bugs. As before, the closer the value is to -1 or $+1$, the more strongly the two measures are correlated. In our case, values close to 1 are desirable, as they indicate that an increase/decrease in the predicted values is accompanied by a corresponding increase/decrease in the actual number of bugs.

5.2 Within-project prediction

To evaluate the predictive performance of the network measures, we used a standard evaluation technique: data splitting [35]. That is, we randomly picked two-thirds of all requirements to build a prediction model and used the remaining one-third to measure the efficacy of the built model. We performed 50 random splits to ensure the stability and repeatability of our results.

We report the results of each random split in Fig. 3, in terms of explanative power (R-Square) and predictive power (MSE and Spearman correlation coefficient). Table 21 lists the descriptive statistics of the presented scatter plots.

Scatter plots and low values for the standard deviation of the performance measures show consistent results. The R-Square of Project C demonstrates slightly higher values and lower variance than for Project A and Project B, indicating that the models of Project C have higher explanative power. However, we can still observe that the network measures of all three projects can explain between 60 and 85 % of the variance, which indicates the efficacy of the built regression models. Every model was significant at 99 %.

This argument is further supported by consistently low MSE values and high Spearman correlation coefficients. Low and consistent MSE values indicate minor differences between the predicted and actual number of bugs. Furthermore, Spearman correlation coefficients of approximately 0.8 demonstrate the sensitivity of linear regression models, i.e., an increase/decrease in the predicted values is accompanied by a corresponding increase/decrease in the actual number of bugs. All correlations were significant at 99 %. Comparing the Spearman correlation coefficients, the prediction within Project B slightly outperforms the prediction within Project A and Project C, signifying a slightly better predictive performance.

5.3 Cross-project prediction

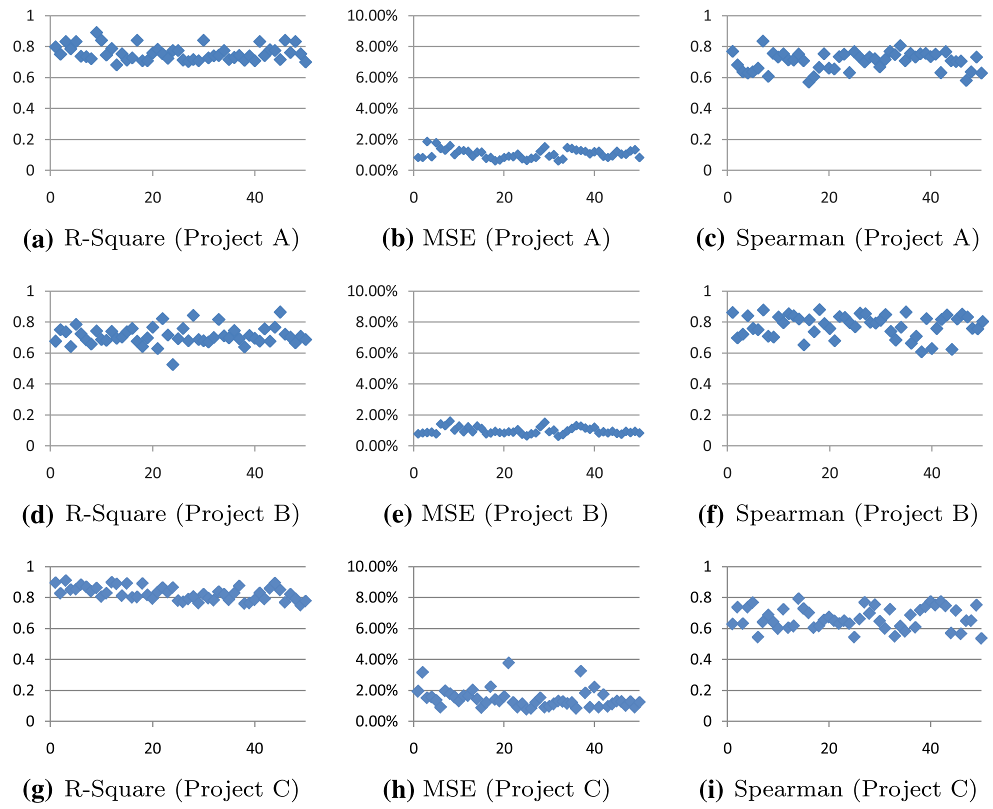
The idea behind predictive models is to obtain models that can be used for predictions in future projects. We trained models in one project and applied them to predict bugs in another project.

Due to data constraints, our experiment is set up as follows: We utilized all requirements in Project A to build a prediction model and then randomly picked two-thirds of all requirements in Project B to measure the efficacy of the built model when applied to another project. The random pick was repeated 50 times to ensure the stability and repeatability of our results.

R-Square of the built regression model is 0.775, which indicates the high explanative power of built model. Figure 4 depicts the predictive power of the models when applied to another project.

As seen in Fig. 4, the scatter plots demonstrate consistent results with low MSE values and high Spearman correlation coefficients. The Spearman correlation coefficients between 0.5 and 0.8 indicate that the model has high sensitivity, even when applied to another project.

Based on the above discussion, we can see that the model built by Project A can be used in a cross-project context with statistical significance. Table 22 also shows



(a) R-Square (Project A) **(b)** MSE (Project A) **(c)** Spearman (Project A)

(d) R-Square (Project B) **(e)** MSE (Project B) **(f)** Spearman (Project B)

(g) R-Square (Project C) **(h)** MSE (Project C) **(i)** Spearman (Project C)

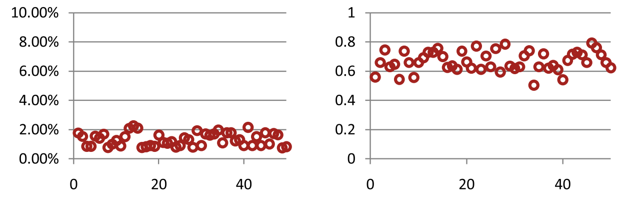
Fig. 3 Results of within-project prediction

Table 21 Statistics of results for within-project prediction

	Min	Max	Mean	SD
Project A				
R-Square	0.681	0.891	0.757	0.048
MSE	0.62 %	1.86 %	1.08 %	0.002
Spearman	0.570	0.835	0.704	0.059
Project B				
R-Square	0.526	0.863	0.710	0.057
MSE	0.62 %	1.59 %	0.98 %	0.002
Spearman	0.607	0.880	0.777	0.071
Project C				
R-Square	0.750	0.909	0.826	0.043
MSE	0.78 %	3.78 %	1.45 %	0.006
Spearman	0.537	0.792	0.670	0.072

the statistical results. Furthermore, how does its predictive performance compare to the within-project models? We compare Table 22 to the within-project prediction data for Project A in Table 21.

Models applied cross-project showed only a tiny decrease in the MSE values (1.34 % compared to 1.08 %) and Spearman correlation coefficients (0.67 compared to 0.70). This decrease is easy to understand, as the models built from Project A should have better performance in predicting Project A itself.



(a) MSE (Project A ->B) **(b)** Spearman (Project A ->B)

Fig. 4 Results of cross-project prediction

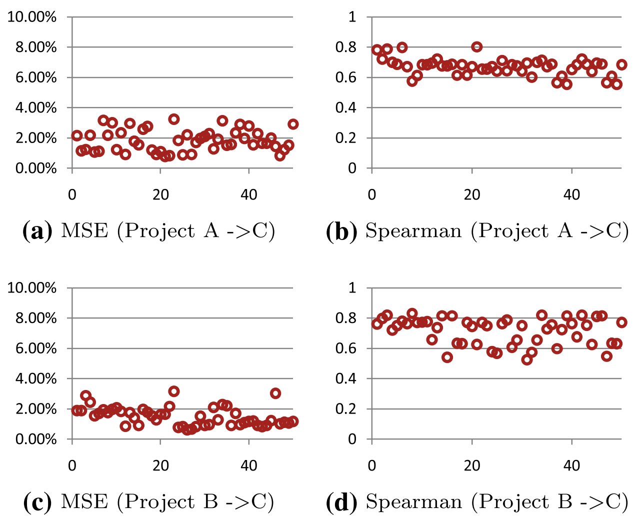
To conclude, the results from regression analysis indicate that we can apply the models in a cross-project context with comparable predictive power to within-project prediction. This result suggests that a new or ongoing project with little or no data can leverage data from previous projects for prediction purposes.

5.4 Cross-company prediction

Cross-company bug prediction is the art of using data from another company to build a bug predictor. It allows small software organizations or ones releasing their first product to use data from other organizations to build their quality prediction model. We trained models using data from one

Table 22 Statistics of results for cross-project prediction

	Min	Max	Mean	SD
Project A → Project B				
MSE	0.77 %	2.28 %	1.34 %	0.004
Spearman	0.505	0.793	0.668	0.067



(**a**) MSE (Project A ->C) (**b**) Spearman (Project A ->C)

(**c**) MSE (Project B ->C) (**d**) Spearman (Project B ->C)

Fig. 5 Results of cross-company prediction

company and applied them to predict bugs in another company.

Due to data constraints, our experiment is set up as follows: We utilized all requirements in Project A (or Project B) to build a prediction model and randomly picked two-thirds of all requirements in Project C to measure the efficacy of the built model when applied to projects of another company. The random pick was repeated 50 times to ensure the stability and repeatability of our results.

R-Square of the built regression model is 0.648 for Project A and 0.642 for Project B, which shows the high explanative power of the built model. Figure 5 depicts the predictive power of the models when applied in a cross-company context.

As seen in Fig. 5, the scatter plots demonstrate consistent results with low MSE values and high Spearman correlation coefficients. Low and consistent MSE values indicate a minor difference between the predicted number of bugs and the actual number of bugs. The Spearman correlation coefficients between 0.5 and 0.8 indicate that the model exhibits high sensitivity even when applied to projects of another company.

Based on the above discussion, we can see that the model built using data in one company can be applied in a cross-company context with statistical significance. Table 23 also shows the statistical results. Furthermore, how

Table 23 Statistics of results for cross-company prediction

	Min	Max	Mean	SD
Project A → Project C				
MSE	0.78 %	3.25 %	1.84 %	0.007
Spearman	0.555	0.803	0.670	0.057
Project B → Project C				
MSE	0.63 %	3.17 %	1.52 %	0.006
Spearman	0.527	0.832	0.716	0.088

does its predicting performance compare to within-project models? We compare Table 23 with the data on the within-project prediction of Project A and Project B in Table 21.

Models applied cross-company showed only a tiny decrease in the MSE values (1.84 % compared to 1.08 % for Project A, 1.52 % compared to 0.98 % for Project B) and Spearman correlation coefficients (0.67 compared to 0.70 for Project A, 0.72 compared to 0.77 for Project B). It can similarly be attributed to the fact that the prediction model within Project A and Project B is built in and applied to the same project.

For models built using the data from Project A, we compared the performance when applied in a cross-project context (to Project B, see Table 22) and in a cross-company context (to Project C, see Table 23). Cross-company prediction only exhibits a tiny decrease in MSE (1.34 % compared to 1.84 %) and remains unchanged in the Spearman correlation coefficient (0.668 compared to 0.670).

In conclusion, the results from linear regression indicate that we can apply the prediction models in a cross-company context with comparable predictive power to within-project prediction, as well as cross-project prediction within the same company. This result suggests that organizations with little historical data can leverage data from other organizations for prediction.

5.5 Which network measures contribute most to bug prediction?

We have mentioned that WeakComp, Closeness and Betweenness are significant for predicting the number of bugs. This result roughly coincides with the observations in call and data flow dependency networks [38]. Furthermore, these three measures can be used for bug prediction with high accuracy and sensitivity. Their positive coefficients in regression models signify that an increase in WeakComp, Closeness and Betweenness values may correspond to an increase in the number of bugs.

Closeness and Betweenness are Centrality measures that denote the "central positions" from different perspectives, which indicates that central requirements are more bug-prone than requirements located in the surrounding areas of the network. This result is supported by the strong correlation between these centrality measures and the number of bugs (see Sect. 4.1). Furthermore, similar observations have been obtained for call and data flow dependency networks [53] and developer-module networks [41]. For example, Pinzger et al. [41] showed that the centrality of software modules in a developer-module network can be used to predict bugs with high performance.

WeakComp, which represents Connection, is the number of unconnected components in the ego network. The rationale is that if the ego depends on many requirements that are not connected to each other, then the ego requirement has a higher risk of bugs. We suppose this value to be an extension of Degree, which is the number of nodes on which an ego depends. The intuitive interpretation of Degree is that when a requirement depends on many other requirements, there is a high risk of bugs. WeakComp adds to this interpretation that if all the dependent requirements are completely independent of each other, then the risk of bugs increases. WeakComp can also be treated as a variant of centrality in the ego network.

The above explanation further confirms the conjecture that requirements of high centrality and connection are more bug-prone than loosely connected requirements and requirements located in the surrounding areas of the network. This result coincides with the observations in correlation analysis (Sect. 4). However, one central question remains open: "What makes the central and highly connected requirements more bug-prone?" Our future work will concentrate on this question. These significant predictors can help to identify areas where the design can be improved or where test efforts should be prioritized.

5.6 Summary

For research question Q2, "Can network measures on requirements dependency network be used to predict software bugs, and what is their predictive performance?', the results of linear regression show that network measures on requirements dependency network can be used to predict the number of bugs with high accuracy and sensitivity. Furthermore, the prediction model can be applied in cross-project and cross-company contexts with comparable predictive performance to within-project prediction.

WeakComp, Closeness and Betweenness are the significant predictors for bug prediction. This result indicates that central and highly connected requirements are more bug-prone than loosely connected requirements and requirements located in the surrounding areas of the network.

Previous efforts have shown that network measures on architecture dependencies [10, 36], call and data flow dependencies [6, 53], communication and coordination relationships [11, 50] are significant predictors of software bugs. The dependency relationship at the requirements level is the inherent driver of architecture dependencies and source code dependencies as well as coordination and interaction structure. From this perspective, requirements dependency relationship should be predictors of software bugs.

It is commonly perceived that different projects or companies usually posses dissimilar contexts, e.g., the process, developers and domain. This phenomenon is commonly known as data shift [45] and can decrease the accuracy when applying prediction models to different projects or companies. Requirements dependency is inherent in software development and is less influenced by development contexts and process. Moreover, we utilized the network structure of requirements dependencies for bug prediction instead of the requirements or the requirements dependencies themselves. The behavior of the network structure is similar across projects or companies, thus rendering the models robust over time and serving as the foundation of cross-project and cross-company prediction.

The explanative power and predictive power obtained in our experiments are of the same order as obtained by later lifecycle predictions, such as Zimmermann et al.'s [53] work predicting defects based on call and data flow dependency networks with R-Square of 0.5 and Spearman correlation coefficients close to 0.6, or Nagappan et al.'s [41] work predicting post-release failures from developer-module networks with R-Square between 0.69 and 0.74 and Spearman correlation coefficients between 0.75 and 0.80. We cannot, however, draw solid conclusions from such comparisons as they are based on studies involving different projects, bug types and bug densities.

6 Discussion

6.1 Threats to validity

Construct validity issues arise when there are errors in the measurement. This problem is addressed to some extent because the engineers in the company had no knowledge that this study was being performed and thus could not artificially modify their behavior to affect our measurements. The study does suffer from experimenter bias, that is, bias toward a result expected by the human experimenters. To minimize this bias, two assistant researchers worked independently to collect and process data, and the first author analyzed the results.

There are many possible sources of bugs, including technical attributes, developer experience, mental and physical stress and tool support. Our approach focuses only on a small portion of bugs: bugs associated with the integration of software components. We aim to predict integration bugs from the standpoint of requirements dependency because practical experience suggests that integration bugs may be caused by complicated interactions or connections under the effects of requirements dependencies.

Integration bugs can be caused by defects introduced throughout the software lifecycle, such as ambiguities or inconsistencies in requirements description, incorrect design or implementation. Instead of detecting defects, we wish to predict bugs, and we wish to predict these bugs from information already present in the requirement phase, serving as a useful complement to existing code-based prediction models.

From an internal validity perspective, the accuracy of identifying requirements dependencies could influence bug prediction performance. We compensated for this unreliability through manual identification of dependencies. Moreover, several practitioners with different backgrounds and experiences such as requirements analysis, design and development were involved in the construction process. In addition, we conducted interviews to discuss the differences and similarity among their separate constructions until a common consensus was reached. However, one can apply automatic identification techniques instead of manual identification to identify requirements dependencies as long as the accuracy is sufficiently high.

In our study, certain performance requirements, such as response time and concurrent user access, were not included in the integration testing. How such non-functional requirements affect integration bugs was not discussed. However, the situation can vary in other software organizations. Regardless, these performance requirements are generally considered to be less influenced by requirements dependencies. Furthermore, all the integration bugs can be associated with requirements in our case (see Sect. 3.4). From this perspective, we believe that this treatment does not affect the final results.

From an external validity perspective, our experiment was performed using data collected from two medium-sized software organizations. However, we cannot assume a priori that the results of our study can be generalized beyond the specific environment in which it was conducted. To alleviate this problem, we utilized empirical data from three separate projects with different development times and development teams. Moreover, we have begun to collect data from other companies and open source systems to replicate these studies to further test for external validity.

Table 24 Cost to build the prediction model

	Project A	Project B	Project C
Dependency identification			
Identification for 1/4 requirements	80 min$_{(77)}$	90 min$_{(84)}$	24 min$_{(19)}$
Follow-up interviews for 1/4 requirements	30 min	40 min	12 min
Identification for remaining requirements	180 min$_{(231)}$	180 min$_{(250)}$	46 min$_{(56)}$
Follow-up interviews for remaining requirements	30 min	35 min	8 min
Network construction and network measures computation	5 min	5 min	5 min
Bug association			
First step	70 min$_{(50)}$	80 min$_{(50)}$	70 min$_{(50)}$
Follow-up interviews for first step	50 min	55 min	40 min
Second step	50 min$_{(50)}$	55 min$_{(50)}$	40 min$_{(50)}$
Follow-up interviews for second step	30 min	25 min	20 min
Third step	520 min$_{(632)}$	340 min$_{(318)}$	160 min$_{(186)}$
Follow-up interviews for third step	30 min	30 min	25 min
Total	17.9 h	15.5 h	7.5 h

The subscript signifies the number of requirements or bugs

6.2 Cost-benefit analysis

We summarize the total cost associated with the prediction in Table 24. It can be divided into three parts: dependency identification (Sect. 3.2), network construction and network measures computation (Sects. 3.3, 3.5) and bug association (Sect. 3.4).

There are also techniques for the automatic identification of requirements dependencies (summarized in Sect. 7.1), which can help identify requirements dependencies and reduce manpower costs, thereby making our approach more cost-effective. Furthermore, techniques for requirements traceability [14] can assist in bug association, which can further reduce the effort.

As described previously, this empirical study was performed in two medium-sized software organizations with mature development processes. The projects selected for analysis have hundreds of requirements and bugs, which is representative and typical in the industry environment. From this perspective, this approach can easily be generalized to other projects to obtain project-specific bug information. The prediction of bug proneness supports the following two tasks in the industry environment.

Resource allocation. Software quality assurance consumes considerable effort in any large-scale software development. To increase the effectiveness of this effort, it is wise to invest more attention in the elements that are most likely to fail and thus most need quality assurance. With our prediction, this investment can be made at the requirements phase, when decisions matter most. In our experimental setup (see Sect. 5), the top 20 % of bug-prone requirements identified by our prediction model are associated with 71–85 % of the total bugs.

Decision making. Such prediction can aid engineers in assessing the risk of forming new dependencies or supporting existing ones. Meanwhile, alternative requirements can be explored and assessed for the lowest risk. Requirements dependencies can propagate to the code level, forming call and data flow relationships, especially the Constraint type of dependencies (see Sect. 4.1.1). Therefore, during the design and implementation phase, additional attention should be paid to these requirements with higher centrality to loosen the potential couplings and lower the risk of software defects.

The aim of this approach is to build prediction models based on network measures of requirements dependency network, and quantitatively analyze and predict software bugs. We have also demonstrated the clear correlation between bug occurrence and such network measures as centrality. These centrality measures can imply the patterns of dependencies, e.g., the star motif (see Sect. 2). From this perspective, when project managers observe a dependency pattern, they can qualitatively infer that those central requirements are more prone to software bugs. It means that the results of this approach can be applied even without constructing the whole dependency network and obtaining network measures. Therefore, although creating dependency network and computing network measures are relatively effort consuming, our approach can be applied for bug indication qualitatively, which is more cost-effective.

7 Related work

In this section, we discuss work related to this study, which falls into three categories: requirements dependency, early bug prediction and network analysis.

7.1 Requirements dependency

More than twenty types of requirements dependency have been proposed to reflect the relationships between requirements at both the structural and semantic levels. Pohl [42] proposed a dependency model including eighteen dependency types based on a literature survey in the area of requirements engineering. Later, Dahlstedt and Persson [15] proposed another dependency model including seven dependency types, compiling existing definitions into an integrated view. Zhang et al. [52] introduced four types of dependency between features and further presented a feature-driven approach to analyze requirements dependencies and design high-level software architecture.

There are various techniques for the automatic identification of requirements dependencies. Techniques for identifying cross-cutting concerns in aspect-oriented software development [4, 12, 16] can be applied to identify the Constraint type of requirements dependencies. Chichyan and Rashid [13] discussed how requirements dependencies can be deduced from the semantics of requirements. Their technique can identify such requirements dependencies as constrain, like, use and compare, covering our three types of dependencies. Our previous approach [27] employed information retrieval techniques to compute similarity among requirement fragments and then design corresponding algorithms to automatically identify Similarity and Reference dependencies. The first type corresponds to the Similar_to dependency, and the second type reflects aspects of the Constraint and Precondition dependencies in this paper. These techniques can help identify requirements dependencies and reduce manpower costs, thereby making this approach more cost-effective.

Requirements dependency can affect various decisions and activities during development. Carlshamre et al. [9] conducted an industrial survey of requirements dependencies in software product release planning. They found that only approximately 20 % of requirements are singular and emphasized the need to explore requirements dependencies. Moreover, their work identified six types of requirement dependencies for release planning. Li et al. [25] introduced an integrated approach for requirement selection and scheduling in software release planning, with a detailed analysis of the influence of requirements dependencies identified in Carlshamre et al.'s research [9].

Similarly, Al-Emran et al. [2] stated that a good release plan should reflect existing dependencies between requirements and further presented a release planning approach taking requirements dependency into account. Sellier et al. [43] developed a meta-model for product line requirements selection decisions that includes dependencies.

Prior research has also demonstrated the effects of requirements dependency on change propagation. In our previous work, we utilized five types of requirements dependency from Dahlstedt's dependency model [15] to simulate the impact of requirement changes on project plan [48]. McGee et al. [33] conducted two case studies to investigate the causes and effects of requirement changes. The results indicated that higher requirements dependency gives rise to both increased change frequency and high volatility. Due to the change propagation effect, Navarro et al. [37] introduced semantic decoupling to restrict the functional dependencies between modules, which further reduced the impact of requirement changes.

Another of our prior works was a case study conducted to evaluate the usefulness and applicability of Pohl's dependency model [42] and Dehlstedt's dependency model [15] covering 25 dependency types, for change propagation analysis [26, 51]. We found three dependency types that do propagate changes and further clarified their definitions. The dependencies involved in this approach are just these three dependency types. We believe that these dependencies, which together capture the interactions between requirements, can serve as a valuable indicator of software quality.

7.2 Early bug prediction

The representative work of early bug prediction utilized causal models based on a wide range of qualitative and quantitative factors concerning a project (such as its size, the regularity of specification reviews, the level of stakeholder involvement, and programmer ability) [17, 30]. Experience has shown that it is usually difficult to obtain reliable data of the required granularity or volume [17]. Our work requires less data than these models.

Park et al. [40] analyzed six requirements-related attributes, including the number of indirect stakeholders and number of related stories, to discover the occurrence of requirements-related defects. They also performed network analysis on stakeholder relationships to further explore predictive ability. In contrast to their work, we construct network based on requirements dependency relationships.

He et al. [22] proposed an early bug prediction method based on enhanced requirements. Fitzgerald et al. [19] performed early failure prediction based on the characteristics of online discussions of feature requests. These two approaches utilized the content of requirements for prediction, while our prediction is based on the relationships among requirements. Intuition suggests that these two sources of information together interact to influence the quality of software, which suggests possible topics for future research.

Another type of research is concerned with defect identification in the early phases of the software lifecycle. Through a systematic review, Walia et al. [46] developed a requirement-related taxonomy of errors that may occur during the requirements phase. Ott [39] investigated the typical defect type distributions in current natural language requirement specifications. Katasonov et al. [23] developed a unifying framework for requirement quality control. Instead of detecting requirement-related defects, our approach is to predict bugs, which can be caused by defects introduced throughout the software lifecycle.

7.3 Network analysis

Network analysis has been frequently used to study the process and organizational aspects of software engineering. Examples include, but are not limited to, examining latent structure based on email exchange networks [7], predicting defects with call and data flow dependency networks [53], and exploring collaboration and communication in task-based social networks [50].

We present a detailed illustration of related approaches that applied network analysis to the requirements engineering domain. Marczak and Damian [31, 32] used social network analysis to explore the information flow patterns and communication structures in the collaboration driven by interdependent requirements in software teams. They focused on the brokerage of information between members of interdependency teams, the emergent communication structure and the reasons for inter-role communication and cross-functional interaction.

Soo Ling Lim et al. [29] developed StakeNet to identify and prioritize stakeholders. StakeNet identified stakeholders and asked them to recommend other stakeholders and stakeholder roles, build a social network whose nodes are stakeholders and links are recommendations, and prioritize stakeholders using a variety of social network measures. They further proposed StakeRare, which uses social networks and collaborative filtering to identify and prioritize requirements and to facilitate the elicitation of requirements [28].

Using network to represent relationships enables the connection and structure to be visualized. Furthermore, by computing network measures, the whole network can be transformed into a series of values based on the nodes' positions from different perspective. Thus, we apply network analysis to requirements dependencies to investigate their influence on software quality.

8 Conclusion and future work

Software engineers can benefit from an early estimate regarding the quality of their product as field quality information is often available only late in the software lifecycle, to affordably guide corrective actions. Identifying early indicators and predictors of field quality is important in this regard.

In this paper, we studied requirements dependency relationships from the standpoint of software integration bugs in three commercial software projects in two software companies.

Our findings showed that network measures on requirements dependency network can indicate software bugs from the following points of view:

- Most of our network measures on requirements dependency network correlate significantly with the number of bugs, and central and highly connected requirements are more bug-prone (Sect. 4.1).
- Many network measures, mainly related to Connection and Centrality, demonstrate significantly greater values for higher severity (or fixing workload) (Sects. 4.2, 4.3).
- The correlation effect of network measures varies among different types of requirements dependency, and the *Constraint* type of dependency contributes more to bug indication (Sect. 4.1.1).

Our findings also showed that network measures on requirements dependency network can be used to predict software bugs with high predictive performance from the following perspectives:

- Network measures on requirements dependency network can predict the number of bugs with high accuracy and sensitivity (Sect. 5.2).
- Prediction models can be applied in cross-project and cross-company contexts with comparable predictive performance as within-project prediction (Sects. 5.3, 5.4).
- WeakComp, Closeness and Betweenness are the significant predictors for bug prediction, which indicates that requirements with high centrality and connection are more bug-prone (Sect. 5.5).

These findings can support managers in the task of allocating resources such as time and cost for quality assurance. In addition, such prediction can aid engineers in assessing the risk of forming new dependencies or supporting existing ones.

Drawing general conclusions from empirical studies is difficult, and we cannot assume a prior that our findings can be generalized beyond the specific environment in which the study was conducted [5]. For this reason, we plan to replicate the study with other commercial and open source software projects, as well as conducting large-scale cross-project and cross-company prediction to further evaluate our approach.

Furthermore, we do not claim that requirements dependency data are the sole predictor of integration bugs; however, our results are another piece in the puzzle of why software fails. In future work, we seek to explore other requirement-related indicators for software bugs, such as the implementation relationship between requirements and program elements, the dependency weighting schema and the communication structure within requirement teams.

Although the results confirmed that high centrality and highly connected requirements are more bug-prone than loosely connected requirements and requirements located in the surrounding areas of the network, one question remains open: “What makes the central and highly connected requirements more bug-prone?” Our future work will concentrate on this question.

Acknowledgments This work is supported by the National Natural Science Foundation of China under Grant Nos. 91318301, 91218302 and 61432001.

References

1. Systems and software engineering—software life cycle processes. ISO/IEC 12207:2008
2. Al-Emran A, Pfahl D, Ruhe G (2010) Decision support for product release planning based on robustness analysis. In: Proceedings of 18th IEEE international requirements engineering conference (RE’2010). Sydney, Australia, pp 157–166
3. Allison PD (1999) Multiple regression: a primer. Pine Forge Press, Thousand Oaks
4. Baniassad E, Clarke S (2004) Finding aspects in requirements with theme/doc. In: Workshop on early aspects: aspect-oriented requirements engineering and architecture design workshop. Lancaster, UK, pp 1–8
5. Basili V, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Trans Softw Eng 25(4):456–473
6. Bhattacharya P, Iliofotou M, Neamtiu I, Faloutsos M (2012) Graph-based analysis and prediction for software evolution. In: Proceedings of IEEE international conference on software engineering (ICSE’2012). Zurich, Switzerland, pp 419–429
7. Bird C, Pattison D, D’Souza R, Filkov V, Devanbu P (2008) Latent social structure in open source projects. In: Proceedings of 16th ACM SIGSOFT international symposium on foundations of software engineering (FSE’2008). Atlanta, Georgia, USA, pp 24–35
8. Borgatti SP, Everett MG, Freeman LC (2002) Ucinet for windows: software for social network analysis. Analytic Technologies, Harvard
9. Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001) An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of 5th IEEE international symposium on requirements engineering (RE’2001). Cape Town, South Africa, pp 84–91
10. Cataldo M, Herbsleb JD (2011) Factors leading to integration failures in global feature-oriented development: an empirical analysis. In: Proceedings of IEEE international conference on software engineering (ICSE’2011). Waikiki, Honolulu, HI, USA, pp 161–170
11. Cataldo M, Mockus A, Roberts JA, Herbsleb JD (2009) Software dependencies, work dependencies and their impact on failures. IEEE Trans Softw Eng 35(6):864–878
12. Chen K, Zhao H, Zhang W, Mei H (2006) Identification of crosscutting requirements based on feature dependency. In: Proceedings of 14th IEEE international requirements engineering conference (RE’2006). Minneapolis, USA, pp 307–310
13. Chitchyan R, Rashid A (2006) Tracing requirements interdependency semantics. In: Workshop on early aspects: traceability of aspects in the early life cycle. Bonn, Germany
14. Cleland-Huang J, Settimi R, Duan C, Zou X (2005) Utilizing supporting evidence to improve dynamic requirements traceability. In: Proceedings of 13th IEEE international conference on requirements engineering (RE’2005). Paris, France, pp 135–144

15. Dahlstedt A, Persson A (2005) Requirements interdependencies: State of the art and future challenges. In: Aurum A, Wohlin C (eds) Engineering and managing software requirements. Springer, Heidelberg, pp 95–116
16. Duan C, Cleland-Huang J (2007) A clustering technique for early detection of dominant and recessive cross-cutting concerns. In: Proceedings of the early aspects at ICSE: workshops in aspect-oriented requirements engineering and architecture design. Minneapolis, USA
17. Fenton N, Neil M, Marsh W, Hearty P, Radlinski L, Krause P (2008) On the effectiveness of early life cycle defect prediction with bayesian nets. Empir Softw Eng 13:499–537
18. Fenton NF, Pfleeger SL (1998) Software metrics: a rigorous and practical approach. Brooks/Cole, CA
19. Fitzgerald C, Letier E, Finkelstein A (2012) Early failure prediction in feature request management systems: an extended study. Requir Eng 17:117–132
20. Giventer L (2007) Statistical analysis in public administration. Jones and Bartlett, Burlington
21. Hanneman RA, Riddle M (2005) Introduction to social network methods. University of California, Riverside
22. He L, Li J, Wang Q, Yang Y (2009) Predicting upgrade project defects based on enhancement requirements: an empirical study. In: Proceedings of international conference on software process (ICSP'2009). Vancouver, Canada, pp 268–279
23. Katasonov A, Sakkinen M (2006) Requirements quality control: a unifying framework. Requir Eng 11:42–57
24. Kobayashi K, Matsuo A, Inoue K, Hayase Y, Kamimura M, Yoshino T (2011) ImpactScale: quantifying change impact to predict faults in large software systems. In: Proceedings of 27th IEEE international conference on software maintenance (ICSM'2011). Williamsburg, USA, pp 43–52
25. Li C, Akker M, Brinkkemper S, Diepen G (2010) An integrated approach for requirement selection and scheduling in software release planning. Requir Eng 15:375–396
26. Li J, Zhu L, Jeffery R, Liu Y, Zhang H, Wang Q, Li M (2012) An initial evaluation of requirements dependency types in change propagation analysis. In: Proceedings of 16th international conference on evaluation and assessment in software engineering (EASE'2012). Ciudad Real, Spain, pp 62–71
27. Li Y (2010) Requirement change impact analysis model and research on related approaches. Phd thesis, Institute of Software Chinese Academy of Sciences, Beijing, China
28. Lim SL, Finkelstein A (2012) StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation. IEEE Trans Softw Eng 38(3):707–735
29. Lim SL, Quercia D, Finkelstein A (2010) StakeNet: using social networks to analyse the stakeholders of large-scale software projects. In: Proceedings of IEEE international conference on software engineering (ICSE'2010). Cape Town, South Africa, pp 295–304
30. Madachy R, Boehm B (2008) Assessing quality processes with ODC COQUALMO. Lecture notes in computer science, making globally distributed software development a success story
31. Marczak S, Damian D (2011) How interaction between roles shapes the communication structure in requirements-driven collaboration. In: Proceedings of 19th IEEE international requirements engineering conference (RE'2011). Trento, Italy, pp 47–56
32. Marczak S, Damian D, Stege U, Schroter A (2008) Information brokers in requirement-dependency social networks. In: Proceedings of 16th IEEE international requirements engineering conference (RE'2008). Barcelona, Catalunya, Spain, pp 53–62
33. McGee S, Greer D (2012) Towards an understanding of the causes and effects of software requirements change: two case studies. Requir Eng 17:133–155
34. Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U (2002) Network motifs: simple building blocks of complex networks. Science 298:824–827
35. Munson J, Khoshgoftaar T (1992) The detection of fault-prone programs. IEEE Trans Softw Eng 18:423–433
36. Nagappan N, Ball T (2007) Using software dependencies and churn metrics to predict field failures: An empirical case study. In: Proc. IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'2007), pp. 364–373. Madrid, Spain
37. Navarro I, Leveson N, Lunqvist K (2010) Semantic decoupling: reducing the impact of requirement changes. Requir Eng 15:419–437
38. Nguyen THD, Adams B, Hassan AE (2010) Studying the impact of dependency network measures on software quality. In: Proceedings of 26th IEEE international conference on software maintenance (ICSM'2010). Timisoara, Romania, pp 1–10
39. Ott D (2012) Defects in natural language requirement specifications at Mercedes-Benz: an investigation using a combination of legacy data and expert opinion. In: Proceedings of 20th IEEE international requirements engineering conference (RE'2012). Chicago, Illinois, USA, pp 291–296
40. Park S, Maurer F, Eberlein A, Fung T (2010) Requirements attributes to predict requirements related defects. In: Proceedings of conference of the center for advanced studies on collaborative research (CASCON'2010). Toronto, Canada, pp 42–56
41. Pinzger M, Nagappan N, Murphy B (2008) Can developer-module networks predict failures? In: Proceedings of 16th ACM SIGSOFT international symposium on foundations of software engineering (FSE'2008). Atlanta, Georgia, USA, pp 2–12
42. Pohl K (1996) Process-centered requirements engineering. Wiley, Hoboken
43. Sellier D, Mannion M, Mansell JX (2008) Managing requirements inter-dependency for software product line derivation. Requir Eng 13:299–313
44. Subramanyam R, Krishnan MS (2003) Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects. IEEE Trans Softw Eng 29:297–310
45. Turhan B (2012) On the dataset shift problem in software engineering prediction models. Empir Softw Eng 17:62–74
46. Walia G, Carver J (2009) A systematic literature review to identify and classify software requirement errors. Inf Softw Technol 51(7):1087–1109
47. Wang J, Li J, Wang Q, Yang D, Zhang H, Li M (2013) Can requirements dependency network be used as early indicator of software integration bugs? In: Proceedings of 21th IEEE international requirements engineering conference (RE'2013). Rio de Janeiro, Brazil, pp 185–194
48. Wang J, Li J, Wang Q, Zhang H, Wang H (2012) A simulation approach for impact analysis of requirement volatility considering dependency change. In: Proceedings of 18th international working conference on requirements engineering: foundation for software quality (REFSQ'2012). Essen, Germany, pp 59–76
49. Wolf T, Schroter A, Damian D, Nguyen T (2009) Predicting build failures using social network analysis on developer communication. In: Proceedings of IEEE international conference on software engineering (ICSE'2009). Vancouver, Canada, pp 1–11
50. Wolf T, Schroter A, Damian D, Panjer LD, Nguyen TH (2009) Mining task-based social networks to explore collaboration in software teams. IEEE Softw 26(1):58–66
51. Zhang H, Li J, Zhu L, Jeffery R, Liu Y, Wang Q, Li M (2014) Investigating dependencies in software requirements for change propagation analysis. Inf Softw Technol 56:40–53
52. Zhang W, Mei H, Zhao H (2006) Feature-driven requirement dependency analysis and high-level software design. Requir Eng 11:205–220
53. Zimmermann T, Nagappan N (2008) Predicting defects using network analysis on dependency graphs. In: Proceedings of IEEE international conference on software engineering (ICSE'2008). Leipzig, Germany, pp 531–540