Role of Domain Ignorance in Software Development

by

Gaurav Mehrotra

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Several have reported observations that sometimes ignorance of the domain in a software development project is useful for promoting the elicitation of tacit assumptions and out-of-the-box ideas. This thesis reports work putting the observation to two empirical tests. First, a survey was conducted among software development managers of varying experience to determine what software development activities they thought were at least helped by domain ignorance. Second, transcripts from fourteen interviews of presumably-domain-ignorant immigrants to new software development projects at one large company were examined to determine if the activities performed by those with the smoothest immigrations were activities that are at least helped by domain ignorance. The conclusions are that ignorance plays an important role in software development but there are a lot of other factors that influence immigration smoothness.

Acknowledgements

I would like to thank my supervisors, Prof. Daniel M. Berry and Prof. Chrysanne DiMarco for their patient guidance, insightful advices, and constant encouragement. I would also like to thank Prof. Mike Godfrey and Prof. Jo Atlee for being my thesis readers. I would like the thank the participants of my study for taking the time to fill out the online survey and helping me achieve this goal.

Finally, I am also thankful to Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard, Jacqueline P. de Vries, and their anonymous subjects for agreeing to share the data obtained from their research for comparison purposes.

Dedication

This thesis is dedicated to my parents who have supported me all the way since the beginning of my studies.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Ignorance of the domain is thought by some to be helpful in software development activities that require some critical, out-of-the-box thinking. An example is brainstorming for requirement-idea generation. Ignorance of a domain is believed to help one to avoid the domain's tacit assumptions and to think outside of the domain's box [3] [5, p. 18]. The right kind of ignorance helps expose all the tacit assumptions that someone experienced in the domain takes for granted. Who has not observed the phenomenon that the one who seems to know the least about a problem seems to come up with the best solutions in a brainstorming session? This observation leads to the suggestion that there may be some software development activities that are aided by some degree of ignorance. A *new hire in an organization* or an *immigrant to a project within an organization* could use her[1] ignorance about the domain of a system under development to perform tasks of the development that are helped or at least not hindered by her ignorance. *New hires* and *immigrants to a project* are collectively called "newbies" in this thesis.

A newbie is often clueless about the domain into which she is thrust upon arrival in the new environment and does not possess the skills necessary to be productive immediately. Either she is left to wander on her own and learn the tasks by trial and error or, in some cases, a senior member of her new team is assigned the job of training her. Despite her ignorance about the new domain, there are some software development activities that a newbie can perform effectively even better than a seasoned expert in the same domain. An expert takes many things as assumed or implied that an ignorant newbie would have to explicitly think about and evaluate from first principles.

[1]The gender of the first general individual in any discourse toggles with each chapter. The gender of the second general individual in any discourse is the opposite of that of the first.

The main goal of this work is to identify software development activities for which ignorance is required or at least helpful. The final result can be used as a checklist by any software development manager in order to assign tasks best suited for any newbie. By giving a newbie the right set of tasks from the beginning, she will be able to learn the domain on her own, with little assistance from project veterans, who often have their own time critical roles.

A review of the existing literature shows that that newbies who perform specific activities *seem* to have smoother start ups than newbies who perform other activities. Here "smoother" is used as in the vernacular to mean that start up occurred with much more positive than negative events so the newbie reports feeling good about the start up and regarding it as a success. Perhaps the activities done by newbies with smoother start ups require or are helped by some level of domain ignorance. The review showed also that some activities seem to yield better results when performed by a domain ignorant than when performed by a domain expert.

These two observations lead to the hypothesis that a newbie who starts with software development activities requiring some ignorance has a smoother start up than a newbie who starts with other activities. The main contribution of this work is the gathering of data that show the role of domain ignorance in various software development activities. The first step was to collect data about the importance of ignorance in various software development activities. The data were collected with the help of an online survey listing various software development activities. An invitation to participate in the survey was sent to people having significant experience managing software development. The next step was to look at histories of newbies' start ups to determine which roles had the smoothest start ups. Finally the two lists were compared to find any correlations.

Specifically, the research aimed to answer two important research questions:

- Are there software development activities that are helped by domain ignorance?
- What role does domain ignorance play in various software development activities?

Related work is discussed in Chapter 2. Chapter 3 presents the survey design. The results of the survey are discussed in Chapter 4. Chapter 5 describes some applications of the results. Chapter 6 concludes the thesis with a discussion of future work.

Chapter 2

Related Work

There is some previous literature highlighting the role of ignorance in software engineering activities. Berry described how ignorance helped him to come up with a requirements document for a networking firm while being totally ignorant of the domain [3]. P. Burkinshaw, an attendee of the Second NATO Conference on software engineering in Rome in 1969 [5, p. 18], said:"Get some intelligent ignoramus to read through your documentation and try the system; he will find many 'holes' where essential information has been omitted. Unfortunately intelligent people do not stay ignorant too long, so ignorance becomes a rather precious resource."

Carver et al. [6] studied the impact of educational background on requirement inspection. They observed that an inspector who had a background that was unrelated to computing was significantly more effective than others in identifying defects during a requirement inspection task. They observed also that an inspector who had a degree in computer science or software engineering was the least effective of all subjects. Also, an inspector with requirements analysis experience was significantly more effective in finding defects than those without such experience, but there was no marked statistical difference between subjects with industrial experience and those with only classroom experience. There is also a large amount of research on knowledge transfer, newbie integration in organizations, and information needs in software engineering maintenance tasks. However, it appears after a thorough literature search that there is no other study regarding the role that ignorance plays in various software development activities and how a newbie can use domain ignorance to his advantage.

Begel and Simon observed eight graduate students during their first months of work at Microsoft [1]. They found that most of the difficulties encountered by the new hires

came from their inexperience with a corporate environment. Many specialized programs have been developed to prepare computer science graduates for appropriate roles in the software development industry. However, employers recognize that students entering the workforce directly from university training often do not have the complete set of software development skills that they will need to be productive, especially in large software development companies [2]. Whereas a significant body of literature has documented the costs of bringing a software developer up to speed in a project or a new team, little has been written about the kinds of tasks a newbie in a team can be given in order to have a faster and smoother ramp up into the project landscape. The study for this thesis tries to discover the tasks best suited for a newbie in a team or project.

Sim and Holt interviewed four recently hired developers at a big software company and identified seven integration patterns [14]. Based on the patterns they drew several conclusions regarding the strengths and weakness of the naturalization process within an organization. Some of their important findings were: mentoring is an effective although inefficient way to train newbies, administrative and environmental issues were a major frustration during the immigration, and initial tasks assigned to newbies were often open-ended problems. Dagenais et al. [7] categorized the landscape features that newcomers needed to learn and also identified the obstacles and orientation aids encountered by newcomers in the context of various integration factors. Schein proposed that there were three main aspects to introducing newbies to organizations: function, hierarchy, and social networking [13]. Function represents the tasks and technical requirements of a position. Hierarchy is the organizational command structure, and social networking is the movement of the newcomer from the periphery of the network towards the centre as new personal connections are made.

In practice, little preparation is put into the training of a newbie, beyond assigning him to a senior developer who acts as a mentor. This mentor is expected to help a newbie become productive by providing to the newbie whatever guidance he needs. While mentoring has its merits, it tends to be an inefficient way to train a newbie, because it results in a net decrease in team productivity. DeMarco and Lister [8] observe, "We all know that a new employee is quite useless on day one or even worse than useless, since someone else's time is required to begin bringing the new person up to speed." Given these facts, a newbie is often left to wander on his own and discover the project landscape by trying out different sets of activities. There is no agreement within the industry on the types of activities a newbie should be given first in order to have a smooth transition into the project landscape.

Andrew Ko performed a field study of software developers at Microsoft in order to identify the information needs (i.e., what information developers look for and why they

look for it) of developers across different teams [10]. Software development is a tough beast to tackle. Designing software that meets specific goals requires the agreement of many parties. Products are often shipped loaded with bugs, and their developments get out of hand despite the developers' best efforts. Fred Brooks observed that adding a new person to a late project makes it even later [4]. In order to get the best out of the developers' efforts, it is essential to assign everyone to the task for which he is best suited. Also, the right mix of people to do an activity is necessary to get the best results. The contribution of this work is to help a software development manager to assign a newbie to tasks in which he will be effective immediately and to assign the right mix of people for each software development activity.

Kenzi, Soffer, and Hadar conducted an exploratory study of the perception of requirements analysts of the role of domain knowledge in requirements elicitation [9]. Their study identified both positive and negative effects of domain knowledge on requirements elicitation. Their conclusions suggest the possibility of forming requirements elicitiation teams with of analysts with different amounts of domain knowledge, that the role played by an analysts can depend on his or her domain knowledge, and that these different roles may create a useful synergy in identifying requirements. Additional research is needed to follow up on this exploratory study.

Chapter 3

Survey Design

A well-designed empirical study is the key to good and meaningful results. I decided to conduct a survey of software development managers in order to study the role of ignorance in software development activities. This chapter discusses the design of the survey.

The first step was to establish the goals of the survey. The next step was to plan the actual survey, keeping the goals in mind. The survey design was influenced by some external environmental variables. The environmental variables that were controlled in this study are:

- characteristics of participants,
- how often the participants will be surveyed, and
- sample size.

3.1 Survey Design

Some common methods for survey design are discussed below.

3.1.1 Cross-Sectional Survey

In a *cross-sectional* survey, data are collected at a single point in time. Thus, a cross-sectional survey represents the opinion of a group of people or an organization at a particular point in time. This kind of survey has advantages such as ease of administration, scope

for planning etc. On the other hand, since a cross-sectional survey represents a snapshot at a particular point in time, its information will quickly become outdated in a rapidly changing environment.

3.1.2 Longitudinal Survey

The other extreme of survey designs is the *longitudinal survey*, in which data are collected over a period of time. Longitudinal surveys can be further classified into three categories.

- **Trend Surveys**
  A *trend survey* involves surveying a particular group of people over a period of time. An example is a survey of all students in grade ten over a period of time. As the first group of tenth graders will be in eleventh grade one year later, in reality, we are sampling a different group of tenth graders each year.

- **Cohort Surveys**
  A group of people having some common traits or experiences within a defined period of time is a *cohort*. The actual people in a cohort might vary over time, but the original traits exhibited by the cohort remain the same. An example is the cohort of people enrolled in a training program aimed at educating individuals regarding the spread of a certain disease. A *cohort survey* aims to collect data from an individual once she has completed the particular training program.

- **Panel Surveys**
  A *panel survey* deals with collecting data from the same population over time. An example is a survey of participants before and after diet educating program, to monitor their attitudes towards diet. A sampling of the program's participants is selected, and they are surveyed throughout the duration of the program.

A cross-sectional survey design was chosen for this study, as the goal was to determine the role ignorance plays in various software development activities by surveying a group a people at a given point in time. The sole aim of this study was to learn the opinions of people having significant experience managing software development regarding the effect of domain ignorance on software development. This sort of opinion is likely to remain unchanged over a substantial period of time.

3.2 Survey Scale

A five-point ordinal scale was used to categorize the importance of or the effect of domain familiarity on any software development activity. An ordinal scale was implemented, as the goal of this research is to study the categorization of individual software development activities without measuring the relative ordering between them. The categories chosen were:

- **Required** - that domain ignorance or domain awareness is required in performing a software development activity.
- **Enhances** - that domain ignorance or domain awareness enhances the performance of a software development activity.
- **Neutral** - that domain ignorance or awareness is irrelevant in performing a software development activity.
- **Impedes** - that domain ignorance or domain awareness impedes the performance of a software development activity.
- **Prevents** - that domain ignorance or domain awareness prevents performing a software development activity.

The scale is thus a 5-point Likert scale [15].

In the scale, "required" and "prevents" are intended to be considered opposing each other, as are "enhances" and "impedes". "Neutral" is the middle. Together, "required" and "enhances" represent the positive, "helps" side of the scale while "impedes" and "prevents" represent the negative, "hinders" side.

The same 5-point scale was applied to each of domain awareness and domain ignorance. Although domain awareness is likely to be thought to enhance all activities, participants were still asked to apply the scale to it in order to make them think about domain ignorance in contrast with domain awareness. In the rest of this thesis, domain awareness and domain ignorance are collectively referred to as kinds of *domain familiarity*.

3.3 Survey Questions

The next step in the survey design was to compile a list of all software development activities that might be performed by any newbie in an organization. The survey would ask

about the importance or effects of domain familiarity for each activity. A comprehensive list of all such activities was taken from `http://www.opfro.org/`. A pilot survey was constructed and deployed in order to ensure that the survey questions were understandable and that the list of activities was complete. Similar activities were grouped together as a result of participant feedback from the pilot survey. The activities relating to a particular aspect of software development like testing, architectural activities etc. were grouped together under common headers as shown below. Similar grouping was done for other software development activities as well.

- **Architectural Activities**
  Designing and specifying software architecture
  Reviewing software architecture
- **Testing Tasks**
  Test planning
  Designing test cases
  Executing test cases
  Automating test cases

An activity was included or excluded from the survey solely on the basis of whether a newbie in an organization is likely to perform the activity. See the survey in Appendix A for the final list of activities chosen.

3.4 Sampling Participants

There are three sets of people that appear in any survey:

1. the set of people considered while making the sampling decision, known as the *target population*,
2. the set of people who are actually chosen out of the target population, known as *participants*, and
3. the set of people who actually respond out of all the participants, known as *respondents*.

Sampling precision is a measure of how representative respondents are of the target population.

Sampling approaches are concerned with ways of choosing participants for a survey. There are two main components in sampling: whom to survey and the number of participants. Some commonly used approaches for sampling participants are described below.

3.4.1 Probabilistic Sampling Methods

A *probabilistic sample* is selected objectively. An example is the use of a random number generator to select the winning lottery number. Statistical methods are available to calculate the probability that any participant has of being chosen. Some of the most commonly used probabilistic sampling methods are:

- **Simple Random Sampling** In *simple random sampling* each participant has an equal chance of being selected from the target population. The target population contains everyone who is eligible for the survey. An unbiased random selection of participants is important so that the sample can represent the entire target population in the long run. However, random sampling does not guarantee that any particular sample is a perfect representation of the target population. Random sampling has many advantages such as requiring a minimum of prior knowledge about the participants and is very simple to achieve.

- **Stratified Random Sampling** It is better to sample each stratum separately if the target population varies immensely. Stratification is the process of splitting the population into homogeneous groups before sampling. The strata should be mutually exclusive: every participant in the population must be assigned to only one stratum. The strata should also be collectively exhaustive: each participant should belong to one stratum or another. Once the stratification process is complete we select a given number or proportion of participants from each stratum to get the final sample. Therefore, a *stratified random sampling* is more representative of the population than a simple random sampling.

- **Simple Random Cluster Sampling** *Simple random cluster sampling* is used primarily for administrative convenience, and it does not enhance the sampling precision. It is done by dividing the target population into clusters and then doing a random sampling in each of these clusters. The clusters should be mutually exclusive and collectively exhaustive. Cluster sampling is similar to random sampling where

the sampling unit is a cluster. In a stratified random sampling, a random sample is drawn from each of the strata, while in a simple random cluster sampling, only the selected clusters are studied, and the remaining clusters are discarded. The main objective of simple random cluster sampling is to reduce costs. Cluster sampling is typically used when a researcher cannot get a complete list of the members of a population she wishes to study but can get a complete list of clusters of the population. It is used also when a random sample would produce a list of participants so widely scattered that surveying them would prove to be far too expensive, for example, people who live in different postal districts in Canada.

3.4.2 Non-Probabilistic Sampling Methods

A *non-probabilistic sampling* includes participants who are available and willing to take the survey. An immediate drawback of this sampling is that an unknown proportion of the entire population was not sampled. Thus, the sample may or may not be representative of the entire population. Therefore, the results of this type of research cannot be used in generalizations pertaining to the entire population without careful consideration of the contents of the sample. Some commonly used non-probabilistic sampling methods are:

- **Systematic Sampling** *Systematic sampling* involves selecting participants from an ordered list. The most common method is an equi-probability method in which every kth element in the list is selected, where k, also known as the skip factor, can be calculated using the formula, $k = N/n$, where N is the population size and n is the sample size.

- **Judgmental Sampling** In *judgemental sampling*, participants are selected to be part of the sample with a specific purpose in mind. With judgmental sampling, the researcher believes that some participants are more fit for the research than others. This type of sampling may lead to highly biased results.

- **Convenience Sampling** In *convenience sampling* participants are selected because they are easy to recruit. The researcher makes no attempt to insure that this sample is an accurate representation of some larger group or population. Participants are selected simply because they are available at a particular place at a particular time. An example is surveying people coming to refill gas in their cars. This technique is considered as easiest, cheapest, and least time consuming.

- **Snowball Sampling** *Snowball sampling* is done when there is a very small population size. Previously identified participants of a group identify other participants. The basis for identification can be any other kind of sampling, e.g., convenience, judgemental, systematic, simple random, stratified random, or simple random cluster sampling. Of course it must be that each identifying participant faithfully reproduces the base sampling method.

Since the participant pool for the survey should be limited to people having significant experience managing software development, a snowball sampling based on judgemental sampling was chosen for the research, in which the judgement targeted people likely to have had experience managing software development. The initial pool of participants consisted of people in academia and the software industry who were believed to have significant software development experience. Each was asked to pass the invitation on to similar people. The lack of control over the participants who were invited at the next and subsequent levels helped to bring some randomness to the participant pool.

Participants were contacted through email. The email invitations included a brief description of the research along with a link to the online survey. The initial response rate was quite low and not everyone invited to participate filled out the survey. The response rate got much better once a reminder was sent to the entire list of participants. In the end, a total of 40 respondents completed the survey.

3.5 Survey Administration

The survey was hosted by SurveyMonkey^{TM}. The survey was built at the SurveyMonkey site and the survey link was sent to all invitees through email along with a cover letter explaining the purpose of the research. See Appendix C for the detailed cover letter. A participant filled out the survey only if she wanted to and was free to withdraw from participation during any stage of the research.

The complete survey is in Appendix A. As required for any study involving human participants at University of Waterloo, this study was approved by the Office of Research Ethics. The completed ethics application is in Appendix B.

Chapter 4

Results and Data Analysis

This chapter discusses the results obtained in this study. A comparison of results with data from another similar study is performed towards the end.

4.1 Survey Respondents

A total of 40 respondents completed the online survey. Respondents came from different countries like India, United States, United Kingdom, Canada, and Israel and were fairly spread out over industry and research. Some additional facts regarding the respondent pool are:

- Type of organization:
  1. Commercial - 30
  2. Research - 10
- Experience in Software Development
  - Maximum experience in software development − 43 years
  - Minimum experience in software development − 1 year
  - Average experience in software development − 14.5 years

The overall distribution of the respondents' software development experience is shown in the graph in Figure 4.1. More than half the respondents had at least 10 years of experience in software development.

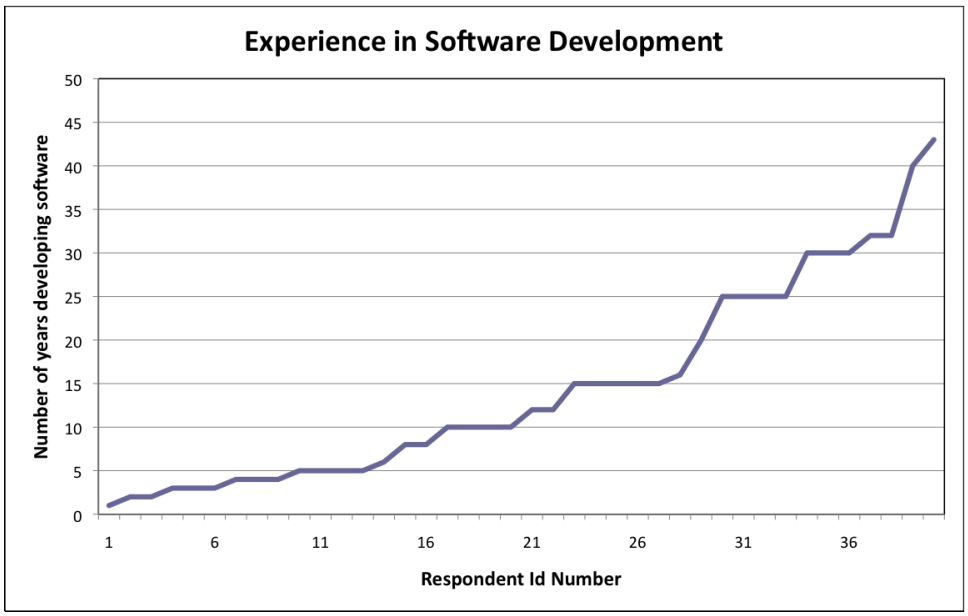

Figure 4.1: Experience in Software Development

- Experience Managing Software Development
  - Maximum experience managing software development – 35 years
  - Minimum experience managing software development – 0 year
  - Average experience managing software development – 9 years

The overall distribution of respondents' experience managing software development is shown in the graph in Figure 4.2. More than half the respondents had at least 5 years of experience managing software development.

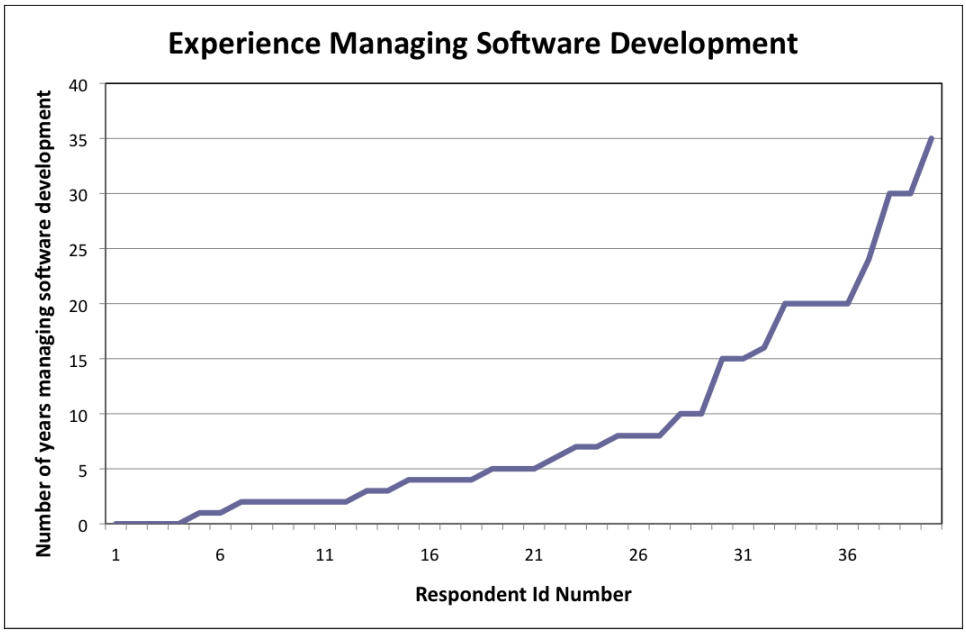

Figure 4.2: Experience Managing Software Development

4.2 Analysis of Results

Due to the small sample size, the test of proportions [12], [11] is used to calculate the statistical significance of the results. This test is useful for predicting whether the observed difference between different categories of responses is statistically valid. There is an inherent ordering in the data obtained but the distance between categories is not clear. Therefore mode was used as the key data measure as both average or median are much less meaningful for this type of data.

The detailed analysis is divided into three sections, for software development activities that are:

1. helped by domain ignorance,
2. hindered by domain ignorance and,
3. unaffected by domain ignorance.

Section 4.3 contains software development activities that are helped by domain ignorance. Section 4.4 contains activities that are not affected by domain ignorance and Section 4.5 contains activities that are hindered by domain ignorance. For each activity, the distribution of respondents' responses is represented by column graphs with the two domain familiarities on the x-axis and the number of people on the y-axis. The statistical significance of the results was calculated using a 5-sample (corresponding to the five values, "required", "enhances", "neutral", "impedes", and "prevents") test of proportions. A 5% error margin was chosen, which is represented by a p-value of 0.05. Therefore a p-value of 0.03 signifies that there is a 97% chance that the difference observed between responses for the domain ignorance category reflects a real difference between populations and a 3% chance that the difference is due to chance occurrence. The statistical tests were done using the R software environment for statistical computing. The test for proving the statistical validity of obtained results is omitted for Sections 4.4 and 4.5, as the aim of this research was to find activities that are helped by domain ignorance.

4.3 Activities Helped by Domain Ignorance

This section lists the software development activities that are helped by domain ignorance, i.e., whose domain ignorance mode is one of the two "helps" scale values, "required" or "enhances".

4.3.1 Eliciting Requirements/Requirements Gathering

Task: Eliciting Requirements/Requirements Gathering

Distribution:

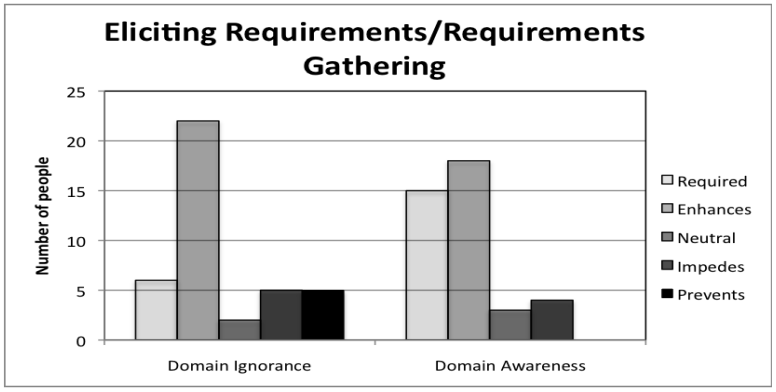

Figure 4.3: Distribution for "Eliciting Requirements/Requirements Gathering"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Enhances

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(6,22,2,5,5),c(40,40,40,40,40),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.150	0.550	0.050	0.125	0.125

Table 4.1: Test of Proportions for "Eliciting Requirements/Requirements Gathering"

p-value = 1.726e-07

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.2 Analyzing Requirements

Task: Analyzing Requirements

Distribution:

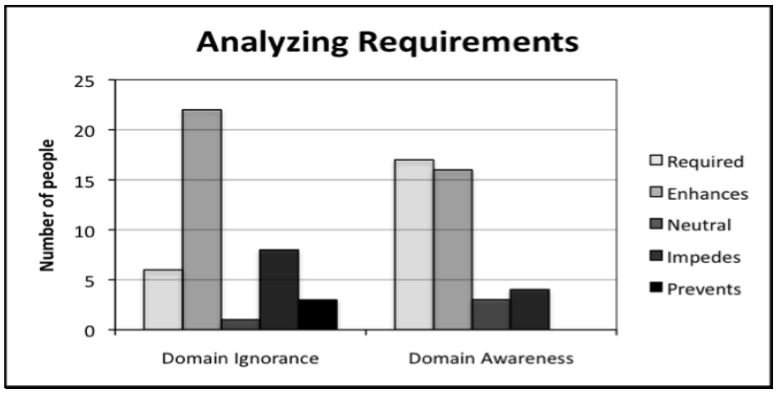

Figure 4.4: Distribution for "Analyzing Requirements"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Required

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(6,22,1,8,3),c(40,40,40,40,40),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.150	0.550	0.025	0.200	0.075

Table 4.2: Test of Proportions for "Analyzing Requirements"

p-value = 4.033e-08

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.3 Identifying Project Risks

Task: Identifying Project Risks

Distribution:

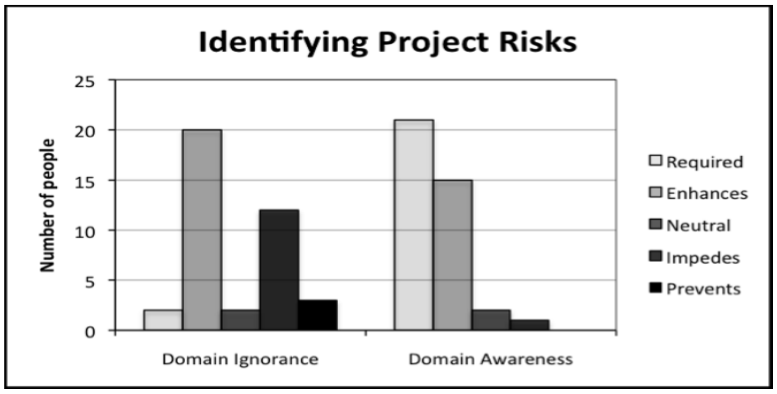

Figure 4.5: Distribution for "Identifying Project Risks"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Required

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(2,20,2,12,3),c(39,39,39,39,39),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.0512	0.5128	0.0512	0.3076	0.0769

Table 4.3: Test of Proportions for "Identifying Project Risks"

p-value = 8.375e-08

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.4 Creating High Level Software Design

Task: Creating High Level Software Design

Distribution:

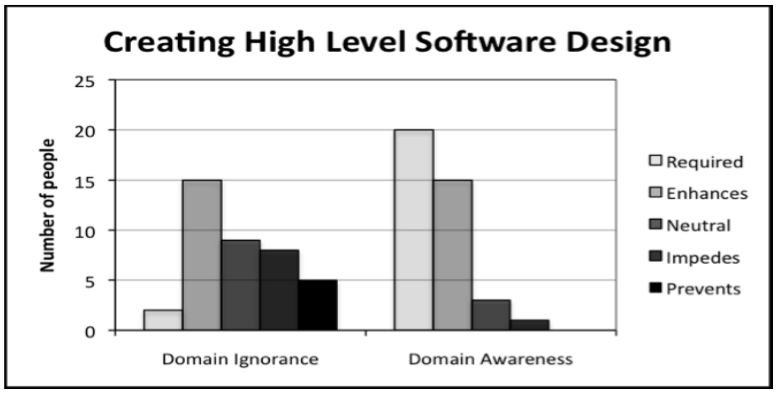

Figure 4.6: Distribution for "Creating High Level Software Design"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Required

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(2,15,9,8,5),c(39,39,39,39,39),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.0512	0.3846	0.2307	0.2051	0.1282

Table 4.4: Test of Proportions for "Creating High Level Software Design"

p-value = .009571

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.5 User Interface Design

Task: User Interface Design

Distribution:

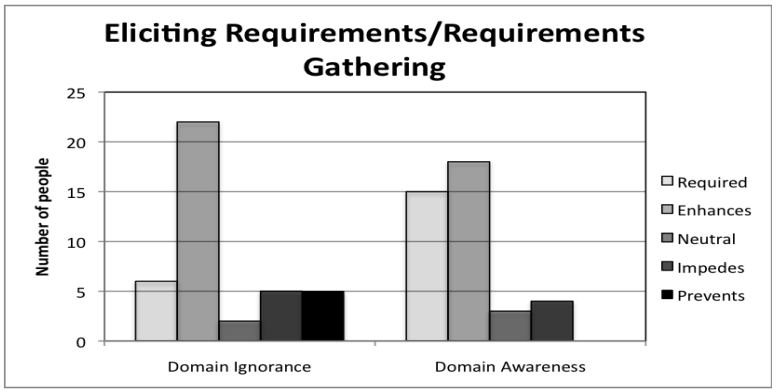

Figure 4.7: Distribution for "User Interface Design"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Enhances

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(6,17,5,4,7),c(39,39,39,39,39),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.1538	0.4358	0.1282	0.1025	0.1794

Table 4.5: Test of Proportions for "User Interface Design"

p-value = 0.003268

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.6 Developing Black Box Test Cases

Task: Developing Black Box Test Cases

Distribution:

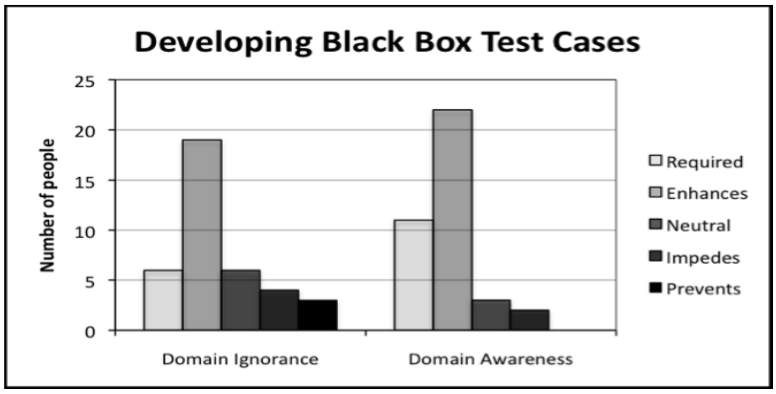

Figure 4.8: Distribution for "Developing Black Box Test Cases"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Enhances

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(6,19,6,4,3),c(39,39,39,39,39),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.1578	0.5000	0.1578	0.1052	0.0789

Table 4.6: Test of Proportions for "Developing Black Box Test Cases"

p-value = 3.931e-05

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.7 Analyzing Defects to Find Common Trends

Task: Analyzing Defects to Find Trends

Distribution:

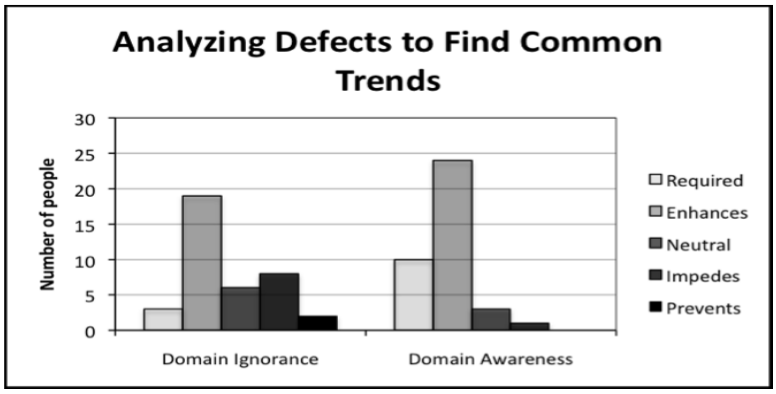

Figure 4.9: Distribution for "Analyzing Defects to Find Common Trends"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Enhances

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(3,19,6,8,2),c(38,38,38,38,38),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.0789	0.5000	0.1578	0.2105	0.0526

Table 4.7: Test of Proportions for "Analyzing Defects to Find Common Trends"

p-value = 1.197e-05

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.8 Identifying Security Risks

Task: Identifying Security Risks

Distribution:

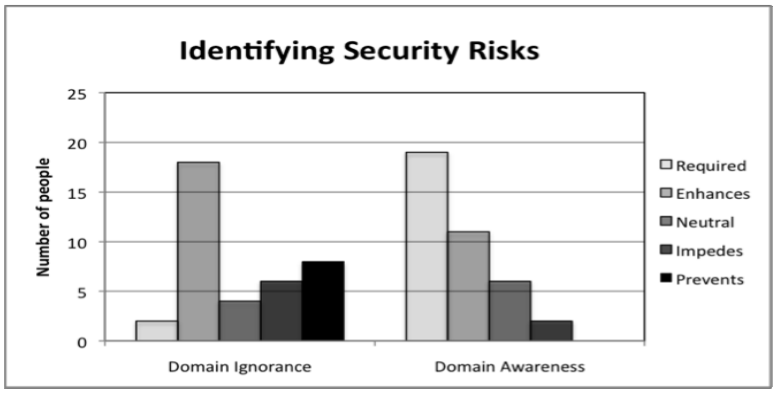

Figure 4.10: Distribution for "Identifying Security Risks"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Required

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(2,18,4,6,8),c(38,38,38,38,38),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.0526	0.4736	0.1052	0.1578	0.2105

Table 4.8: Test of Proportions for "Identifying Security Risks"

p-value = 0.0001102

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.9 Writing User Manuals and Release Notes

Task: Writing User Manuals and Release Notes

Distribution:

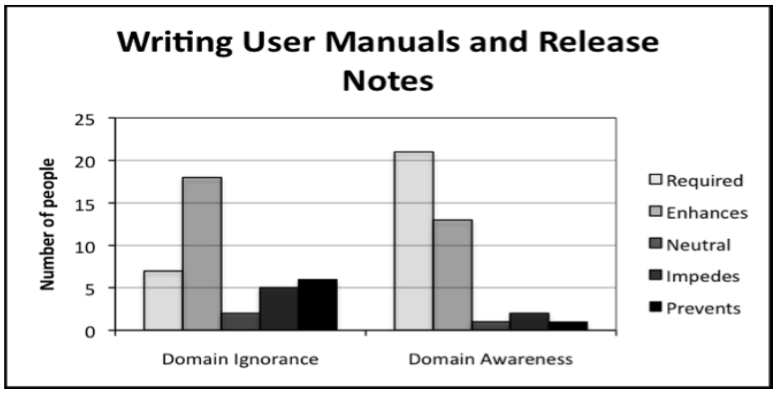

Figure 4.11: Distribution for "Writing User Manuals and Release Notes"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Required

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(7,18,2,5,6),c(38,38,38,38,38),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.1842	0.4736	0.0526	0.1315	0.1578

Table 4.9: Test of Proportions for "Writing User Manuals and Release Notes"

p-value = 0.0001710

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.10 Inspecting/Reviewing Design Documents

Task: Inspecting/Reviewing Design Documents

Distribution:

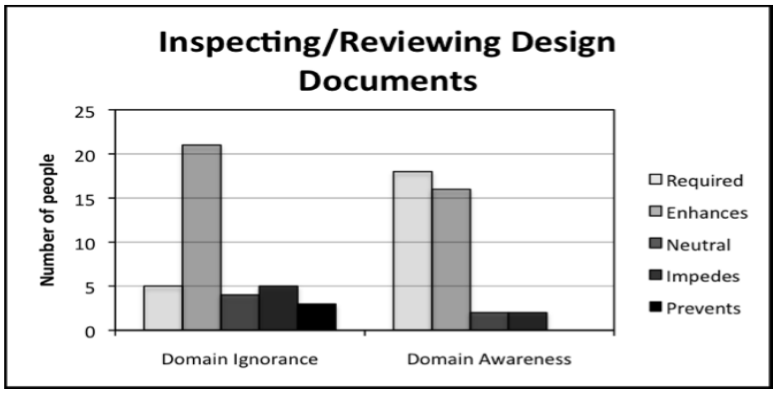

Figure 4.12: Distribution for "Inspecting/Reviewing Design Documents"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Required

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(5,21,4,5,3),c(38,38,38,38,38),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.1315	0.5526	0.1052	0.1315	0.0789

Table 4.10: Test of Proportions for "Inspecting/Reviewing Design Documents"

p-value = 5.052e-07

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.11 Inspecting/Reviewing User Manuals

Task: Inspecting/Reviewing User Manuals

Distribution:

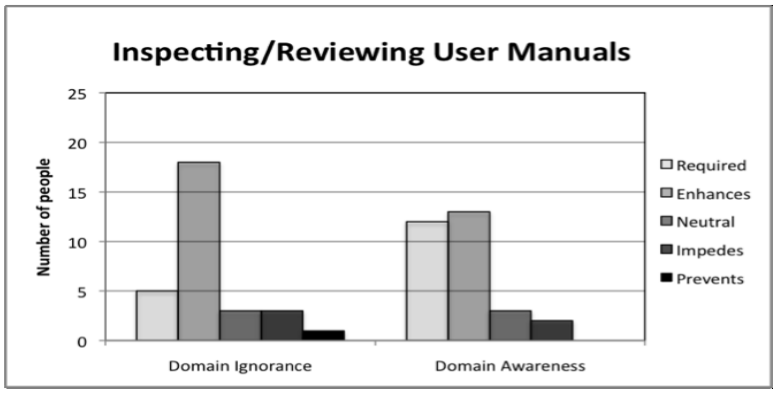

Figure 4.13: Distribution for "Inspecting/Reviewing User Manuals"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Enhances

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(5,18,3,3,1),c(30,30,30,30,30),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.1666	0.6000	0.1000	0.1000	0.0333

Table 4.11: Test of Proportions for "Inspecting/Reviewing User Manuals"

p-value = 2.198e-07

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.12 Inspecting/Reviewing Test Plans

Task: Inspecting/Reviewing Test Plans

Distribution:

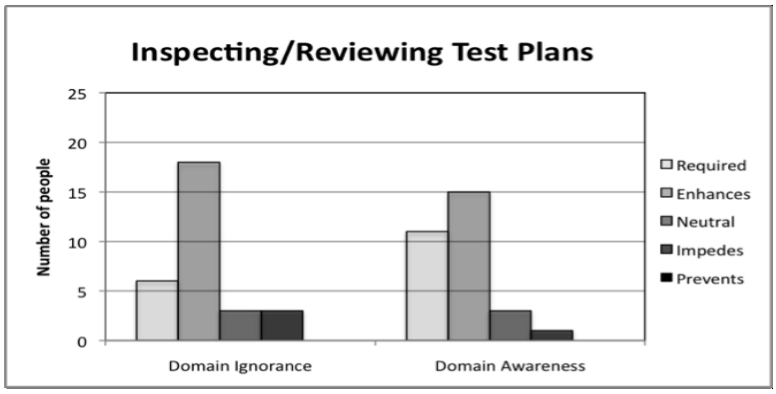

Figure 4.14: Distribution for "Inspecting/Reviewing Test Plans"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Enhances

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(6,18,3,3,0),c(30,30,30,30,30),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.2	0.6	0.1	0.1	0.0

Table 4.12: Test of Proportions for "Inspecting/Reviewing Test Plans"

p-value = 8.353e-08

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.13 Inspecting/Reviewing Requirements Document

Task: Inspecting/Reviewing Requirements Document

Distribution:

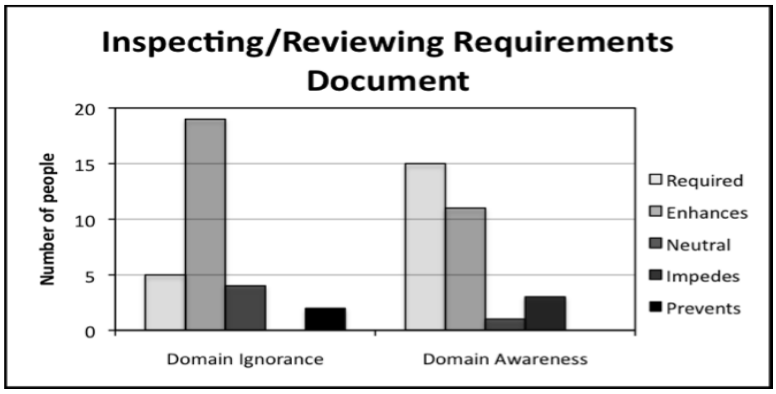

Figure 4.15: Distribution for "Inspecting/Reviewing Requirements Document"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Required

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(5,19,4,0,2),c(30,30,30,30,30),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.166	0.6333	0.1333	0.0000	0.0666

Table 4.13: Test of Proportions for "Inspecting/Reviewing Requirements Document"

p-value = 5.463e-09

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.14 Reading Product Documentation

Task: Reading Product Documentation

Distribution:

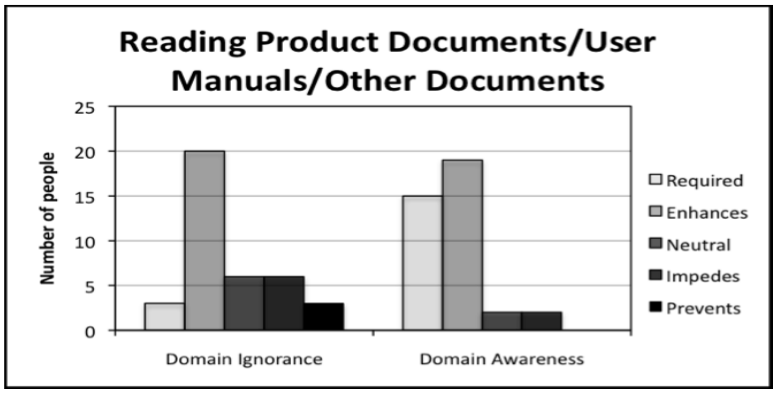

Figure 4.16: Distribution for "Reading Product Documentation"

Modes:

- Mode (domain ignorance) = Enhances
- Mode (domain awareness) = Enhances

Test of proportions for domain ignorance:

Command to the R environment:

prop.test(c(3,20,6,6,3),c(38,38,38,38,38),c(0.2,0.2,0.2,0.2,0.2))

Output:

	proportion1	proportion2	proportion3	proportion4	proportion5
expected	0.2	0.2	0.2	0.2	0.2
observed	0.0789	0.5263	0.1578	0.1578	0.0789

Table 4.14: Test of Proportions for "Reading Product Documentation"

p-value = 3.608e-06

Verdict: Since p-value < 0.05, the results are statistically significant.

4.3.15 Discussion

The activities that the respondents believe to be helped by domain ignorance are:

- requirements gathering,
- analyzing requirements,
- identifying project risks,
- creating high-level software design,
- user interface design,
- developing black box test cases,
- analyzing defects to find common trends,
- identifying security risks,
- writing user manuals/release notes,
- inspecting/reviewing design documents,
- inspecting/reviewing test plans,
- inspecting/reviewing requirement documents,
- inspecting/reviewing user manuals,
- reading user manuals/design documents/other product documentation, and
- learning processes/technology/practices used in the project.

It was surprising that for some software development activities, both ignorance and awareness were perceived to help the activity. These activities are:

- eliciting requirements,
- user interface design,
- developing black box test cases,

- analyzing defects to find trends,
- inspecting/reviewing user manuals,
- inspecting/reviewing test plans, and
- reading product documentation.

Another surprising element was the activity: "Inspecting/Reviewing Requirements Documents" for which the mode of domain ignorance is "enhances" while that of domain awareness is "required". This combination of modes is unusual given that each of the other two inspecting/reviewing activities, "Inspecting/Reviewing Test Plans" and "Inspecting/Reviewing User Manuals", has "enhances" as the mode of both domain awareness and domain ignorance.

4.4 Activities Not Affected by Domain Ignorance

This section lists activities that are not affected by domain ignorance, whose domain ignorance mode is "neutral".

4.4.1 Learning Processes/Practices/Technology Used

Task: Learning Processes/Practices/Technology Used

Distribution:

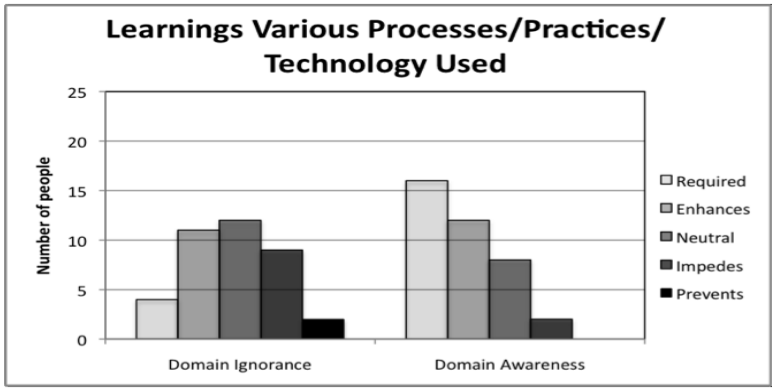

Figure 4.17: Distribution for "Learning Processes/Practices/Technology Used"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Enhances

4.4.2 Source/Version Control Tasks

Task: Source/Version Control Tasks

Distribution:

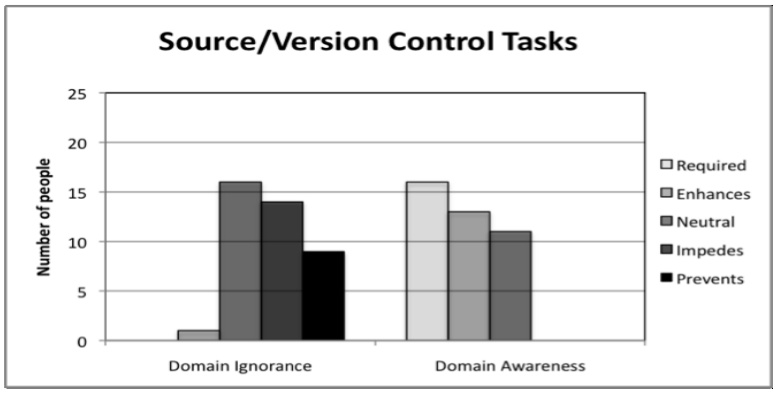

Figure 4.18: Distribution for "Source/Version Control Tasks"

Modes:

- Mode (domain ignorance) = Neutral, Mode (domain awareness) = Required

4.4.3 Coding Simple Features

Task: Coding Simple Features

Distribution:

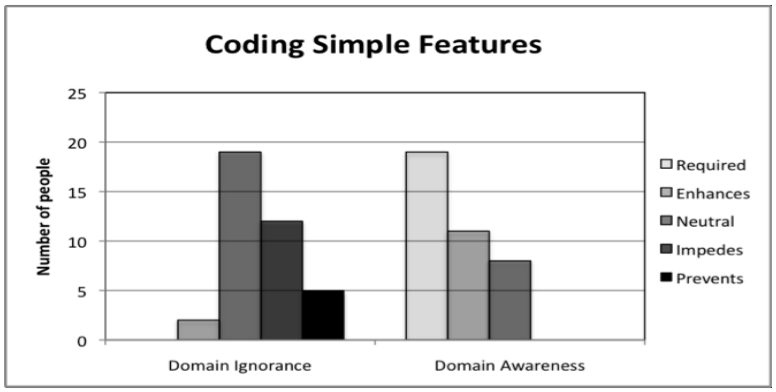

Figure 4.19: Distribution for "Coding Simple Features"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Required

4.4.4 Other Code Oriented Tasks

Task: Other Code Oriented Tasks

Distribution:

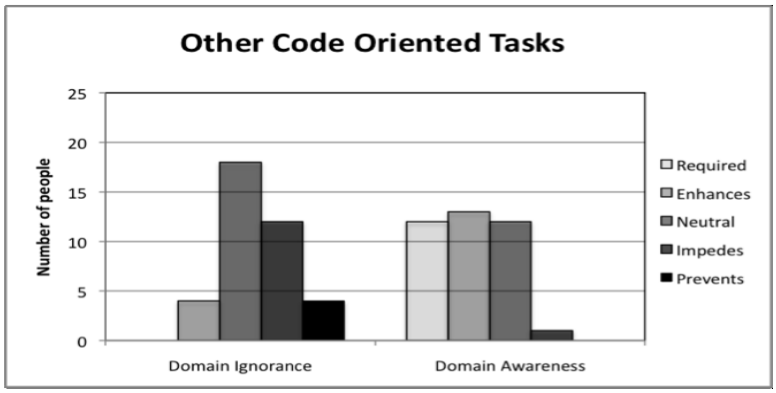

Figure 4.20: Distribution for "Other Code Oriented Tasks"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Enhances

4.4.5 Automating Test Cases

Task: Automating Test Cases

Distribution:

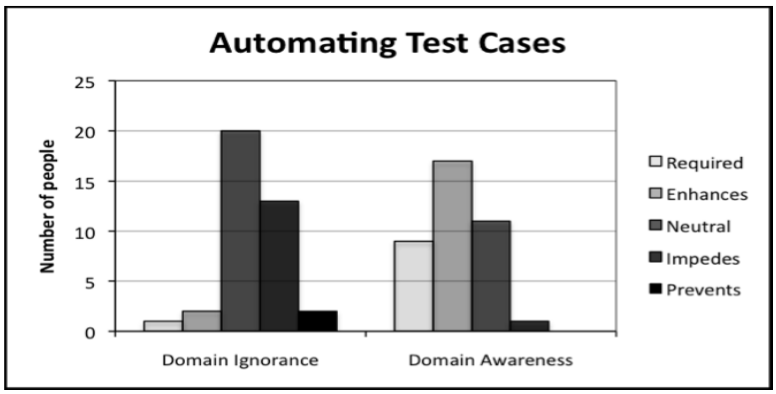

Figure 4.21: Distribution for "Automating Test Cases"

Modes:

- Mode (domain ignorance) = Neutral, Mode (domain awareness) = Enhances

4.4.6 Reviewing Trace Information

Task: Reviewing Trace Information

Distribution:

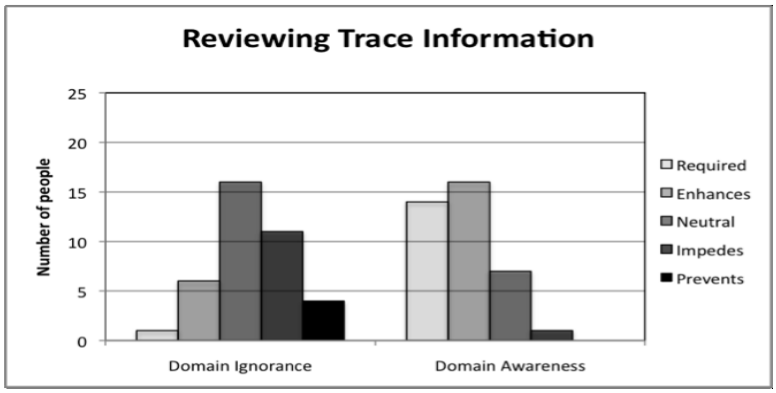

Figure 4.22: Distribution for "Reviewing Trace Information"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Enhances

4.4.7 Attending Courses/Trainings

Task: Attending Courses/Trainings

Distribution:

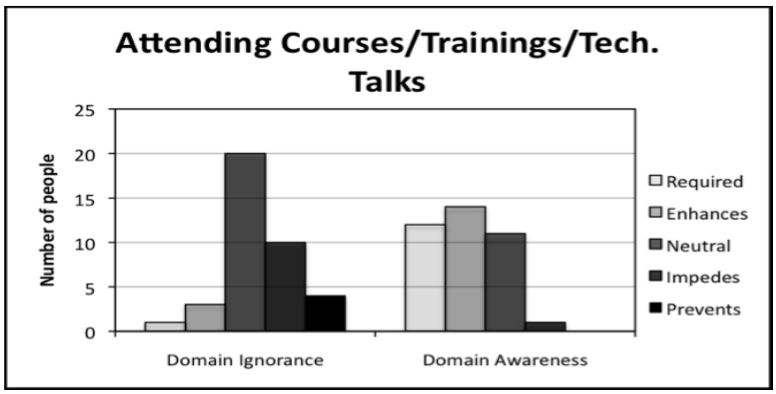

Figure 4.23: Distribution for "Attending Courses/Trainings"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Enhances

4.4.8 Attending Formal Project Meetings

Task: Attending Formal Project Meetings

Distribution:

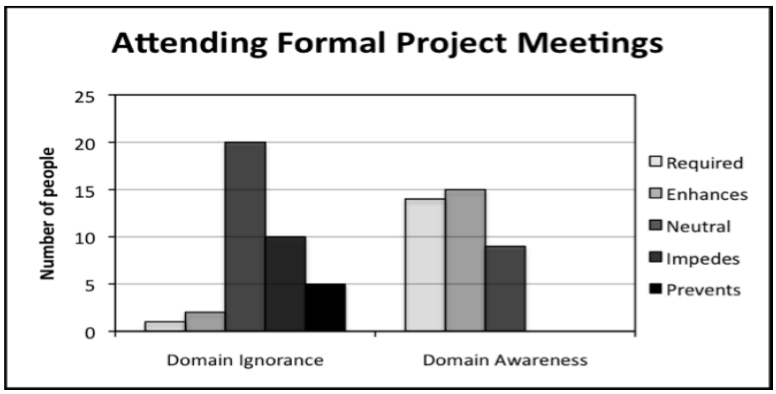

Figure 4.24: Distribution for "Attending Formal Project Meetings"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Enhances

4.4.9 Attending Code/Project Walkthroughs

Task: Attending Code/Project Walkthroughs

Distribution:

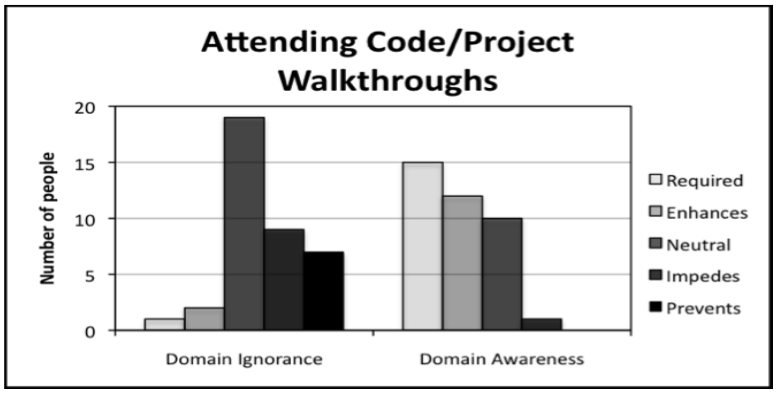

Figure 4.25: Distribution for "Attending Code/Project Walkthroughs"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Required

4.4.10 Compiling Project Code

Task: Compiling Project Code

Distribution:

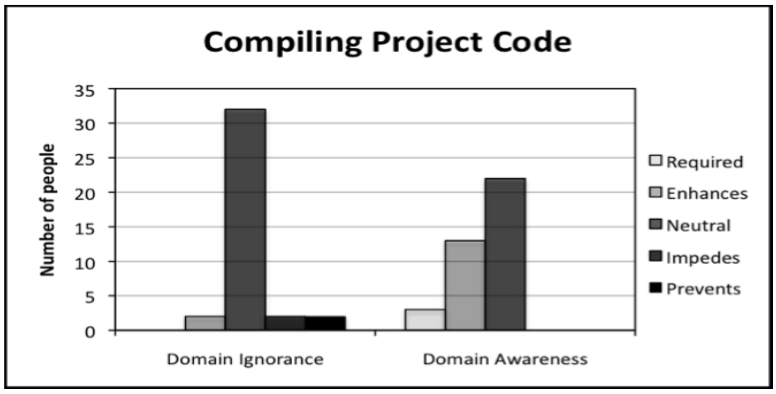

Figure 4.26: Distribution for "Compiling Project Code"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Neutral

4.4.11 Installing and Configuring Development Environment

Task: Installing and Configuring Development Environment

Distribution:

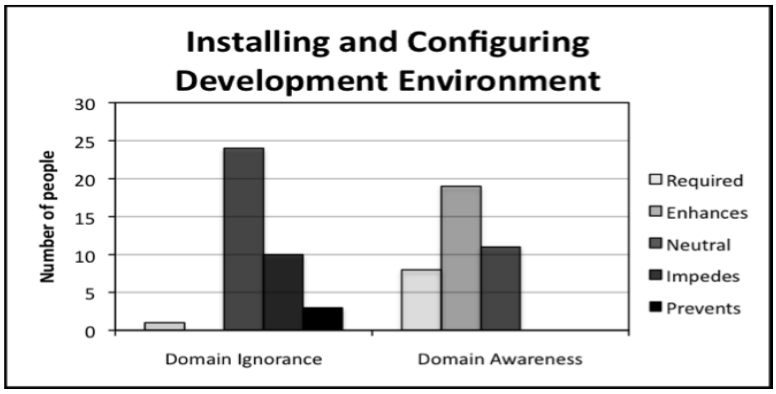

Figure 4.27: Distribution for "Installing and Configuring Development Environment"

Modes:

- Mode (domain ignorance) = Neutral
- Mode (domain awareness) = Enhances

4.4.12 Discussion

The activities that the respondents believe to be not affected by domain ignorance are:

- learning processes/practices/technology used,

- source/version control tasks,
- coding simple features,
- other code oriented tasks,
- automating test cases,
- reviewing trace information,
- attending courses/trainings,
- attending formal project meetings,
- attending code/project walkthroughs,
- compiling project code, and
- installing and configuring development environment.

4.5 Activities Hindered by Domain Ignorance

This section lists software development activities that are *hindered* by domain ignorance, i.e., whose domain ignorance mode is one of the "hinders" scale values, "impedes" or "prevents".

4.5.1 Designing and Specifying Software Architecture

Task: Designing and Specifying Software Architecture

Distribution:

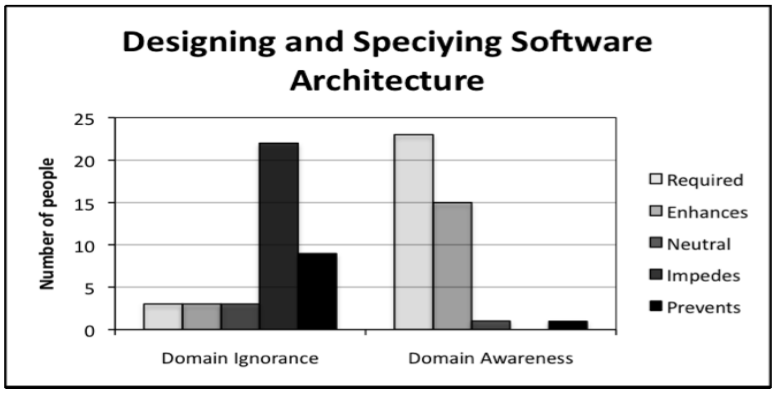

Figure 4.28: Distribution for "Designing and Specifying Software Architecture"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.2 Reviewing Software Architecture

Task: Reviewing Software Architecture

Distribution:

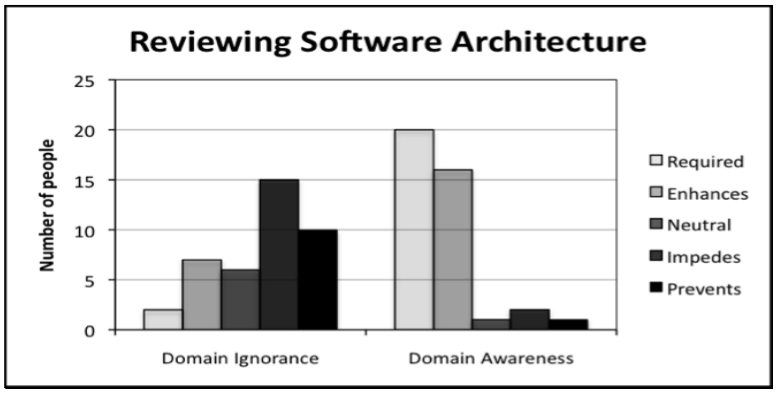

Figure 4.29: Distribution for "Reviewing Software Architecture"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.3 Specifying Requirements

Task: Specifying Requirements

Distribution:

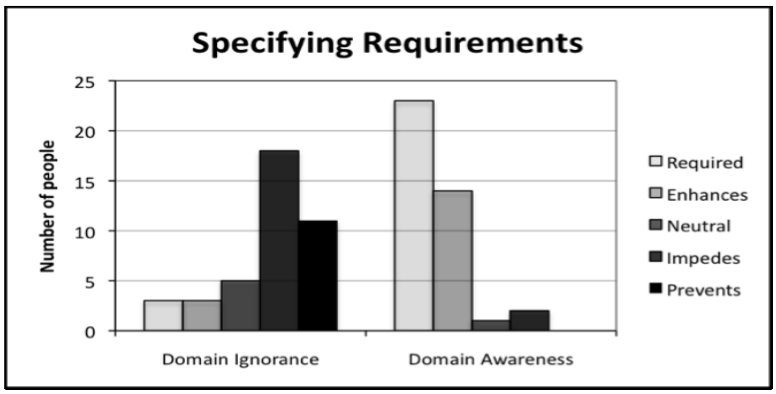

Figure 4.30: Distribution for "Specifying Requirements"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.4 Validating Requirements

Task: Validating Requirements

Distribution:

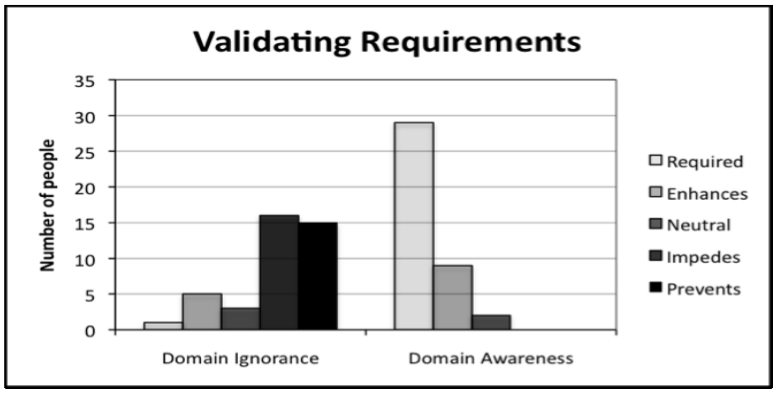

Figure 4.31: Distribution for "Validating Requirements"

Modes:

- Mode (domain ignorance) = Impedes, Mode (domain awareness) = Required

4.5.5 Reusing and Managing Requirements

Task: Reusing and Managing Requirements

Distribution:

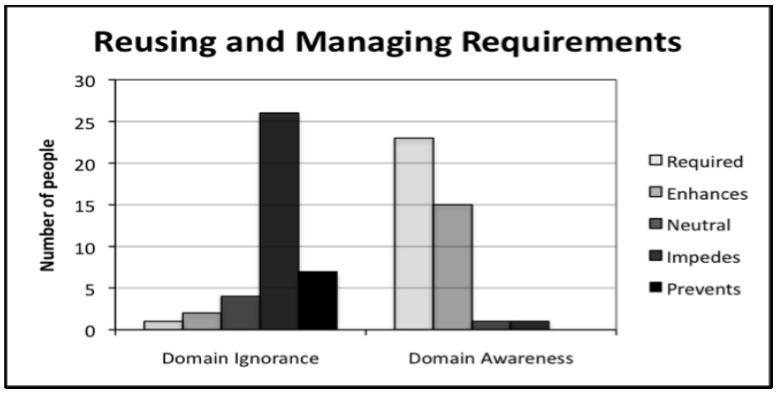

Figure 4.32: Distribution for "Reusing and Managing Requirements"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.6 Managing Builds of a Software

Task: Managing Builds of a Software

Distribution:

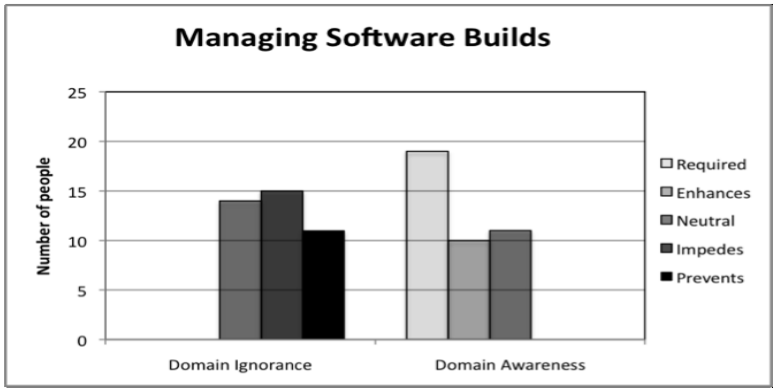

Figure 4.33: Distribution for "Managing Builds of a Software"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.7 Deployment Planning

Task: Deployment Planning

Distribution:

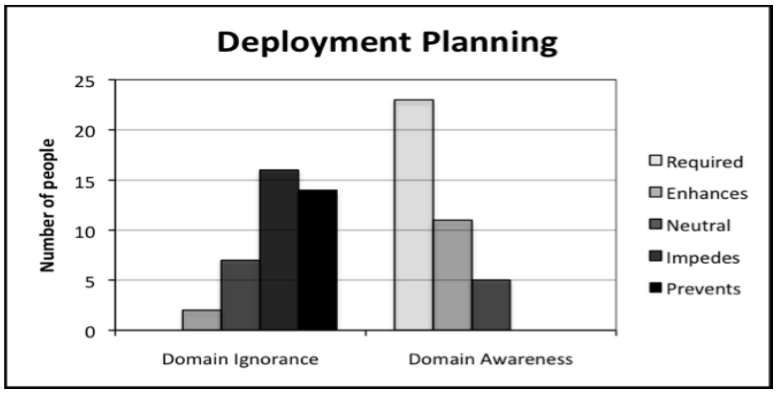

Figure 4.34: Distribution for "Deployment Planning"

- Mode (domain ignorance) = Impedes, Mode (domain awareness) = Required

4.5.8 Risk Planning/Monitoring and Control

Task: Risk Planning/Monitoring and Control

Distribution:

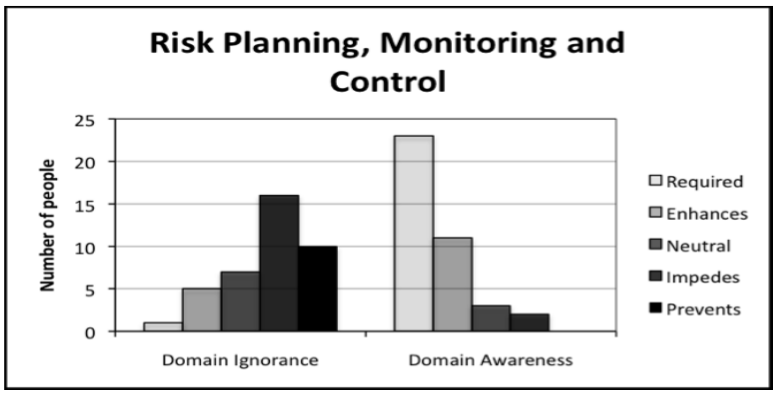

Figure 4.35: Distribution for "Risk Planning/Monitoring and Control"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.9 Creating Low Level Software Design

Task: Creating Low Level Software Design

Distribution:

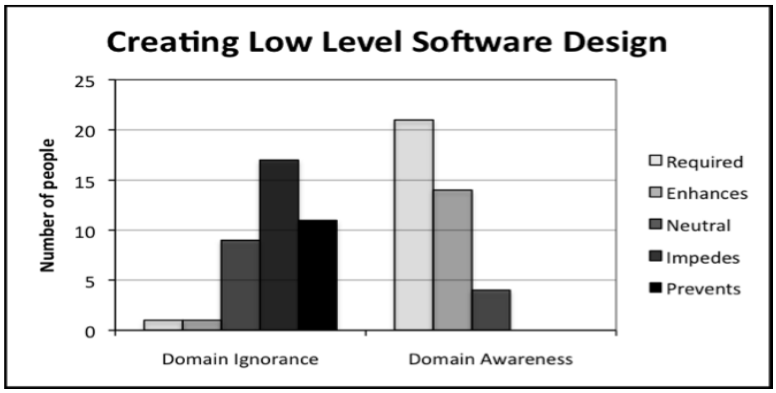

Figure 4.36: Distribution for "Creating Low Level Software Design"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.10 Identifying Design and Implementation Rationale

Task: Identifying Design and Implementation Rationale

Distribution:

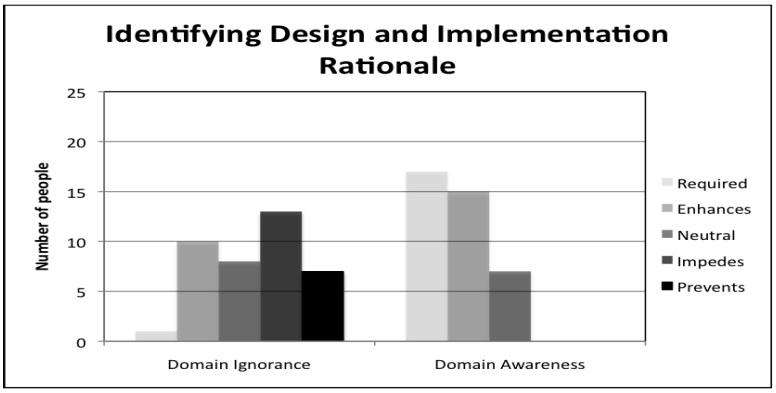

Figure 4.37: Distribution for "Identifying Design and Implementation Rationale"

Modes:

- Mode (domain ignorance) = Impedes, Mode (domain awareness) = Required

4.5.11 Fixing Bugs

Task: Fixing Bugs

Distribution:

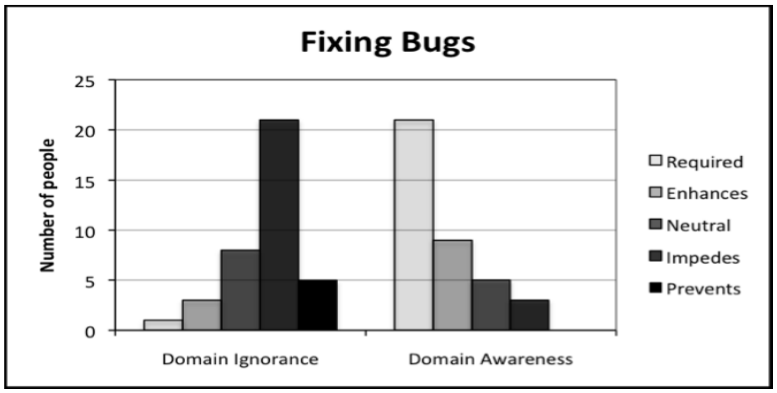

Figure 4.38: Distribution for "Identifying Design and Implementation Rationale"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.12 Developing Unit Test Cases

Task: Developing Unit Test Cases

Distribution:

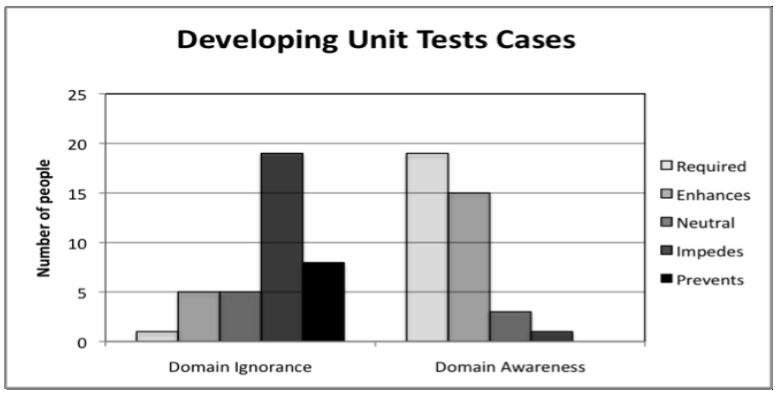

Figure 4.39: Distribution for "Developing Unit Test Cases"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.13 Developing White Box Test Cases

Task: Developing White Box Test Cases

Distribution:

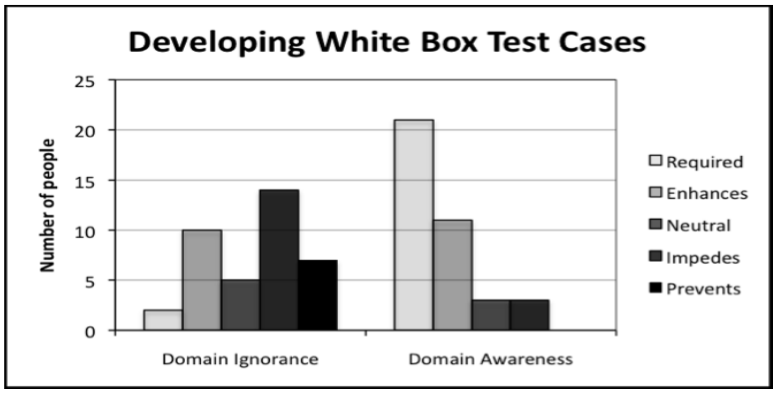

Figure 4.40: Distribution for "Developing White Box Test Cases"

Modes:

- Mode (domain ignorance) = Impedes, Mode (domain awareness) = Required

4.5.14 Developing Integration Test Cases

Task: Developing Integration Test Cases

Distribution:

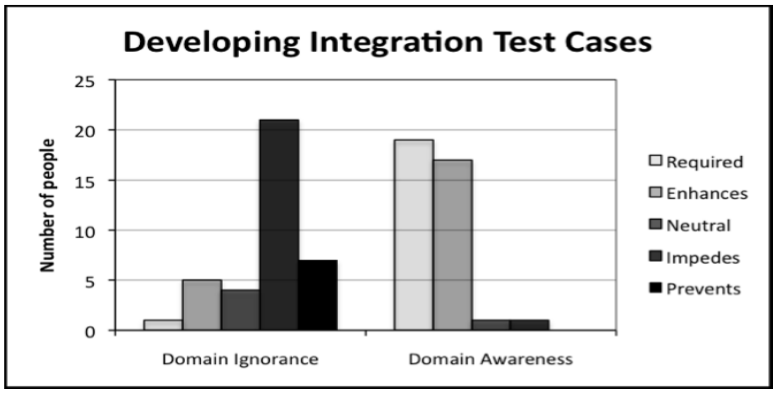

Figure 4.41: Distribution for "Developing Integration Test Cases"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.15 Determining Source of a Bug

Task: Determining Source of a Bug

Distribution:

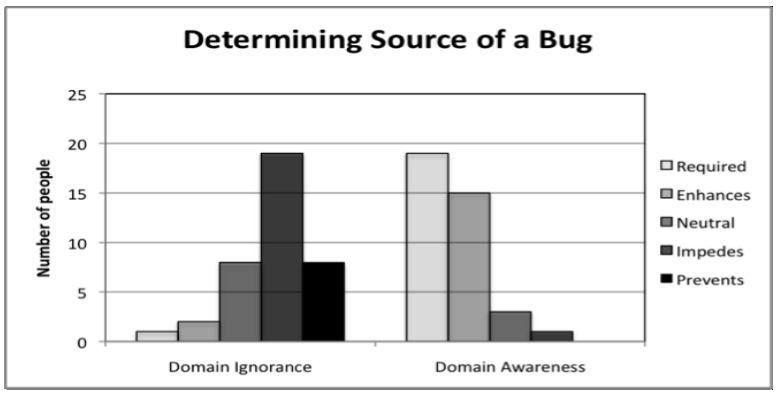

Figure 4.42: Distribution for "Determining Source of a Bug"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.16 Test Planning for a Release

Task: Test Planning for a Release

Distribution:

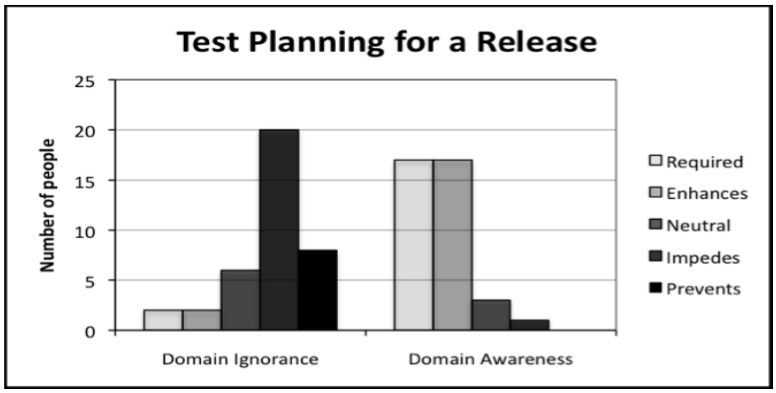

Figure 4.43: Distribution for "Test Planning for a Releases"

Modes:

- Mode (domain ignorance) = Impedes, Mode (domain awareness) = No mode present

4.5.17 Developing System/Performance Test Cases

Task: Developing System/Performance Test Cases

Distribution:

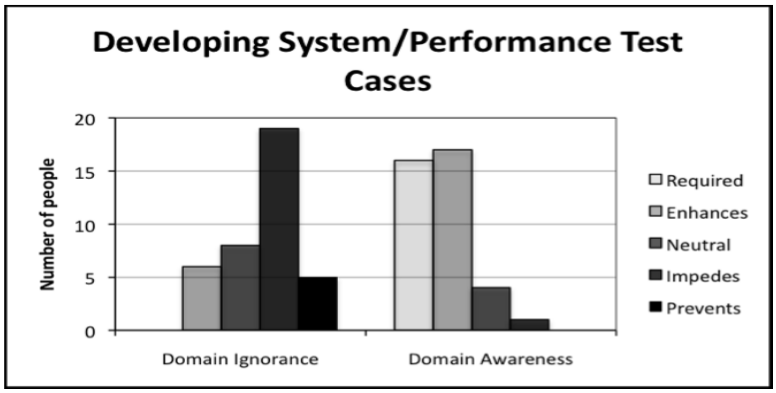

Figure 4.44: Distribution for "Developing System/Performance Test Cases"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Enhances

4.5.18 Manually Executing Test Cases

Task: Manually Executing Test Cases

Distribution:

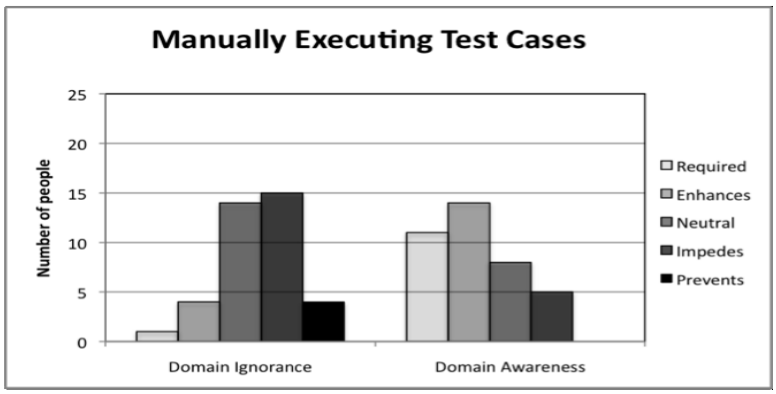

Figure 4.45: Distribution for "Manually Executing Test Cases"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Enhances

4.5.19 Preventing Security Threats

Task: Preventing Security Threats

Distribution:

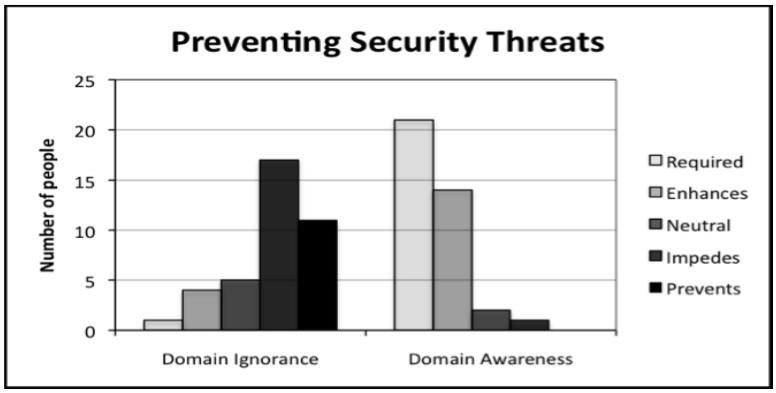

Figure 4.46: Distribution for "Preventing Security Threats"

Modes:

- Mode (domain ignorance) = Impedes, Mode (domain awareness) = Required

4.5.20 Providing Technical Support to Users

Task: Providing Technical Support to Users

Distribution:

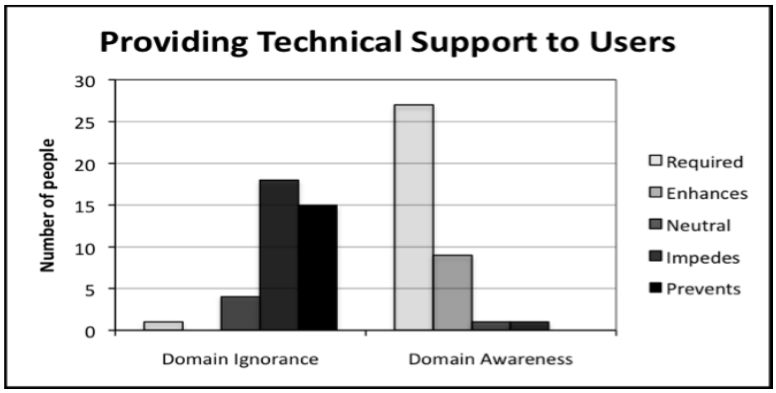

Figure 4.47: Distribution for "Providing Technical Support to Users"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.21 Inspecting Code

Task: Inspecting Code

Distribution:

Figure 4.48: Distribution for "Inspecting Code"

Modes:

- Mode (domain ignorance) = Impedes
- Mode (domain awareness) = Required

4.5.22 Discussion

A surprise was that there was no domain awareness mode for the software development activity "Test Planning for a Release". Also, none of the software development activities in this section have "prevents" as the domain ignorance mode. Thus, the respondents believe that domain ignorance never prevents a newbie from performing any software development activity, although it might impede the performance of some activities.

4.6 Comparison with Data from Other Studies

At ICSE 2010 prior to the beginning of the work on this thesis, Berry had heard the presentation of a paper by Barthélémy Dagenais, Harold Ossher, Rachel Bellamy, Martin Robillard, and Jacqueline de Vries [7] studying the immigration of newbies into software development projects, with an aim to determine how to make these immigrations smoother. Based on the one example presented in the talk of a smooth immigration, and aware of his earlier work [3] on the importance of ignorance in requirements engineering, Berry hypothesized that

> an immigrant's immigration was smoothest when the immigrant was put to work doing a task for which domain ignorance is helpful.

The example newbie who had a smooth immigration had been assigned the task of fixing bugs, and Berry's experience told him that fixing bugs is an activity that benefits from domain ignorance[1].

Armed with the list of software development tasks that are believed to benefit from domain ignorance, the next step is to try to test Berry's hypothesis by re-examining the data from the immigration study [7] to determine the tasks performed by the immigrants with the smoothest immigrations.

Berry and I approached Dagenais *et al.* for access to their raw data, transcripts describing their newbies' immigration experiences. After some discussions, we signed a non-disclosure agreement and obtained the transcripts about the immigrations of 14 newbies. These transcripts contained information regarding the tasks a newbie was assigned during his initial days and the difficulties faced by him in doing those tasks. I decided to analyse the transcripts and grouped the activities performed by the 14 newbies into two lists,

1. a positive list that contains each activity for which at least one newbie said that the activity helped him immigrate, and
2. a negative list which contains each activity for which at least one newbie said that the activity did not help him immigrate.

If an activity were initially in both lists, then I would put it finally in the positive list if it helped more people than it did not help, and I would put it finally in the negative list

[1]Note that the results of the survey say otherwise, but this hypothesis was formed long before the survey was even written, and it prompted the research leading to this survey.

if it did not help more people than it helped. As it turned out, no activity was initially in both lists. The results are summarized in the two lists below. For each entry in each list, the number in the parentheses is the number of newbies mentioning the activity for the list, and the text in the square brackets at the end of the entry denotes the mode of domain ignorance of the entry's activity in the results of the survey.

The activities in the positive list are:

- Reading product documentation (4) [enhances]
- Inspecting test plans, design documents (1) [enhances]
- Fixing bugs (1) [impedes]
- Learning processes/practices/technology (4) [neutral]
- Coding simple features (1) [neutral]
- Reviewing trace information (1) [neutral]
- Attending code/project walkthroughs (1) [neutral]
- Compiling project code (2) [neutral]

The activities in the negative list are:

- Installing/configuring development environment (2) [neutral]
- Source/version control tasks (1) [neutral]
- Writing design documents/software architecture (1) [impedes]
- Attending formal project meetings (3) [neutral]

If the activities that are thought to be neutral are eliminated from the two lists,

- the positive list is left with a majority of activities that domain ignorance is thought to enhance, and
- the negative list is left with one activity that domain ignorance is thought to impede.

Therefore, there is very marginal support for Berry's hypothesis.

A drawback of the transcripts is that they did not contain any data about how smooth the immigrations were. It would be very nice to be able to correlate these results with such data. Recognizing that the best judge of the smoothness of a newbie's immigration is the newbie himself, I decided to request additional data from Dagenais *et al.* I asked Ossher to send the following question to each of the 14 participants of his study whose transcripts I received:

> Rate your immigration experience using the scale:
> Torture, Painful, Neutral, Smooth, Ecstatic

As the immigration study was done a long time ago, Ossher was reluctant to contact the participants again and did not agree to send the question to the 14 participants. He did offer instead some additional data derived by Dagenais during the study for each participant, a binary classification, successful or non-successful, of his immigration experiences and the reason for the classification. Note that this classification was performed by an independent third party using some definition that he chose. I decided to accept this classification at face value, taking "successful" to have its vernacular meaning. Therefore, I gladly accepted these data, deciding to use "successful" as a best available, albeit imperfect, proxy for "smoothness".

Out of the total 14 participants, Participants 6, 7, 11, and 14 reported an overall non-successful immigration. Some of the reasons given for the non-successful immigration are:

Participant 6: Participant 6[2] got assigned to a job that he was not qualified for; his colleagues told him in various indirect ways that he should not have this position; and he got assigned to critical tasks with insufficient support.

Participant 7: Participant 7 was a team leader and his team was supposed to take over a project from another team, but the original team did not want to relinquish the project; the original team put a lot of obstacles in the way, including rude comments, outdated documentation, and long delays in answering emails.

I divided the newbies into two groups:

[2]Recall that the gender of the first arbitrary individual in a discourse toggles for each section, but remains constant in a section. Thus, the use of the male gender in this even numbered chapter does not imply that the referenced participant is necessarily a male.

1. one of those who had a smooth immigration and
2. and another of those who did not have a smooth immigration.

I decided to build two lists of activities.

1. one of all activities done by anyone who had a smooth immigration and
2. and another of all activities done by anyone who did not have a smooth immigration.

If an activity were initially in both lists, then I would put it finally in the smooth immigration activities list if more people in the smooth immigration group performed the activity than people in the other group, and I would put it in the non-smooth immigration activities list if more people in the non-smooth immigration group performed the activity than people in the other group. As it turned out, no activity was initially in both lists.

The activities in the smooth immigration list are:

- Reading product documentation (4) [enhances]
- Inspecting test plans, design documents (1) [enhances]
- Fixing bugs (1) [impedes]
- Learning processes/practices/technology (4) [neutral]
- Coding simple features (1) [neutral]
- Reviewing trace information (1) [neutral]
- Attending code/project walkthroughs (1) [neutral]
- Installing/configuring development environment (2) [neutral]

The activities in the non-smooth immigration list are:

- Writing design documents/software architecture (1) [impedes]
- Inspecting Code (2) [impedes]
- Source/version control tasks (1) [neutral]

- Attending formal project meetings (3) [neutral]

If the activities that are thought to be neutral are eliminated from the two lists,

- the smooth immigration list is left with two activities that domain ignorance is thought to enhance, and
- the non-smooth immigration list is left with two activities that domain ignorance is thought to impede.

Therefore, here too, there is very marginal support for Berry's hypothesis.

I now had two pairs of lists that should be the same if Berry's hypothesis held and the transcripts provided full information. While the two pairs of list are not exactly the same, there is good overlap

- between the positive and the smooth immigration activities lists and
- between the negative and the non-smooth immigration activities lists.

The activities that ended up in only one of the pairs of lists are:

1. **Compiling project code** [**neutral**]: only in the positive group in the first pair of lists,
2. **Installing/configuring development environment** [**neutral**]: only in the negative group in the first pair of lists,
3. **Inspecting Code** [**impedes**]: only in non-smooth immigration activities group in the second pair of lists, and
4. **Writing design documents/software architecture** [**impedes**]: only in the negative group in the first pair of lists.

Nevertheless, if the activities that are thought to be neutral are eliminated from the two pairs of lists,

- the combined positive and smooth immigration lists are left with a majority of activities that domain ignorance is thought to enhance, and

- the combined negative and non-smooth immigration lists are left with only activities that domain ignorance is thought to impede.

Therefore, in the end, there is very marginal support for Berry's hypothesis.

It is somewhat ironic that the original task that prompted Berry to make his hypothesis, the task of fixing bugs, that Berry's experience told him benefited from domain ignorance, ended up being thought as one that is impeded by domain ignorance.

4.6.1 Discussion

That the support for Berry's hypothesis, that immigration was smoothest when the immigrant was put to work doing a task for which domain ignorance is helpful, is only very marginal is not surprising. In real life, there are many factors affecting smoothness of one's immigration, including his personality. There are not enough data in the immigration study to determine root causes of the outcome of any immigration. The ultimate cause of a smooth or non-smooth immigration could be any of the other factors, some combination of factors, or yet other factors not even considered. Without doing a controlled experiment, which perhaps will not simulate real life, we cannot isolate any factor. The best that can be said is:

> All other factors being equal, there is some support that a newbie should be assigned an activity that is helped by domain ignorance.

A newbie should be assigned an activity that is helped by domain ignorance, even if for no other reasons than that

- he becomes useful to his project immediately, and
- he learns the domain of the project in a more leisurely natural manner with less pressure to apply his knowledge prematurely.

4.7 Threats

Each conclusion of this thesis has its own set of threats, and each set can be divided into two classes,

1. threats to internal validity that concerns how well the case study was executed and
2. threats to external validity that concerns whether the conclusions obtained are generalizable.

4.7.1 Threats to Validity of Survey Conclusions

The first conclusion of this thesis is the classification of software development activities according to whether they are helped by, hindered by, or unaffected by domain ignorance. The threats to internal validity of this conclusion are:

- **the bias of the chosen sampling method**:
  As mentioned in Section 3.1, snowball sampling is believed to produce highly biased results. This threat was mitigated by coupling snowball sampling with judgemental sampling, in which the judgement chose participants that were likely to give usable answers.
- **the survey questions**:
  - **Were the survey questions understandable**
    I conducted a pilot study to test the understandability of the questions. The consistency of the results and the specific comments received from the pilot participants indicate that for the most part the questions were understandable, and the few that were not were changed for the actual study.
  - **Were the questions interpreted correctly and the same way by all**
    Ultimately there is no way to know for sure, except by interviewing each respondent personally and asking follow up questions, something that is hard to do when the respondents are anonymous. Nevertheless, the high consistency among the answers to related questions in any one response and the fact that the results were statistically significant indicate that the questions were probably interpreted correctly and in at least a similar way by all. Moreover, the participant pool for the survey consisted of people having significant experience in software development who should have a fair understanding of the terms used in the survey questions.
  - **Did the length of the survey induce survey fatigue, with its attendant deteriorated answers?**:

There were no incomplete questionnaires, and that the answers to related questions in any one response were highly consistent with each other indicates that survey fatigue was not a problem.

- **the method to compute the results using modes**:
  Considering the type of data in the study, no other measure i.e., mean or median, made much sense. Therefore, the mode of the data was used to determine the results. In the future, the survey could ask the respondent how confident he is about his answers.

The threats to external validity of the conclusion are:

- **representativeness of the sample**:
  The judgemental part of the sampling that tried to select people experienced in software development management succeeded to get a collection of respondents who were 75% commercial people with an average of 9 years of experience managing software developments.
- **number of respondents**:
  The high confidence level of the tests for statistical significance says that 40 respondents were enough.

4.7.2 Threats to Validity of Hypothesis Conclusion

The second conclusion of this thesis is that the immigration of a newbie is the smoothest if a newbie is assigned a task which is helped by domain ignorance. The threats to internal validity of this conclusion are:

- **the lack of control over variables**:
  Because I used transcripts supplied by a third party from a study in which they controlled the variables they needed for their study, I had no control over the data that transcripts reported. All I could do was hope that I would be able to see evidence of the variables I needed in the transcripts provided. Fortunately, I was able to find some usable evidence in all of the 14 transcripts provided.
- **the methods of determining positive, negative, smooth, and non-smooth activities**:
  There may be other possible methods for doing the categorization, but they did not

present themselves. Note however, that the positive–negative activities classification was done in a direction different from that of the smooth–non-smooth activities classification. Moreover, the smooth–non-smooth activities classification was based on an independent classification of immigration success. That the two pairs of lists resulting from the classifications agreed so well strengthens confidence in the correctness of the methods.

The threats to external validity of the conclusion are:

- **representativeness of the sample**:
  The threat to the present study is the same as to the Dagenais et al. study.
- **number of respondents**:
  The threat to the present study is the same as to the Dagenais et al. study.

The threats described above, particularly, those about

- the survey questions,
- the method to compute the results using modes,
- the lack of control over variables, and
- the methods of determining positive, negative, smooth, and non-smooth activities

could easily have conspired to make it impossible to draw *any* conclusions. After all, what are the chances of drawing any conclusion if people do not agree on the meanings of the descriptions of the activities? For example, deciding whether Berry's hypothesis is supported depends on the

1. (1) survey respondents',
2. (2) my,
3. (3) Dagenais et al.'s subjects', and
4. (4) Dagenais et al.'s

all agreeing enough on the meaning of the descriptions of the activities, e.g., that one person's, "eliciting requirements/requirements gathering" is similar enough to all others'. The fact that these four independent sources of data have come together to support Berry's hypothesis even marginally when there are so many other variables that could have affected the results is something of a miracle. Nevertheless, each reader must decide for himself whether to believe the conclusions.

Chapter 5

Applications

This chapter lists some of the areas to which the results of this study can be applied.

5.1 As a Checklist

Often a newbie in a company or project is left to wander alone and explore the project landscape by trial and error. Trial and error is slow and it takes a lot of time until a newbie finally becomes productive. In some cases, a newbie is assigned a mentor who guides her through the project landscape. Mentoring produces results faster but the mentor ends up spending a lot of his time in mentoring rather than doing his normal activities. A software development manager can use the results of this study as a checklist to help assign suitable roles to a newbie in any team. The tasks that are more suitable for a newbie are the ones for which ignorance helps.

There are two aspects to a newbies' immigration within a project. The first is productivity during the immigration and the second is learning about the new domain. By assigning the right task to a newbie, a manager can ensure that she will be productive earlier because her ignorance is put to good use while she learns the domain, i.e., to not be ignorant. As a result, her immigration into the new project is likely to be much smoother thereby increasing the productivity of the team as whole. The tasks thought by the survey respondents likely to be suitable for a newbie in a team are:

- requirements gathering,
- analyzing requirements,

- identifying project risks,
- creating high-level software design,
- user interface design,
- developing black box test cases,
- analyzing defects to find common trends,
- identifying security risks,
- writing user manuals/release notes,
- inspecting/reviewing design documents
- inspecting/reviewing test plans
- inspecting/reviewing requirement documents,
- inspecting/reviewing user manuals,
- reading user manuals/design documents/other product documentation, and
- learning processes/technology/practices used in the project.

5.2 Crowdsourcing

Another application of this work may be to find activities suitable for crowdsourcing.

Wikipedia describes crowdsourcing as "the act of outsourcing tasks, traditionally performed by an employee or contractor, to an undefined, large group of people or community (a crowd), through an open call". Figure 5.1 describes the various steps involved in crowdsourcing.

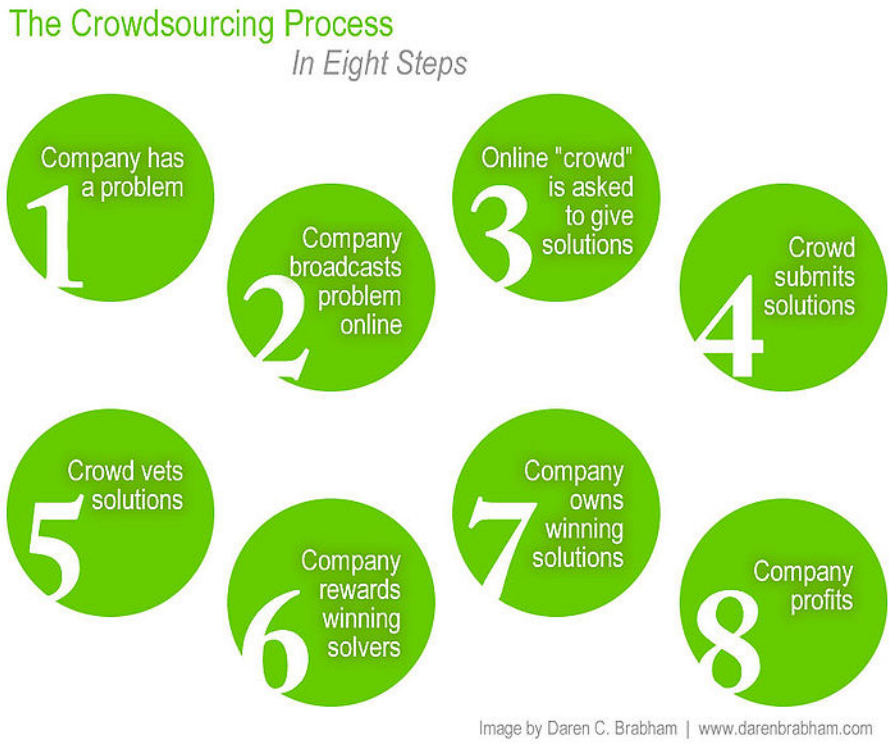

Figure 5.1: Steps in Crowdsourcing

Crowdsourcing follows a distributed approach in which the original problems are broadcast by the crowdsourcer to an unknown group of people in the form of an open-for-all call-for-solutions. Users, also referred to as the crowd, submit solutions. The users in the crowd also vote for the solutions, in order to find the best ones. The best solutions are the property of the entity that broadcast the problem and the winning users in the crowd are sometimes rewarded. In some cases, the winners are well compensated, either monetarily, with prizes, or with recognition. In other cases, the only rewards may be virtual or intellectual satisfaction.

Some of the perceived benefits of crowdsourcing include:

- quick, easy, and cheap way of exploring new problems,
- payment absent or depending upon quality of results,

- helps tap a wider range of talent than available in the normal work force of an organization,
- crowd opinions are good representations of the end users' desire, and
- good brand building opportunity for the organization.

Since crowdsourcing delegates a task to a group of people who may or may not be aware of the problem domain, the activities likely to be suitable for crowdsourcing are the ones for which domain ignorance is thought to be helpful.

5.3 Selecting the Right Mix of People

One final use of the checklist is to assign the right mix of people needed for the success of a particular software development activity. A software manager can use the checklist to help assign the best mix of people for the successful completion of an activity. For example, the activities that are helped by domain ignorance should be assigned to a team consisting of newbies as well as experts in the domain.

Chapter 6

Conclusions and Future Work

This research highlights the importance of domain ignorance in various software development activities. The survey has shown that there is a consensus among software development managers on how ignorance can help the performance of some software development activities. A manager can use the results of this research in order to assign the right tasks to a newbie in his team. A newbie in turn can use his domain ignorance to be productive right from the start while beginning to learn the domain under less pressure to do it too quickly. The productivity of the entire team is increased, and the precious time of other experienced team members is saved.

There is a lot of scope for future work in this area. It would be interesting to repeat the study using a focus group of senior managers in order to have finer grained data. A focus group could also help eliminate any confusions that survey participants might have regarding the survey. The current survey data are only as good as the participants' understanding of the questions. It might be useful to try a different grouping of the software development activities to make the participants think in a different manner. Lastly, it would be useful to perform a study in a real work environment where newbies can be observed working on the assigned tasks during their immigration.

APPENDICES

Appendix A

Survey for Role of Ignorance

Role of Ignorance in Software Development Activities

1.

Background

Ignorance of the domain is commonly thought to be helpful in software development activities which require some critical, out of the box thinking on the part of the person doing it. An example of one such activity is brainstorming for requirement idea generation. Ignorance of the domain is believed to help one to avoid the domain's tacit assumptions or to think outside of the domain's box. The right kind of ignorance helps expose all the buried assumptions that someone experienced in the domain takes for granted. Who has not observed the phenomenon that the one who seems to know the least about a problem seems to come up with the best solutions in a brainstorming session?

This observation leads to the suggestion that there may be some software development activities which are aided by the presence of some degree of ignorance. A new hire in an organization or project could use his or her ignorance about the domain of a system under development to perform tasks of the development that are helped or at least not hindered by his or her ignorance.

The following quotation by P. Burkinshaw, an attendee of the Second NATO Conference on software engineering in Rome in 1969 (Buxton and Randell, 1969) helps in better understanding the role played ignorance in software development.

"Get some intelligent ignoramus to read through your documentation and try the system; he will find many `holes' where essential information has been omitted. Unfortunately intelligent people do not stay ignorant too long, so ignorance becomes a rather precious resource."

However, that ignorance of the domain is helpful for any activity does not preclude that also expertise in the same domain is helpful for the same activity.

For each software development tasks given on subsequent pages, give your estimate of how helpful each of domain ignorance and domain awareness is in performing the task. The estimates range from "Required" through "Prevents".

If you find that you want to give more than one answers to any question because its activity has subparts for which different answers apply, then see questions 47-50 at the end.

1. This information will be used ONLY for a possible follow up should it prove necessary. Once the research is complete, it will be destroyed. Moreover, it will not show up in any analysis report.

Email Address:

*** 2. The organization at which you got most of your experience is**

- Commercial
- Research
- An Open Source Consortium
- Other (please specify)

*** 3. The organization at which you got most of your experience produces software:**

- Yes.
- No.

Role of Ignorance in Software Development Activities

*** 4. The organization at which you got most of your experience is in what country?**

*** 5. Years of experience in software development.**

*** 6. Years of experience managing software development.**

2.

Architectural Tasks

*** 7. Designing and specifying software architecture.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 8. Reviewing software architecture.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Requirements Engineering Tasks

*** 9. Eliciting requirements/Requirements gathering.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 10. Analyzing Requirements.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 11. Specifying requirements.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 12. Validating requirements.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Role of Ignorance in Software Development Activities

*** 13. Reusing and managing requirements.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Configuration Management Tasks

*** 14. Source/Version control tasks.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 15. Managing different builds of a software.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

3.

Deployment Tasks

*** 16. Deployment planning.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Risk Management Tasks

*** 17. Identifying project risks.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 18. Risk planning, monitoring and control.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Design Tasks

*** 19. Creating low-level software design.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Role of Ignorance in Software Development Activities

*** 20. Creating high-level software design.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 21. User Interface design.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 22. Identifying design and implementation rationale.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

4.

Coding/Implementation Tasks

*** 23. Coding simple/less complex features.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 24. Fixing bugs.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 25. Developing unit test cases.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 26. Developing white box test cases.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 27. Developing integration test cases.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Role of Ignorance in Software Development Activities

*** 28. Other coding oriented tasks.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 29. Determining source of a bug.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Testing/Quality Assurance Tasks

*** 30. Test planning for a release.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 31. Developing black box test cases.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 32. Developing system/performance test cases.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 33. Manually executing test cases developed by someone else.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 34. Automating test cases.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 35. Analyzing defects to find common trends.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

5.

Security Engineering Tasks

Role of Ignorance in Software Development Activities

*** 36. Identifying security risks.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 37. Preventing security threats.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

User Support Tasks

*** 38. Providing technical support to users.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 39. Writing user manual and release notes.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Reviewing/Inspection Tasks

*** 40. Inspecting code.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 41. Reviewing trace information (source code change history, bug history).**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 42. Inspecting/Reviewing design documents.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 43. Inspecting/Reviewing user manuals.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Role of Ignorance in Software Development Activities

*** 44. Inspecting/Reviewing test plans.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 45. Inspecting/Reviewing requirements document.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Attending Activities

*** 46. Attending courses/trainings/tech talks.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 47. Attending formal project meetings.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 48. Attending code/project walkthroughs.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

6.

Miscellaneous Activities

*** 49. Reading product documentation, user manuals and other design documents.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 50. Learning various processes/practices/technology used in the project.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

*** 51. Compiling the project code.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Role of Ignorance in Software Development Activities

*** 52. Installing and configuring the development environment.**

	Required	Enhances	Neutral	Impedes	Prevents
Domain Ignorance	○	○	○	○	○
Domain Awareness	○	○	○	○	○

Additional Activities If any of the tasks has a subpart not defined here for which you would have given a different answer, then enter its parent task number and its rating in the columns below. Use the space below also for any software development activities that you know of, but were not mentioned in this survey.

53. List an additional task and its rating here.
Example:
Task Name - Parent task number(If present)
Domain Ignorance - Rating(example-Neutral)
Domain Awareness - Rating(example-Required)

54. List an additional task and its rating here.
Example:
Task Name - Parent task number(If present)
Domain Ignorance - Rating(example-Neutral)
Domain Awareness - Rating(example-Required)

55. List an additional task and its rating here.
Example:
Task Name - Parent task number(If present)
Domain Ignorance - Rating(example-Neutral)
Domain Awareness - Rating(example-Required)

56. List an additional task and its rating here.
Example:
Task Name - Parent task number(If present)
Domain Ignorance - Rating(example-Neutral)
Domain Awareness - Rating(example-Required)

Appendix B

Ethics Application

ORE OFFICE USE ONLY

ORE #_____

APPLICATION FOR ETHICS REVIEW OF RESEARCH INVOLVING HUMAN PARTICIPANTS

Please remember to PRINT AND SIGN the form, and forward TWO copies to the Office of Research Ethics, Needles Hall, Room 1024, with all attachments.

A. GENERAL INFORMATION

1. Title of Project: The impact of ignorance(lack of domain knowledge) on various software development activities.

2. a) Principal and Co-Investigator(s)

Name	Department	Ext:	e-mail:

2. b) Collaborator(s)

Name	Department	Ext:	e-mail:

3. Faculty Supervisor(s)

Name	Department	Ext:	e-mail:
Dr. Daniel Berry	Computer Science, School of		dberry@uwaterloo.ca

4. Student Investigator(s)

Name	Department	Ext:	e-mail:	Local Phone #:
Gaurav Mehrotra	Computer Science, School of	33509	gmehrotr@uwaterloo.ca	2262209362

5. Level of Project: MMath **Specify Course:**

Research Project/Course Status: New Project\Course

6. Funding Status (if there is an industry sponsor and procedures pose greater than minimal risk, then Appendix B is to be completed):

Is this project currently funded? No

- If No, is funding being sought OR if Yes, is additional funding being sought? No
- Period of Funding:

7. Does this research involve another institution or site? No
If Yes, what other institutions or sites are involved:

8. Has this proposal been, or will it be, submitted to any other Research Ethics Board/Institutional Review Board? No

9. For Undergraduate and Graduate Research:

Has this proposal received approval of a Department Committee? Not Dept. Req.

10. a) Indicate the anticipated commencement date for this project: 11/15/2010

b) Indicate the anticipated completion date for this project: 8/31/2011

B. SUMMARY OF PROPOSED RESEARCH

1. Purpose and Rationale for Proposed Research

a. Describe the purpose (objectives) and rationale of the proposed project and include any hypothesis(es)/research questions to be investigated. For a clinical trial/medical device testing summarize the research proposal using the following headings: Purpose, Hypothesis, Justification, and Objectives. Where available, provide a copy of a research proposal. For a clinical trial/medical device testing a research proposal is required:
The goal of this research is to empirically study the impact of ignorance/lack of domain knowledge on various software development activities. The research will be conducted with the help of a web based survey listing various software development activities and participants will be asked to rate the various activities on the basis of level of ignorance involved in doing various activities. Specifically we are interested in answering two main research questions: is there a relation between the ramp up time for newcomers and the kind of software development activities they are involved in and the role ignorance plays in various software development activities?

b. In lay language, provide a one paragraph (approximately 100 words) summary of the project including purpose, the anticipated potential benefits, and basic procedures used.
The goal of the project is to study the impact of ignorance/lack of domain knowledge on various software development activities. The study will be conducted with the help of web survey which will be sent to the participants by email. The survey lists various software development activities and the participants are asked to rate each activity on the basis of level of ignorance needed while doing the various activities.

The participants will be benefited with understanding the role ignorance plays in software development which in turn can be used to increase the productivity of the entire team.

C. DETAILS OF STUDY

1. Methodology/Procedures

a. Indicate all of the procedures that will be used. Append to form 101 a copy of all materials to be used in this study.

Survey(s) or questionnaire(s) (mail-back) All are standardized.

b. Provide a detailed, sequential description of the procedures to be used in this study. For studies involving multiple procedures or sessions, provide a flow chart. Where applicable, this section also should give the research design (e.g., cross-over design, repeated measures design).
The participants will be asked to fill out a web survey which will be administered through SurveyMonkey. The survey link will be sent to the participants through email and they will be asked to respond by a specific date. The email will include a cover letter(attached) which describes the purpose of the study and potential benefits to the participants.

c. Will this study involve the administration/use of any drug, medical device, biologic, or natural health product? No

2. Participants Involved in the Study

a. Indicate who will be recruited as potential participants in this study.
Non-UW Participants:
Adults

b. Describe the potential participants in this study including group affiliation, gender, age range and any other special characteristics. Describe distinct or common characteristics of the potential participants or a group (e.g., a group with a particular health condition) that are relevant to recruitment and/or procedures (e.g., A group with asbestosis is included. People with this condition tend to be male, 50+ years, worked with

asbestos.). If only one gender is to be selected for recruitment, provide a justification for this.

The participants will typically be people holding the title of Software Development Managers, Senior Software Engineers having around 5-30 years of work experience in software industry. People belonging to both genders will be used and their typical age will be in between 30-75 years.

c. How many participants are expected to be involved in this study? For a clinical trial, medical device testing, or study with procedures that pose greater than minimal risk, sample size determination information is to be provided, as outlined in Guidance Note C2c.

~50

3. Recruitment Process and Study Location

a. From what source(s) will the potential participants be recruited?
Businesses, industries

b. Describe how and by whom the potential participants will be recruited. Provide a copy of any materials to be used for recruitment (e.g. posters(s), flyers, cards, advertisement(s), letter(s), telephone, email, and other verbal scripts).
Recruitment will be done through email using the cover letter attached.

c. Where will the study take place? On campus: On World Wide Web via Survey Monkey.

4. Remuneration for Participants
Will participants receive remuneration (financial, in-kind, or otherwise) for participation? No

5. Feedback to Participants

Describe the plans for provision of study feedback and attach a copy of the feedback letter to be used. Wherever possible, written feedback should be provided to study participants including a statement of appreciation, details about the purpose and predictions of the study, restatement of the provisions for confidentiality and security of data, an indication of when a study report will be available and how to obtain a copy, contact information for the researchers, and the ethics review and clearance statement.
Refer to the Checklist for Feedback Sheets on ORE web site:
http://iris.uwaterloo.ca/ethics/human/application/samples/checklistfeedback.htm
Each participant will be provided with a copy of feedback letter including the summary of the study and its anticipated results. The final results will be sent to them as soon as they are available.

D. POTENTIAL BENEFITS FROM THE STUDY

1. Identify and describe any known or anticipated direct benefits to the participants from their involvement in the project.
As our participants will typically be software development managers therefore they will be able to utilize the results of our study in order to assign appropriate roles to new hires in their team. Assigning the right roles to new hires in a team helps increase the team's productivity which in turn benefits the participants.

2.Identify and describe any known or anticipated benefits to the scientific community/society from the conduct of this study.
The study will significantly contribute to the body of knowledge on knowledge transfer, newcomer integration in organizations, and information needs in software engineering maintenance tasks.

E. POTENTIAL RISKS TO PARTICIPANTS FROM THE STUDY

1. For each procedure used in this study, describe any known or anticipated risks/stressors to the participants. Consider physiological, psychological, emotional, social, economic risks/stressors. A study–specific current health status form must be included when physiological assessments are used and the associated risk(s) to participants is minimal or greater.
No known or anticipated risks

Appendix C

Cover Letter

Cover Letter

Dear _____,

You are invited to participate in a research study conducted by **Gaurav Mehrotra**, under the supervision of **Dr. Daniel Berry, School of Computer Science** of the University of Waterloo, Canada. The objective of the research study is to empirically study the impact of ignorance or lack of domain knowledge on various software development activities. The study is for a Master's project.

If you decide to volunteer, you will be asked to complete a 20-minute online survey. Survey questions focus on various software development activities. Participation in this study is voluntary. You may decline to answer any questions that you do not wish to answer and you can withdraw your participation at any time by not submitting your responses. There are no known or anticipated risks from participating in this study.

It is important for you to know that any information that you provide will be confidential. All of the data will be summarized and no individual could be identified from these summarized results. Furthermore, the web site is programmed to collect responses alone and will not collect any information that could potentially identify you (such as machine identifiers).

This survey uses Survey Monkey™ whose computer servers are located in the USA. Consequently, USA authorities under provisions of the Patriot Act may access this survey data. If you prefer not to submit your data through Survey Monkey™, please contact one of the researchers so you can participate using an alternative method (such as through an email or paper-based questionnaire). The alternate method may decrease anonymity but confidentiality will be maintained.

If you wish to participate, please visit the survey link at
http://www.surveymonkey.com/s/2ZNHZHD.

The data, with no personal identifiers, collected from this study will be maintained on a password-protected computer database in a restricted access area of the university. As well, the data will be electronically archived after completion of the study and maintained for two years and then erased.

Should you have any questions about the study, please contact either Gaurav Mehrotra at 1-519-888-4567 ext. 33509 or by email at *gmehrotr@uwaterloo.ca* or Dr. Daniel Berry by email at *dberry@uwaterloo.ca*. Further, if you would like to receive a copy of the results of this study, please contact either investigator.

I would like to assure you that this study has been reviewed and received ethics clearance through the Office of Research Ethics at the University of Waterloo. However, the final decision about participation is yours. If you have any comments or concerns resulting from your participation in this study, please feel free to contact Dr. Susan Sykes, Director, Office of Research Ethics, at 1-519-888-4567 ext. 36005 or by email at *ssykes@uwaterloo.ca* .

Thank you for considering participation in this study.

References

[1] Andrew Begel and Beth Simon. Novice software developers, all over again. In *Proceeding of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 3–14, New York, NY, USA, 2008. ACM.

[2] Andrew Begel and Beth Simon. Struggles of new college graduates in their first software development job. *SIGCSE Bull.*, 40:226–230, March 2008.

[3] Daniel M. Berry. The importance of ignorance in requirements engineering. *J. Syst. Softw.*, 28:179–184, February 1995.

[4] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley, 1975.

[5] John N. Buxton and Brian Randell. Software engineering techniques: Report on a conference, 1969. `http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF`.

[6] Jeffrey C. Carver, Nachiappan Nagappan, and Alan Page. The impact of educational background on the effectiveness of requirements inspections: An empirical study. *Software Engineering, IEEE Transactions on*, 34(6):800–812, Nov.-Dec. 2008.

[7] Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard, and Jacqueline P. de Vries. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 275–284, New York, NY, USA, 2010. ACM.

[8] Tom DeMarco and Tim Lister. *Peopleware: Productive Projects and Teams.* Dorset House Publishing, 1987.

[9] Keren Kenzi, Pnina Soffer, and Irit Hadar. The role of domain knowledge in reqirements elicitation: An exploratory study. In *Proceedings of the Fifth Mediterranean Conference on Information Systems (MCIS)*, 2010. `http://aisel.aisnet.org/mcis2010/48/`.

[10] Andrew J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Robert G. Newcombe. Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in Medicine*, 17(8):873–890, 1998.

[12] Robert G. Newcombe. Two-sided confidence intervals for the single proportion: comparison of seven methods. *Statistics in Medicine*, 17(8):857–872, 1998.

[13] Edgar H. Schein. The individual, the organization, and the career: A conceptual scheme. *The Journal of Applied Behavioral Science*, 7(4):401–426, 1971.

[14] Susan E. Sim and Ric C. Holt. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, pages 361–370, April 1998.

[15] Wikipedia. Likert scale, Viewed 1 May 2011. `http://en.wikipedia.org/wiki/Likert_scale`.