Requirements Attributes to Predict Requirements Related Defects

Shelly Park
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, AB, Canada

Frank Maurer
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, AB, Canada

Armin Eberlein
Dept. of Computer Science & Engineering
American University of Sharjah
PO Box 26666
Sharjah, United Arab Emirates

Tak-Shing Fung
Information Technologies
University of Calgary
2500 University Drive NW
Calgary, AB, Canada

Abstract

Literature suggests that requirements defects are a very costly problem to fix. Understanding how requirements changes influence the overall quality of software is important. Having some defect predictors at the requirements stage may help the stakeholders avoid making choices that could bring about catastrophic defect numbers at the end or at least be prepared for it. In this paper, six requirements-related attributes are analyzed to discover if they can be used for determining the occurrences of requirements-related defects. We measured two types of attributes: point and aggregate. The point attributes include time estimates, priority and ownership. The aggregate attributes include the number of indirect stakeholders, the number of related stories and the story creation time. Our analysis is based on data from the development of the IBM Jazz system. Our result shows that the number of indirect stakeholders and the number of related stories are good predictors for the number of defects, but other attributes show no or little correlation with the defects.

1 Introduction

The aim of our research is to data mine a structure, particularly on the relationships among requirements, people and software defects in large software development data. Literature suggests that requirements-related defects are a very costly problem to fix. According to Fairleys estimation, the cost of fixing requirements defects may rise by 20 to 50 times if the defects are fixed in the later stage of the development [7]. Boehm and Basili put that number as high as 100 times [2]. Up to 85% of the defects are estimated to come from the requirements [11]. Literature states that requirements changes or introducing new requirements increase the defect rate to about 50% [14]. Software is developed by people, which means that much of the software development processes and their results are influenced by people. We want to find out how these numbers hold in an iterative development environment that used stories and a Jazz software collaboration tool to communicate the requirements.

The discipline of statistics and data mining are both concerned with discovering structures in data [10]. A large body of data may contain some valuable structures which may provide more interesting or useful insights into a phenomenon under study. Statistics is gener-

ally concerned with how to make statements about a population by examining a sample of the population. On the other hand, data mining is concerned with an entire population. In such situations, statistical model building is used to find significance of the model fit rather than the probabilistic statement about the generalization ability from a sample [10]. Data mining deals with searching for variables that may have some good predictive abilities and try to find potential explanatory variables. In data mining, we are more interested in the exploratory aspect of discovering potentially interesting relationships between many variables [10].

In this paper, we report the results of data mining a software repository of a team that uses stories for communicating requirements. We data mined their development repository for over a year to find out if any interesting structures or patterns exist on the relationships among requirements, people and defects. Stories are requirements that are broken down into smaller features, the implementation effort of which can be easily estimated. We are interested in whether defects can be linked to one or more stories and explore the variables that are related in these two artifacts. To the best of our knowledge, this is the first defect prediction research that looked at requirements attributes from stories.

Defect prediction is a research area with a long tradition that aims to find metrics that are available in the early phase of software development and that are good defect predictors [19]. However, defect prediction research tends to evaluate defects from the coding stage onwards using attributes that are available at the coding stage, such as code churn, lines of code or the number of file changes [21]. Even though analyzing defects using the code will likely yield better prediction accuracy, being able to predict defects, even with less accuracy, already during requirements specification may allow teams to make better implementation decisions. Our goal was to find out whether there are lightweight requirements definitions that can be used for predicting defects or at least find out if such patterns are available at the requirements level.

Based on the Jazz systems change history and the people who work on the project, we hypothesized that easily attainable requirements-related attributes could exist in our data for predicting the estimated defects count. Our reasoning is that stakeholders may hold tacit knowledge about a projects health, which may manifest itself in some human-based attributes that can be measured at the time of the requirements specification. Therefore, our hypothesis is that there are measurable story attributes that can provide reasonable defect prediction. Our analysis was exploratory in nature and set out to discover whether such variables exist in our data. The benefit of being able to predict defects using requirements expressed as stories is to help prepare better for negotiations, estimations and planning for the risks of a high number of defects.

We performed a case study on a large software development project the IBM Jazz development project - that used stories for communicating requirements. More details about the data will be presented in Section 3. We performed a statistical analysis on six variables that we data mined from our repository. From those variables, we found that two variables had high correlations with the number of defects: the Number of Indirect stakeholders and the Number of Related Stories. And then we performed a network analysis on our data based on these two variables to find out what kind of patterns exists.

The pattern analysis is performed using network analysis. Network analysis is a popular analysis technique for understanding the social relationships between humans and their interactions. Each actor (human) is tied to other actors through dependency ties. In our study, ties between stakeholders are made if the stakeholder reports a defect that is dependent on another defect created by a different person. We then find out if there is a pattern in the clustering of people and defect types. Thus, we determine whether the two variables we discovered through the statistical analysis had some interesting network patterns that can further explain their ability to predict the number of defects.

The organization of the paper is as follows. Section 2 motivates our research. Section 3 describes the development project, the data of

which we analyzed. Section 4 describes the research design. Section 5 describes our results. In Section 6, we discuss our findings. Section 7 describes the threats to the validity of our research. Section 8 concludes the paper.

2 Literature Survey

2.1 Association between Requirements and Defects

The literature currently provides many numbers for the magnitude of requirements defects [7, 2, 11]. According to Mogyorodi, 56% of all the bugs in software projects are inserted in the requirements phase. From them, half of these bugs are due to incomplete and ambiguous requirements and the other half are due to requirements omissions [18]. However, we still do not know whether there were already some good measurable attributes at the time of the requirements that could be used to predict defects.

Jones state that the defect rates increase to about 50% if the requirements are changed in mid-development [14]. The surveys suggest that about half of the respondents said the major cause of defects was poor requirements [12]. Javed et al. performed a study that compared the defect rate for pre-release and post-release requirements changes [13]. Their analysis showed that 81% of change requests came after the project was shipped to the client and that these requests have caused 67% of the most severe defects. Zowghi and Nurmuliani found that requirements volatility can contribute to a schedule overrun, but frequent communications and usage of a methodology can help stabilize the schedule [28].

Although the literature suggests that requirements changes are dangerous and should be avoided, requirements changes are often an unavoidable part of software development. What software engineers require is a way to make informed decisions about the tradeoff between the benefit of requirements changes against the potential defects that the changes may produce at the end. An estimated defect count (and maybe the severity of the defect) could provide a better foresight required to plan more effectively.

2.2 Defect Prediction

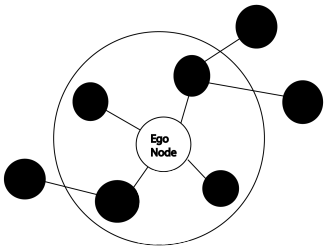
A defect is a common terminology for a fault in a program [17]. Common terminologies such as defects or bugs refer to faults in a program that lead to failures [17]. For example, misunderstanding requirements specifications introduces faults into a program, but these faults are only observed when a failure occurs.

Defect prediction is a research area that aims to answer the following questions [20]: 1) Find metrics that are available in the early phase of software development that are good defect predictors; 2) Develop models that can be used for defect prediction; 3) Evaluate the accuracy of the model; 4) Calculate the cost of utilizing the model in a software organization. Defect prediction requires various kinds of knowledge repositories that can be easily mined for obtaining the status of the project at a given point in time.People have used faults databases, code repositories and feature requests databases for their defect prediction analysis based on code [20]. Manual inspection of finding errors in code is currently at around 60% [20].

Nagappan and Ball [21] and Munson and Elbaum [20] used code churn to predict the defect density in software systems. There are other predictors such as debug churn [16], code churn analysis using neutral networks [15], file status [22] and network analysis [27]. Currently, the estimation for these code-based predictors for defects has reported accuracies of up to 70% to 89% [21, 20], although it needs further evidences.

Our aim is to find out whether there are attributes present in the requirements specification, particularly stories, that may be used to predict the occurrence of defects before code has been developed. Our aim is not about deriving a defect prediction model that can predict better than code-based models. Inherently, prediction becomes much more concrete once we have code. Getting an indication of potential defects using requirements specifications allows a development team to focus effort and collaboration on avoiding po-tential failures. As far as we know, defect prediction using requirements specification attributes available

Figure 1:
The ego network consists of the nodes immediately connecting the ego nodes. The global network refers to all nodes.



in stories has not yet been attempted.

2.3 Network Analysis

The purpose of the network analysis in our research is to provide explanatory patterns as to how attributes are related to each other. The motivation behind network analysis is to understand the structure and evolution of the relationship between entities. Many natural networks have a few nodes that have many more connections than the average node has. Therefore, most real world networks emerge using the Power Law [1]. It means the community has a strongly connected core. Network analysis shows if there are multiple communities within a network and determines how segregated or unified group members are. It can also show how the actors in the network influence each other. For software engineering application of network analysis, refer to [27, **?**, **?**] In network analysis, an ego network is concerned with its immediate neighbors. Each node in the network has an ego network. A node is often called "ego" in network analysis [24]. The global network looks at the entire nodes.

3 Case Study

In this section, we present the data that we used for our case study. We used the IBM Jazz development project data for our analysis [9, 26, 25]. The repository has many variables that can be explored, which provides an interesting case study for data mining. As mentioned, data mining is an exploratory analysis. Our aim was to find out whether any interesting structures or patterns exist in the given data that may provide insights into the relationship between requirements (stories) and defects and whether such information is embedded in the repository produced by the Jazz team.

Jazz is a development environment produced by IBM to support collaboration on software development. The Jazz system includes an integrated programming environment as well as communication and project management tools [9]. The Jazz development team used Jazz as their collaboration and programming environment. As a result, a substantial repository of development data was created. We data mined this Jazz development repository for our study. The data we extracted includes data from December 8, 2006 to June 26, 2008. 151 contributors (user accounts) exist in our data, but only 93 unique users were relevant to our study because others did not participate in the project during our time frame in terms of work items that can be traced from stories. The teams are distributed over 16 different sites, including the United States, Canada and Europe. Seven of these sites were active in the development and testing. The development project had 90 components.

The team used the "Eclipse Way" methodology for their development. Their development methodology is reported in [9, 26, 25]. Each iteration consists of six weeks. A project management committee sets up a goal for each iteration and breaks down the goal into features, called work items. A work item represents an assignable and traceable task that can be categorized into different subtypes, such as defect work items, story work items, enhancement work items or retrospective work items. Each work item was then assigned to a development team. In addition, each work item had a specific owner who tracked the work item from the beginning to the end. However, many people could contribute to the work item, such as contributing to its implementation, joining discussions and subscribing to the work item to keep an eye on its progress.

One of these work item types is Story. This is the work item that specifies a goal (or a requirement). Another type of work item is En-

hancement. In this paper, when we mention a story, we mean both Story and Enhancement work items. If a defect is found, a work item of type Defect is created. Here is an example story.

> Provide an integration option for the Visual Studio client: Our current option for writing an SCM client is to use a combination of server REST and command line tools. It turns out that the command line (CLI) is both too slow and far from feature complete. In addition, calling the CLI and parsing stdout isn't an option for providing a rich integration into another IDE. This story is about enhancing the client side integration to allow:
>
> - a rich and feature complete visual studio client or any other client written in Java or another language.
> - an integration which is as fast as the current RTC UI client.
> - an integration architecture which ensures that feature X added in the RTC UI can easily be leveraged in client Y.
> - a CLI with feature parity with the RTC UI for improved command line usage and scripting.

In the case of a defect, the team goes through a short discussion phase to make sure the work item is indeed a defect before it is assigned to a team member who will fix the defect. If the work item is vague, the team members can ask questions for clarification. The Jazz system allows work items to be linked to each other if they are conceptually related. For instance, if a defect can be traced to a requirement, these two work items can be linked. We use these links to determine if a defect is a requirements-related defect. i.e., for each individual requirement (work item), we measure the number of linked defects.

In our research with Jazz development data, defect refers to any work items that are labeled as defects in IBM Jazzs data by the Jazz developers. Requirements-related defects are defect work items that can be traced to a story work item. We noticed that 94% of the defects can be traced to one or more of the story work items. The rest of the defects appeared without any relationship to stories. In the Jazz repository, a story work item can be and often is linked to several defect work items. The process of linking can lead to very complex links of defect fixes. The Jazz system maintains these links as a part of the tools functionality. What we are interested in is whether there are some overarching structures or patterns that arise from these individual links between work items. Developers report the relationship between work items because they think that knowledge about the existence of other work items may help solve the problem. In other words, the links between story work items and defect work items are based on human understanding of the problem, not an automatic association generated by the code. In this sense, the linking between defects and story work items is different from linking of the change sets to the bug reports or linking of bug reports to failed tests, which are automatically generated by Jazz through code submissions. Because the linking of stories to defect work items represents a human understanding of the problem, there is no absolutely right or wrong way to interpret what the linking may mean other than that the developers who were assigned to the task thought that they were relevant and important. When individual developers make these links to help solve ones own problem, it may eventually emerge into a network of linked work items that may together have greater meaning. In addition, there is no valid instrument to check whether these links are right or wrong. These networks are different from networks that you would generate from code dependencies or other code-based metrics, because the links are inherently based on qualitative reasoning that came from each developer.

The entire data, including work items, any file attachments, code base, all of their history, is about 21.3 GB. The code base consists of 2.34 GB of data. Our extracted data includes data from December 8, 2006 to June 26, 2008 with a total of 2,860 story and enhancement work items and their 215,099 related defect work items.

4 Research Design

The literature survey on code-based defect predictions suggests that code churn is a good defect predictor at the coding level with accuracy of about 70% to 89% [21, 20]. However, we want to find out if there are attributes that are available at the requirements stage that can be used to predict defects. We modeled our research similar to [21, 27]. However, we looked specifically at non-code attributes and considered the types of information that are available at the requirements level. We used the Jazz web interface as well as Jazz Team Concert (which are tools within Jazz) to extract data relevant for our analysis. Then we built a script that calculates the metrics used in our investigation.

We present all the variables that we explored whether they had correlations or not. Data mining is about exploration; thus, negative correlations are as interesting as positive correlations. We categorized our variables for the purpose of obtaining explanatory powers, but the results were originally obtained from an exploratory process.

For the purpose of presentation, we categorized the attributes into point and aggregate variables. The point variables are attributes whose values are determined in the beginning of the story specification and can be obtained from a single story. For example, a time estimation to implement a story will be done in the beginning and is available from a single story. The team may decide to change the time estimate at a later time if new information becomes available and if the story is not implemented.

On the contrary, aggregate variables are values that are accumulated across multiple work items. Therefore, it is not possible to obtain these values using only one value from one story. For example, the number of related stories may change over time as new requirements are added to the project. Therefore, a story with only one related story may have two or more related stories at a later point in time as additional requirements are added to the project.

In our study, the dependent variable is the number of defects and the independent variables are the point/aggregate variables that we are going to present below. We are trying to map one-to-one relationships between each of these variables to the number of defects. In other words, we are measuring whether these independent variables have an influence over our dependent variable, the number of defects.

4.1 Point Variables

For the point variables, we have the following attributes for each story:

1. Time Estimates: The time estimates are developers' estimation of how long a story will take him/her to implement. Not all of the stories had time estimate information available. Our assumption for data mining the time estimates is that some tacit knowledge about the difficulty of the implementation may be reflected in the time estimates. For example, stories that are expected to take longer to implement may be more difficult (e.g. more complex or simply more comprehensive), thus could be prone to more defects. Since we do not have code complexity information at the requirements stage (as we stated that we designed our research to only consider information available prior to coding), time estimates may provide an alternate way of predicting the developers projection of the possible code complexity.

2. Priority: Priority is measured as "Unassigned","Low", "Medium" or "High". This measurement is the stakeholders view of when the story should be implemented in comparison to other stories. A story that is assigned a high priority should be implemented before a story that is assigned a low priority.

3. Ownership: Each work item usually has an owner who makes sure that the work item is finished. This is usually the person who finally signs off the work item as resolved, although not always. We can interpret the correlation to mean that someone who owns many work items may have better knowledge about the projects health because he/she has an understanding of how different work items are integrated together. On the contrary, a person may

get overwhelmed by many work items and then make mistakes. Either a positive correlation or a negative correlation would confirm that prediction can be made based on this variable. No correlation means the ownership is a poor predictor for the possible number of defects.

4.2 Aggregate Variables

1. Number of Indirect Stakeholders for the Story: We define a stakeholder as any user who had an account in the Jazz project repository. This includes developers, user interface designers, requirements analysts, testers, project managers, etc. We define indirect stakeholders as people who report defects or additional related enhancements but have not been involved in the initial definition of the story. If there is a positive correlation, it suggests that defects arise due to not recognizing the true extent of the indirect stakeholders. A negative correlation would suggest that having more indirect stakeholders actually leads to less number of defects. No correlation would mean that this variable is not a good indicator for obtaining defect predictions.

2. Number of Related Stories Based On Shared Defects: In Jazz data, we identified those defect work items that are linked to two or more story work items. We interpreted these defects to mean that there were unexpected interactions between requirements. If there is a positive correlation, there is strong support that defects arise due to unexpected interactions between requirements or a larger network of interactions between work items. If defect fixes require knowledge about other story work items, the person who implements the fixes needs to consult with other team members. The need to be aware of many work items could mean that there is more potential to change the behavior of requirements that someone else wrote in an unintended way. If there is no correlation, it suggests that story interactions do not provide a predictable trend that can be used for defect prediction.

3. Story Creation Time: We decided to test whether introducing a story at a later time (after some iterative code implementations) leads to more defects. We measured the time when a story was introduced to the project and measured the subsequent number of related defects. While introducing new stories later in the development stage does not directly measure requirements change, it does represent a lack of such requirements information before they were introduced. Since some implementation had already happened before these new stories are introduced to the team, the developers may have designed code without the knowledge that such requirements may be coming up in the future.

4.3 Null Hypothesis

The null hypothesis states that there is no correlation between any of the six attributes suggested above and requirements-related defects. Literature suggests that a p-value below 0.05 is considered to have high statistical significance [8]. If the statistical significance is below 5% [8], we are going to suggest that the alternative hypothesis, which is that there is a correlation between the selected attribute and the occurrence of requirements-related defects, is supported. Based on our data, we cannot absolutely prove the relationship between the attributes and the occurrence of requirements-related defects, but it may suggest that there may be a strong relationship.

4.4 Network Analysis

For part two of our analyses, we look at the network patterns in our data based on the attributes that show the highest correlations. These associations between work items and people are drawn up into a network graph, where the nodes represent people and edges represent the related defects on the stories they worked on. The purpose of the network analysis is to provide explanatory patterns as to how attributes are related to each other.

- Size: The size is the number of nodes in the ego network. It includes nodes that are one step away from the node, n_i.

- Two-step reach: The two-step reach measures the percentage of nodes that can be reached in two directed steps from the node.
- Brokerage: The brokerage is the number of times the node appears in other nodes connection paths. The brokerage value would be high for a node that is connected to many nodes, because it can play the role of a broker in connecting two unconnected pairs.
- Effective Size: The effective size is measured by the number of its neighbors minus the average number of directed ties between these nodes. Lets suppose there are three nodes, n_1, n_2, n_3, and n_2 and n_3 have a directed connection and n1 has a directed connection to n_2. The effective size for n_1 is 2-1=1.
- Degree Centrality: The degree centrality measures the number of dependencies for each stakeholder. We measured In-Degree, Out-Degree and InOut-Degree of a node. In-Degree measures the number of incoming connections to the node. Out-Degree measures the number of outgoing connections to the other nodes. The InOut-Degree is the sum of In-Degree and Out-Degree.
- Betweenness Centrality: The betweenness centrality measures how many times the node appears in the other nodes shortest paths calculations. First, we need to calculate the probability index of communication paths between two nodes. If the network offers more than one shortest paths between two nodes, n_j and n_k, then all of them have the same probability to be chosen. Suppose one of these shortest paths contain the node, n_i and let $g_{jk}(n_i)$ be the number of shortest paths linking n_j and n_k, then the probability that n_i is beween n_j and n_k is $g_{jk}(c_i)/g_{jk}$. Then the betweenness centrality is measured using the following formula:

  $C_B(c_i) = \frac{\sum_{j<k} g_{jk}(n_i)/g_{jk}}{(g-1)(g-2)}$, where

  C_B is the degree centrality and g is the number of nodes in the network.

Finally, instead of categorizing developers individually, we put them into teams. There are 90 project components. Each component is assigned to a team. Each team has its own stream. A stream is a workspace with a separate branch in the source repository. Each team commits their code into their stream only. We wanted to see if dependent defects are found by the members inside the team or members outside of the team.

- Percentage of People Outside of the Team: In the ego network, we want to find out how many of these connections are with people outside of their team. If this value is high, it denotes that requirements-related defects are mostly found when there is an interaction with outside teams.
- Associated Team Areas: A person is assigned to many team areas or none at all depending on their job description. We want to know if a person assigned to many teams and overseeing many projects could detect more requirements-related defects.

5 Result

In this section, we describe the results of the case study performed on Jazz development data. Section 5.1 presents the correlation analysis between the requirements attributes and the code attributes. Section 5.2 presents the regression analysis and section 5.3 presents the data splitting in order to measure the ability to predict system defect density.

5.1 Correlation Analysis

We used Pearson correlation coefficient [4] to verify the correlation between the specified attributes and defect occurrences. The closer a correlation value is to -1 or +1, the higher the correlation between the two attributes: +1 means they are perfectly positive correlated and -1 means they are perfectly negative correlated. A value of 0 indicates that the two measures are uncorrelated. The Pearson correlation values are shown in Table 1. We based our threshold on Coltons rule of thumb for interpreting the size of correlations, which is follows [6]:

> Correlations from 0 to 0.25(or -0.25) indicate little or no relationship; those from .025 to .50 (or -0.25 to -0.5) indicate a fair degree of relationship; those from 0.50 to 0.75 (or -0.50 to -0.75) a moderate to good relationship; and those greater than 0.75 (or -0.75) a very good to excellent relationship.

While the correlation coefficient measures the strength of the relationship, the significance measures the probability of an event occurring by chance only. The significance is measured using a probability level denoted as p. A smaller p means that the result is unlikely to be caused by pure chance. As defined in the research design section, a p value that is smaller than 5% is considered significant for our research and we will reject the null hypothesis [3]. The result is presented in Table 1. To summarize, we observe that there is a strong correlation relationship between the number of defects and the

- Number of Indirect stakeholders
- Number of Related Stories

The significance value for Story Creation Time is higher than our threshold of 0.05; therefore, we cannot make any general conclusion about this variable and it is eliminated from the candidate variable. However, the other variables provide high significance values. In terms of Time Estimates, Priority and Ownership, our data shows that there is clearly no relationship between these variables and the defects count. They all show high statistical significance to support our observation. See table 1. Based on our result, the two variables in the aggregate variables category, the Number of Indirect Stakeholders and the Number of Related Stories, both show high correlations with the number of defects. The point variables all show no correlations with the number of defects.

5.2 Regression Analysis

The purpose of a regression analysis is to develop an equation of a line that best fits most of the data points. The Standard Error of Estimate is calculated to check for the discrepancy between the data and the regression model [8]. This is the distance between the actual data points and the regression line.

At this point, we have narrowed down our analysis to the two variables that have shown high correlation coefficients: the Number of Indirect Stakeholders and the Number of Related Stories. Therefore, we performed the regression analysis for the two attributes that show statistical significance and strong correlation. The model generated from the regression analysis and Standard Error of Estimate provides a way to identify how closely we can fit our data on a line. In our study, the dependent variable is the number of defects and the independent variables are the number of indirect stakeholders and the number of related stories, each measured separately. In Table 2, x represents the number of defects and y represents the attribute being examined.

Our analysis shows that a power regression and a polynomial regression fit our data best as seen in Table 2. The number of indirect stakeholders has a good regression model with a relatively small Standard Error of Estimate. The correlation analysis and the regression analysis both confirm that there are indeed positive trends in the relationship between these two variables and the number of defects that are beyond a random occurrence of events. MMRE of <0.25 and PRED (0.3) of >0.75 are considered to be highly acceptable model of accuracy [23]. The MMRE and PRED(0.3) in our analyses are both in the range of highly acceptable numbers, which suggests that our regression model can fit our data with good accuracy.

5.3 Cross Validation

Based on our analyses, the variables that show consistently high correlation with the defects are the Number of Indirect Stakeholders and the Number of Related Stories. Therefore, we used the data splitting technique on the Number of Indirect Stakeholders and the Number of Related Stories. Data splitting [21] is a technique to independently assess the ability to predict from a population sample. We randomly select two thirds of the stories (1906 stories) from a population to build the prediction model and then use the remaining one third (954 sto-

Table 1: Correlation coefficient between the specified story attributes and the number of defects

Attributes	Pearson Coeffc.(r)	R^2	Signific.	Mean	Variance	Std. Dev.	Std.Err.
Time Estimates	-0.0222	0.001	$<0.01^a$	1119.83	1,463,879.2	1209.91	110.91
Priority	0.0602	0.004	$<0.01^a$	2.04	0.07	0.27	<0.01
Ownership	-0.0316	0.001	0.04^a	772.86	1,302,499.28	1141.27	21.31
Number of Indirect Stakeholders	0.9048	0.819	$<0.01^a$	5.36	37.56	6.13	0.11
Number of Related Stories	0.7591	0.576	$<0.01^a$	17.01	1417.89	37.6	0.70
Story Creation Time	0.0144	0.001	0.22	332.41	22,465.18	149.88	2.8

Table 2: Regression Analysis

Attributes	Regression Model	Standard Error of Estimate	MMRE	PRED (0.3)
Number of Indirect Stakeholders	$y = \frac{x^{1.56}}{3.55}$	0.30	0.27	0.76
Number of Related Stories	$y = (0.009x + 1.244)^6$	0.37	0.20	0.87

Table 3: Data Splitting Regression and Correlation Analysis for Number of Indirect Stakeholders

Trial#	R^2	Significance
Random 1	0.8248	$<0.01^a$
Random 2	0.8102	$<0.01^a$
Random 3	0.8199	$<0.01^a$

Table 4: Data Splitting Regression and Correlation Analysis for Number of Related Stories

Trial#	R^2	Significance
Random 1	0.5432	$<0.01^a$
Random 2	0.5628	$<0.01^a$
Random 3	0.6294	$<0.01^a$

Table 5: Correlation Analysis on the Network Measures for Stakeholders and Related Stories

Measures	Pearson Coefficient(r)	Sig. (p)
Size	0.9182	$<0.01^a$
Two-Step Reach	0.9367	$<0.01^a$
Brokerage	0.9096	$<0.01^a$
In Degree Cent.	0.4356	$<0.01^a$
Out Degree Cent.	0.26625	$<0.01^a$
InOut Degree C.	0.2617	$<0.01^a$
Betweenness	0.5130	$<0.01^a$
Effective Size	0.9182	$<0.01^a$
%Outside Team	0.6775	$<0.01^a$
Associated Team	0.5475	$<0.01^a$

[a]Significant at alpha=0.05 level

ries) to verify the prediction accuracy. Then we find out whether our variables still hold the prediction ability even with different training and evaluation values.

Using the regression equation, we estimate the defect density for the remaining third of the stories. Then we compare the estimated values with the actual values for the remaining one third of the stories that were used for the evaluation. We ran the correlation analysis between the estimated and actual values. A high positive correlation coefficient means there is a positive relationship in the attributes being measured and the estimated defect density.

All trials show consistent positive correlation and statistical significance as shown in Table 3 and 4. The magnitude of the correlation provides the sensitivity of the predictions. A higher correlation means the prediction has a higher sensitivity. The result shows that the Number of Indirect Stakeholders is a very good predictor of defect density and the Number of Related Stories is a moderate to good predictor.

5.4 Networks of People and Stories

The result so far suggests that the two variables, the number of indirect stakeholders and the number of related stories, can predict the number of defects very well. From a statistical perspective, it suggests that these two variables are good predictors of the number of defects. The next question is how these two variables relate to the number of defects and provide some characteristics about their relationships. To evaluate the nature of their relationships between the stakeholders and stories, we analyzed how each person (stakeholder) is linked to stories. Two people are linked on a network if they both share some work for the same story. Table 5 shows the statistical analysis of how these values relate in terms of the network measures, which were defined in Section 4.

As shown in Table 5, size, two-step reach, brokerage and effective size are showing high correlation. It means the person who is linked with many people also has many related stories, which is not a surprising observation. Table 5 also shows that the betweenness measure shows very high correlation, but the degree centrality measure does not. What this means is that the person who can explain the most number of related stories using the smallest number of related stories (in other words, shortest path between stories networks) is the most important person, not the person who is linked to the most number of stories. Finally, the Percentage of People Outside of the Team and Associated Team Area show moderately positive correlations. It means that there are some reasonable trends that defects are discovered by people outside of the immediate core team and more likely to be detected by people who are working on multiple teams.

6 Discussion

In this section, we discuss the results of our analyses. Our aim of the research project was to data mine a large development project to find out whether there are requirements-related attributes that can predict the number of defects. We have shown through our analyses that the Number of Indirect Stakeholders and the Number of Related Stories can predict the trends in the number of defects. In addition, we have shown that stakeholders and stories are related in terms of size, two-step reach, brokerage, effective size and betweenness centrality. In addition, we have also discovered that people outside of the core team may find more defects as well as people who participate in the development of more than one component.

The other attributes, such as time estimation, priority and ownership, did not show that they were good predictors for the number of defects. The story creation time did not meet the significance threshold; thus, this value is inconclusive. However, what is important from our findings is that trends for the number of defects can be predicted from requirements attributes. In this section, we are going to discuss the implication of our findings in terms of what this could mean for defect prevention.

6.1 Indirect Stakeholders and Related Stories

The two attributes that show very high correlations with the number of defects are the Number of Indirect Stakeholders and the Number of Related Stories. Both of these numbers are quantitative; therefore, they can be measured at any given point in time. However, both of these are aggregate variables, which mean these values cannot be measured alone or only at one point in time. Rather, they grow and change as more stories and people are added to the project.

The findings can be interpreted in many ways. It is unlikely to predict who or how many people will report the defects at the time of the specification. However, if more of different people report the defects as time progresses for a story, the chance of having more defects being reported in the future will only likely to increase. Our result suggests that the interaction of people really matter in explaining the number of defects. The result is suggesting that a tool that can make these increasing numbers of indirect stakeholders and related stories obvious may be important in detecting what are the defect-prone areas of the software development project. There is no doubt that people are important in requirements engineering. Cheng and Atlee states that "successful RE involves understanding the needs of users, customers and other stakeholders; understanding the context in which the to-be-developed software will be used; negotiating and documenting stakeholders requirements and validating that the documented requirements match the negotiated requirements" [5]. This definition emphasizes that there are a lot of human social activities involved in RE, such as identifying the needs and negotiating for agreements.

Our analysis suggests that we need more studies in identifying and understanding stakeholders better if we want to prevent defects. We could possibly interpret our result to mean that if more people are involved in a feature, more careful the team will have to look at the requirements and the more collaboration/communication should be spent on it.

6.2 Network Analysis

The second part of the analyses is to find out how these two variables, the Number of Indirect Stakeholders and the Number of Related Stories relate to each other. To understand the characteristics of their relationship, we measured 10 network attributes. As shown in Table 5, four attributes show very high positive correlations and three attributes show moderately positive correlations. Other attributes do not show any correlation.

First, the ego network values all show very high correlation values. This suggests that ones own knowledge about the people who are working on the related stories is very important in predicting the number of defects. If a person is working on a story that associates a lot of people, then it is likely that this story is also related to many defects. It shows the need for developers to find people who are working on similar or related stories in the team as early in the development as possible.

A person may be assigned to multiple team areas depending on their functional specializations or their breadth of knowledge. Our result suggests that most of the defects are discovered by people who did not belong to the same team as the person who created the story. A moderately positive trend shows that defects are sometimes discovered by people outside of the team. Finally, we measured the total number of teams represented per stories. Our result moderately supports the proposal that a component with people from many different teams do end up with more defects.

6.3 Predictability

The result does confirm our hypothesis. There are requirements attributes that have strong correlations with the number of defects. Even at the requirements stage, the number of indirect stakeholders plays a crucial role in the defects count. Making sure that everyone knows how changes will impact their part of the work may be important. A tool that can help show how ones story is related to other peoples story may be helpful in discovering these unexpected defects before they are implemented into the system.

7 Threats to Validity

In this section, we discuss the validity of our findings with respect to internal and external validity.

For conclusion validity, we have shown that our result has very high statistical correlations. Each attribute is measured independently from other attributes. We are only working with one project, so there is no risk of random heterogeneity of subjects. We believe that the data is sufficiently large enough to warrant statistical significance of our result. We also assume that everyone in the team consistently used the Jazz repository to communicate and record their development progress.

For construct validity, we have made effort to discuss the limitations of our attributes and any assumptions we made to obtain the measurement. We measured multiple attributes in both point and aggregate categories. There is no participants bias toward the research, because the data represents their normal development progress, rather than a response to a research study. The selection of measurements was based on a literature survey as well as on what was available in the data set.

The Jazz development team is using Jazz in order to build Jazz. Therefore, depending on when a feature was developed, some data might not be available. Before the concept of Story work item was introduced, Jazz was already keeping track of some defects. We ignored these defects from our analysis if they did not relate back to one of the Stories or Enhancements. However, 94% of the defects are accounted for through the relationship between Stories and Enhancements. In terms of the population selection, our data had 93 unique contributors, which is large enough to account for any natural variation in human performance.

One of the threats to internal validity is the interpretation of the causal influence. Based on our analysis, we can state that there is a strong correlation between the Number of Indirect Stakeholders and the Number of Related Stories to the number of defects, but we cannot suggest that these attributes can cause defects. In addition, defect prediction using knowledge, code, or defect repository does not show any social dynamics that may exist in the team under study.

For the external validity, the study was performed on a single, large development project. Therefore, there is a risk of single group threats, which applies when the result looks at a single group. More empirical studies are needed to generalize our result. The size of the code base and the development organization are at a much larger scale than many commercial products. Therefore, it may be that smaller projects may not show similar trends. We also cannot generalize our results to software using other languages or platforms. More replication studies are needed for other types of development projects.

Our data spans almost 1.5 years of development work. The time scale is large enough to compensate for any unusual events that may skew the result. If there were any special events that may have influenced the result, it is unlikely to have contributed to a significant deviation of our data.

8 Conclusion

For our evaluation, we divided the story attributes into two categories: point and aggregate. The point variables include the time estimate, priority and ownership. For aggregate variables, the number of indirect stakeholders, related stories and story creation time are analyzed. Our analysis shows that the number of indirect stakeholders and the number of related stories are good predictors for the number of defects that can be obtained from the requirements specification, such as story, for estimating the number of defects. Our research was exploratory in nature. Our intention was to identify the requirements attributes that can also be used as a defect predictor, because being able to predict, even with a limited accuracy rate, using the requirements specification may provide the team with a better plan of attack for their implementation decisions.

A correlation analysis cannot define the causality of the relationship without further experiments. Therefore, we invite other researchers to validate our results with additional project data. Our empirical study can contribute towards a better understanding about

collecting measurable attributes for controlling and monitoring requirements-related defects. We also discovered that data collected by tools, such as Jazz, can provide a good basis to point to potential requirements defects and can rationalize decisions on where to spend inspection and testing effort.

Acknowledgements

The study is supported by NSERC PGS, AIF/iCore and IBM JAZZ grant.

About the Author

Shelly Park is currently pursuing her Ph.D. in Computer Science at the University of Calgary. She was an instructor for human-aspects of software engineering course at the University of Calgary. Her research interests include agile software engineering and human aspects of software engineering.

Dr. Frank Maurer is a Full Professor at the University of Calgary. His research interests are agile software methodologies, engineering digital table applications, executable acceptance test driven development, integrating agile methods and interaction design, framework and API usability and tools for agile teams. He is a member of the Agile Alliance, a Certified Scrum Master, a founding member of the Canadian Agile Network (CAN) Le Rseau Agile Canadien (RAC) and Associate Editor of IEEE Software responsible for the Process and Practices area.

Dr. Armin Eberlein is currently an Associate Professor and the Head of the Computer Engineering Department at the American University of Sharjah in the United Arab Emirates. His research interests focus on the improvement of requirements engineering practices and techniques. He worked previously as a hardware and software developer at Siemens in Munich, Germany, and has consulted for various companies in Germany, the UK, and Canada.

Dr. Tak-Shing Fung is currently a statistical consultant at Research Consulting Services at the University of Calgary. He received Ph.D. in Statistics at the University of Calgary. He worked in the industry for many years as a statistics consultant. He also teaches statistics at the University of Calgary.

References

[1] A. Barabasi. *Linked: How Everything is Connected to Everything Else*. Perseus Publishing, Cambridge, MA,, 2002.

[2] B. Boehm and V. Basili. Software defect reduction top 10 list. *IEEE Computer*, 34(1):2–6, January 2001.

[3] B.Winer. *Statistical Principles in Experimental Design*. McGraw-Hill, 1971.

[4] CHAOS. Standish group report. *Standish*, .:., 1994.

[5] B. Cheng and J. Atlee. Research directions in requirements engineering. *Future of Software Engineering, IEEE Computer Society*, .(May):., 2007.

[6] T. Colton. *Statistics in Medicine*. Little, Brown, 1974.

[7] R. Fairley. *Software Engineering Concepts*. McGraw-Hill, New York, 1985.

[8] D. Freedman, R. Pisani, and R. Purves. *Statistics, 3rd Ed.* Norton & Company, 1998.

[9] R. Frost. Jazz and eclipse way of collaboration. *IEEE Software*, 24(6):581–603, 2007.

[10] D. Hand. Statistics and data mining: Intersection disciplines. *ACM SIGKDD Exploration*, 1(1):16–19, June 1999.

[11] I. Hooks and K. Farry. *Customer-centered products: Creating successful products through smart requirements management*. American Management Association, New York, NY, 2001.

[12] European Software Institute. European user survey analysis. *Report USV EUR ESPITI Project*, 2.1:., 1996.

[13] T. Javed, M. Maqsood, and Q. Durrani. A study to investigate the impact of requirements instability on software defects.

ACM Software Engineering Notes, 29(4):., May 2004.

[14] C. Jones. *Software Quality: Analysis and Guidelines for Success*. International Thomson Computer Press, 1997.

[15] N. Karunanithi. A neural network approach for software reliability growth modeling in the presence of code churn. In *Proc. of International Symposium on Software Reliability Engineering*, pages 310–317, 1993.

[16] T. Khoshgoftaar, E. Allen, N. Goel, A. Nandi, and J. McMullan. Detection of software modules with high debug code churn in a very large legacy system. In *Proc. of International Symposium on Software Reliability Engineering*, pages 364–371, 1996.

[17] A. Mathur. *Foundations of Software Testing*. Pearson Education, 2002.

[18] G. Mogyorodi. What is requirements-based testing? *CROSS TALK, The Journal of Defense Software Engineering*, .:., March 2003.

[19] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and point code attributes for defect prediction. In *Proc. of 30th International Conference on Software Engineering,Leipzig, Germany*, pages 191–190, 2008.

[20] J. Munson and S. Elbaum. Code churn: A measure for estimating the impact of code change. In *Proc. of IEEE International Conference on Software Maintenance,r*, pages 24–31, 1998.

[21] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proc. of the 27th International Conference on Software Engineering, St. Louis, USA*, pages 284–292, 2005.

[22] T.J. Ostrand, E.J. Weyuker, and E.M. Bell. Where the bugs are. In *Proc. of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 86–96, 2004.

[23] D. Port and M. Korte. Comparative studies of the model evaluation criterions mrre and pred in software cost estimation research. In *Proc. of International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Kaiserlautern, Germany, 2008.

[24] G. Sabidusi. The centrality index of a graph. *Psychometrika*, 31:581–603, 1966.

[25] L. t. Cheng, C. de Souza, S. Hupfer, J. Patterson, and S. Ross. Building collaboration into ides. *Queue*, 1(9):40–50, 2003.

[26] L. t. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up eclipse wit collaborative tools. In *OOPSLA workshop on eclipse technology eXchange, Proc. of the 2003 OOPSLA workshop on eclipse technology eXchange,*, pages 45–49, Anaheim, California,, 2003.

[27] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *Proc. of 30th International Conference on Software Engineering, Leipzig, Germany,*, pages 531–540, 2008.

[28] D. Zowghi and N. Nurmuliani. A study of the impact of requirements volatility on software project performance. In *In Proc of the 9th Asia-Pacific Software engineering Conference, Washington, DC, USA*. IEEE Computer Society, 2002.