Role of Domain Ignorance in Software Development: Software Development Tasks Benefiting from Domain Ignorance

Gaurav Mehrotra and Daniel M. Berry
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
gaurav.iiita@gmail.com, dberry@uwaterloo.ca

***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 paper reports work putting the observation to an empirical test. A survey was conducted among software development managers of varying experience to determine what software development activities they thought are at least helped by domain ignorance. A companion paper subjects this list to an additional empirical test.**

***Keywords*-assigned software engineering tasks, creative ideas, domain awareness, domain ignorance, importance of ignorance, questionnaire, tacit assumptions**

I. Introduction

This paper is the first of two companion papers that explore the role of domain ignorance in software development. This first paper determines, with the help of a survey conducted among software development managers of varying experience, what software development activities they thought are at least helped by ignorance of the domain of the software system under development, hereinafter called simply "domain ignorance". The opposite of domain ignorance is domain awareness. Moreover, it is assumed that ignorance in the development personnel is restricted to the domain of the software system under development. That is, it is assumed that all personnel involved in the development are competent in the skills required for their tasks in the development.

The second paper [1] examines transcripts from fourteen interviews of presumably-domain-ignorant immigrants to new software development projects at one large company to determine if the activities performed by those with the most successful immigrations were activities that are at least helped by domain ignorance.

Domain ignorance 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. Domain ignorance is believed to help one to avoid the domain's tacit assumptions and to think outside of the domain's box [2] [3, p. 18]. The right kind of ignorance helps evoke questions that 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? Perhaps, there are some software development activities that are aided by some degree of domain ignorance.

A review of the existing literature (See Section II) shows that some activities seem to yield better results when performed by a domain ignorant than by a domain expert.

The main contribution of this paper is the gathering of data about the effects of domain 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.

Specifically, this survey aimed to answer one important research question:

RQ1: Are there software development activities that are helped by domain ignorance?

Related work is discussed in Section II. Section III presents the survey design. The results of the survey are discussed in Section IV. Section V discusses the threats to the conclusions in Section IV. Section VI describes applications of the results, and Section VII summarizes the conclusions and discusses future work.

II. 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 application while being very ignorant of the domain [2]. P. Burkinshaw, an attendee of the Second NATO Conference on software engineering in Rome in 1969 [3, 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. [4] 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. However, 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.

Fred Brooks suggested that 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 [5]. Also, the right mix of people to do an activity is necessary to get the best results.

Kenzi, Soffer, and Hadar conducted an exploratory study of the perception of requirements analysts of the role of domain knowledge in requirements elicitation [6]. Their study identified both positive and negative effects of domain knowledge on requirements elicitation. Their conclusions suggest the possibility of forming requirements elicitation teams with of analysts with different amounts of domain knowledge, that the role played by an analysts can depend on his[1] 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.

III. Survey Design

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

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 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.

A five-point ordinal scale was used to categorize the importance of or the effect of domain ignorance or domain awareness on any software development activity. An ordinal scale was implemented, as the goal of this research was 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 [7]. 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 paper, domain awareness and domain ignorance are collectively referred to as kinds of *domain familiarity*.

A. 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/, the Website of the OPEN Process Framework (OPF). The activities relating to a particular aspect of software development such as testing, architecting, etc. were grouped together. 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 the Appendix of Mehrotra's master's thesis [8] for the final list of activities chosen.

Mehrotra decided that the participant pool for the survey should be limited to people having significant experience managing software development. So, he used a snowball sampling based on judgemental sampling [9], 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

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

levels helped to bring some randomness to the participant pool.

Participants were contacted through e-mail. The e-mail invitations included a brief description of the research along with a link to the online survey hosted by SurveyMonkey™. After a reminder, a total of 40 respondents had completed the survey. See the Appendix of Mehrotra's thesis for the e-mailed invitation and the complete survey.

IV. Results and Data Analysis

This section discusses the results obtained in this study.

A. Survey Respondents

A total of 40 respondents completed the online survey. Respondents came from different countries like India, the United States, the United Kingdom, Canada, and Israel. 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 1. More than half the respondents had at least 10 years of 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 2. More than half the respondents had at least 5 years of experience managing software development.

B. Analysis of Results

Due to the small sample size, the test of proportions [10] [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 Likert scale data, but the distance between scale values is not clear. Therefore, mode was used as the key data measure as both average or median are less meaningful for this type of data.

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

1) helped by domain ignorance, in Section IV.C,
2) unaffected by domain ignorance, in Section IV.D, and
3) hindered by domain ignorance, in Section IV.E.

For any activity, e.g., as in Figure 3, 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 test of proportions, corresponding to the five values, "required", "enhances", "neutral", "impedes", and "prevents". A 5% error margin was chosen, which is represented by a p-value of 0.05. The tests for proving the statistical validity of obtained results were omitted for activities not affected by domain ignorance and for activities hindered by domain ignorance, because the aim of this research was to find activities that are helped by domain ignorance.

C. 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".

Below is the analysis of the survey data about one example task, namely that called "Eliciting Requirements/ Requirements Gathering":

Task: Eliciting Requirements/Requirements Gathering
Distribution: as shown in Figure 3
Modes:

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

Output of test of proportions for domain ignorance: as shown in Table I
p-value = 1.726e-07
Verdict: Since p-value < 0.05, the results are statistically significant.

Space constraints dictate that these analyses, which can be found in Chapter 4 of Mehrotra's thesis, be replaced by a single table, Table II, summarizing the analyses. Note that all expected proportions are identically .20; so they are not shown. Also, since each p-value in the table is less than .05, each conclusion about the mode of domain ignorance is statistically significant.

The results of the analysis say that 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.

There were a number of unexpected surprises in the results. For example, 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 surprise was with the activity: "Inspecting/Reviewing Requirements Documents". Its mode of domain ignorance is "enhances" while its mode 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.

D. Activities Not Affected by Domain Ignorance

This section lists software development activities that are thought by the respondents not to be affected by domain ignorance, i.e., whose domain ignorance mode is "neutral". Since none of these activities is regarded as helped by domain ignorance, as explained in Section IV.B, Mehrotra did not calculate the statistical significance of the mode of domain ignorance. Hence, the summarizing Table III has only three columns, giving only tasks and their two modes.

E. Activities Hindered by Domain Ignorance

This section lists software development activities that are thought by the respondents to be *hindered* by domain ignorance, i.e., whose domain ignorance mode is one of the "hinders" scale values, "impedes" or "prevents". Since none of these activities is regarded as helped by domain ignorance, as explained in Section IV.B, Mehrotra did not calculate the statistical significance of the mode of domain ignorance. Hence, the summarizing Table IV has only three columns, giving only tasks and their two modes.

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.

V. THREATS

The conclusion of this paper is the classification of software development activities according to whether they are helped by, hindered by, or unaffected by domain ignorance. The threats to the validity of this conclusion can be divided into two classes,

1) threats to internal validity that concerns how well the survey was executed and
2) threats to external validity that concerns whether the conclusion obtained is generalizable.

The threats to internal validity of the conclusions are:

- **the bias of the chosen sampling method**:
  Snowball sampling is believed to produce highly biased results [12]. 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?**
    Mehrotra 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?**:

That there were no incomplete questionnaires and that the answers to related questions in any one response were highly consistent with each other indicate 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 she is about her 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.

A deeper question is whether a survey can be used to test a question of fact. Any survey can report only the opinions of the participants. Left unanswered is whether the participants' opinions reflect reality. We really desire to know which software development activities are at least helped by domain ignorance. A proper test would be to conduct experiments with large numbers of software developers with varying domain familiarities doing a large variety of software development activities. Each participant's performance on each activity would be evaluated by a team of quality assurers. All of the data would be subjected to an ANOVA test to determine the effect of domain ignorance on each kind of software development activity. Such a study would be enormously expensive and nearly impossible to control and would be seriously challenged for external validity, because the necessarily small sizes of the artifacts to be able to do a controlled experiment. Therefore, as in many similar situations, we need to rely on surveys reporting opinions of people. The survey was designed to target people who are in a position to have made accurate observations of the phenomenon being tested.

VI. Applications of Results

One use of the results is to help assign a good mix of people for a software development activity. For example, a task that is helped by both domain ignorance and domain awareness should be done a team consisting of both domain ignorants and domain awares.

The companion paper [1] describes a use of these results to determine how the presumed domain ignorance of a new member of a software system development team affects his immigration into the team.

Table I
Test of Proportions Output

	Proportion of Responses for Each Likert Value				
	Required	Enhances	Neutral	Impedes	Prevents
expected	0.2	0.2	0.2	0.2	0.2
observed	0.150	0.550	0.050	0.125	0.125

Table III
Activities Not Affected by Domain Ignorance

	Modes of Domain ...	
Task	Ignorance	Awareness
Learning Processes/ Practices/Technology Used	Neutral	Enhances
Source/Version Control Tasks	Neutral	Required
Coding Simple Features	Neutral	Required
Other Code Oriented Tasks	Neutral	Enhances
Automating Test Cases	Neutral	Enhances
Reviewing Trace Information	Neutral	Enhances
Attending Courses/Trainings	Neutral	Enhances
Attending Formal Project Meetings	Neutral	Enhances
Attending Code/ Project Walkthroughs	Neutral	Required
Compiling Project Code	Neutral	Neutral
Installing and Configuring Development Environment	Neutral	Enhances

VII. Conclusions and Future Work

This research highlights the importance of domain ignorance in various software development activities. A 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 the personnel in her team.

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 questions. A survey's 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.

Acknowledgements

The authors thank Chrysanne DiMarco, Mike Godfrey, and Jo Atlee for their comments on the thesis on which this paper is based.

References

[1] G. Mehrotra and D. M. Berry, "Role of domain ignorance in software development: How domain ignorance helps immigration to software development projects," School of Computer Science, University of Waterloo, Tech. Rep., 2012, http://se.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/MehrotraBerryNewbie.pdf.

Table II
ACTIVITIES HELPED BY DOMAIN IGNORANCE

Task	Modes of Domain ...		Observed Proportions					p-value
	Ignorance	Awareness	Required	Enhances	Neutral	Impedes	Prevents	
Eliciting Requirements/Requirements Gathering	Enhances	Enhances	0.150	0.550	0.050	0.125	0.125	1.726e-07
Analyzing Requirements	Enhances	Required	0.150	0.550	0.025	0.200	0.075	4.033e-08
Identifying Project Risks	Enhances	Required	0.0512	0.5128	0.0512	0.3076	0.0769	8.375e-08
Creating High Level Software Design	Enhances	Required	0.0512	0.3846	0.2307	0.2051	0.1282	0.009571
User Interface Design	Enhances	Enhances	0.1538	0.4358	0.1282	0.1025	0.1794	0.003268
Developing Black Box Test Cases	Enhances	Enhances	0.1578	0.5000	0.1578	0.1052	0.0789	3.931e-05
Analyzing Defects to Find Common Trends	Enhances	Enhances	0.0789	0.5000	0.1578	0.2105	0.0526	1.197e-05
Identifying Security Risks	Enhances	Required	0.0526	0.4736	0.1052	0.1578	0.2105	0.0001102
Writing User Manuals and Release Notes	Enhances	Required	0.1842	0.4736	0.0526	0.1315	0.1578	0.0001710
Inspecting/Reviewing Design Documents	Enhances	Required	0.1315	0.5526	0.1052	0.1315	0.0789	5.052e-07
Inspecting/Reviewing User Manuals	Enhances	Enhances	0.1666	0.6000	0.1000	0.1000	0.0333	2.198e-07
Inspecting/Reviewing Test Plans	Enhances	Enhances	0.2	0.6	0.1	0.1	0.0	8.353e-08
Inspecting/Reviewing Requirements Document	Enhances	Required	0.166	0.6333	0.1333	0.0000	0.0666	5.463e-09
Reading Product Documentation	Enhances	Enhances	0.0789	0.5263	0.1578	0.1578	0.0789	3.608e-06

Table IV
ACTIVITIES HINDERED BY DOMAIN IGNORANCE

Task	Modes of Domain ...	
	Ignorance	Awareness
Designing and Specifying Software Architecture	Impedes	Required
Reviewing Software Architecture	Impedes	Required
Specifying Requirements	Impedes	Required
Validating Requirements	Impedes	Required
Reusing and Managing Requirements	Impedes	Required
Managing Builds of a Software	Impedes	Required
Deployment Planning	Impedes	Required
Risk Planning/Monitoring and Control	Impedes	Required
Creating Low Level Software Design	Impedes	Required
Identifying Design and Implementation Rationale	Impedes	Required
Fixing Bugs	Impedes	Required
Developing Unit Test Cases	Impedes	Required
Developing White Box Test Cases	Impedes	Required
Developing Integration Test Cases	Impedes	Required
Determining Source of a Bug	Impedes	Required
Test Planning for a Release	Impedes	*no mode*
Developing System/ Performance Test Cases	Impedes	Enhances
Manually Executing Test Cases	Impedes	Enhances
Preventing Security Threats	Impedes	Required
Providing Technical Support to Users	Impedes	Required
Inspecting Code	Impedes	Required

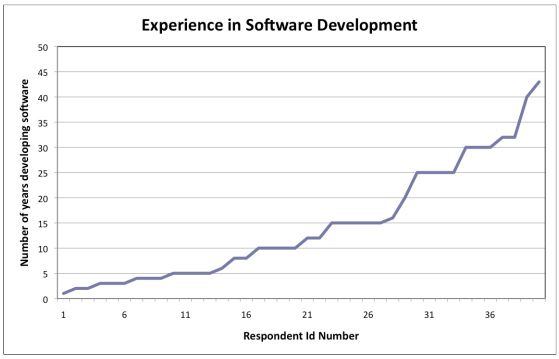

Figure 1. Experience in Software Development

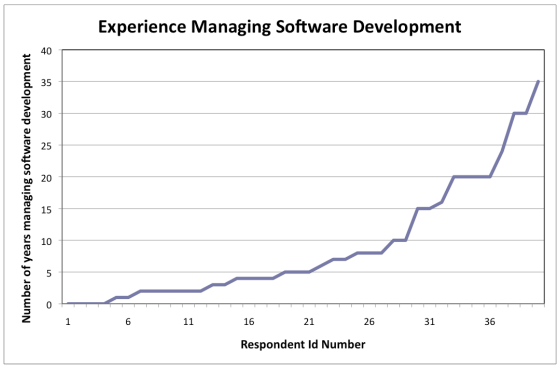

Figure 2. Experience Managing Software Development

[2] D. M. Berry, "The importance of ignorance in requirements engineering," *Journal of Systems and Software*, vol. 28, 1995.

[3] J. N. Buxton and B. Randell, "Software engineering techniques: Report on a conference," 1969, http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF.

[4] J. C. Carver, N. Nagappan, and A. Page, "The impact of educational background on the effectiveness of requirements inspections: An empirical study," *IEEE Transactions on Software Engineering*, vol. 34, no. 6, pp. 800–812, 2008.

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

[6] K. Kenzi, P. Soffer, and I. 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/.

[7] Wikipedia, "Likert scale," Viewed 1 September 2011, http://en.wikipedia.org/wiki/Likert_scale.

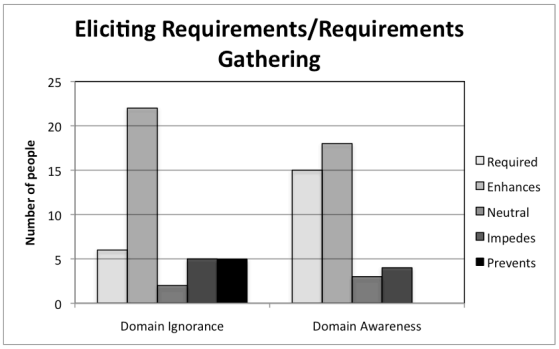

Figure 3. Distribution for "Eliciting Requirements/Requirements Gathering"

[8] G. Mehrotra, "Role of domain ignorance in software development," Master's thesis, University of Waterloo, Waterloo, ON, Canada, 2011, http://se.uwaterloo.ca/~dberry/FTP_SITE/students.theses/gaurav.mehrotra/gauravMehrotraThesis.pdf.

[9] Wikipedia, "Nonprobability sampling," Viewed 1 September 2011, http://en.wikipedia.org/wiki/Nonprobability_sampling.

[10] R. G. Newcombe, "Two-sided confidence intervals for the single proportion: Comparison of seven methods," *Statistics in Medicine*, vol. 17, no. 8, pp. 857–872, 1998.

[11] ——, "Interval estimation for the difference between independent proportions: Comparison of eleven methods," *Statistics in Medicine*, vol. 17, no. 8, pp. 873–890, 1998.

[12] Wikipedia, "Snowball sampling," Viewed 13 February 2012, http://en.wikipedia.org/wiki/Snowball_sampling.