# The Impact of Domain Knowledge on the Effectiveness of Requirements Idea Generation during Requirements Elicitation

Ali Niknafs and Daniel M. Berry

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

27 September 2012

# Outline

1. Introduction
   - Study
2. Methodology
   - Pilot Studies
3. Controlled Experiment
   - Design
   - Results

With some digressions for earlier and related work!

If time permits, some other cool stuff from earlier and related work!

# Introduction, Definition of RE

The process of arriving at a specification of a set of features that need to be developed is referred to as *requirements engineering (RE)*.

# The Role of People in RE

- Of the three Ps, *process, product, and people*, in software engineering, people have been least scrutinized.
- Boehm observed that the quality of the development personnel is the most powerful factor in determining an organization's software productivity.
- While there is empirical evidence of the importance of the quality of the personnel in software development, there is not much in RE.

# The Role of People in RE

The qualifications of the personnel involved in an RE process highly affects the effectiveness of the process, but most decisions about staffing RE teams arise from anecdotes and folklore, not from scientific studies.

# The RE Gap

- One issue in RE is the gap between what the customer wants and what the analyst thinks the customer wants.
- To bridge this gap, many believe that an analyst needs to know the customer's problem domain well to do RE well for a system in the domain.
- However, deep knowledge of the problem domain can lead to falling into the *tacit assumption tarpit*.

# Benefits of Domain Ignorance

The benefits of domain ignorance include:

- the ability to think out of the domain's box, leading to ideas that are independent of the domain assumptions,
- the ability to ask questions that expose the domain's tacit assumptions, leading to a common explicit understanding.

# First Observations of Benefits of Ignorance

In 1995, Berry observed the benefits of domain ignorance when he performed better than expected when he helped specify requirements for software in domains he was quite ignorant of.

This all started with earlier observations:

# Importance of Ignorance

**Based on:**

# Importance of Ignorance in Requirements Engineering

Daniel M. Berry

*Journal of Systems and Software*
28:2, 179–184, February, 1995

This, in turn, came out of an even earlier publication:

# Requirements Engineering (RE)

**“Programmer-Client Interaction in Writing Program Specifications” by Daniel M. and Orna Berry was written in 1980.**

**One of the first papers on requirements engineering.**

# Ignorance Hiding -1

**This paper is about how Dan, in 1979, managed to write the best requirements document he had ever written for a statistics application needed by Orna, despite the fact that he was (and still is) totally ignorant in statistics.**

**This requirements document turned out to be totally resistant to requirements creep.**

# Ignorance Hiding -2

**Resistant to requirements creep?**

- **It did not have to be changed at all while the program was being written,**
- **it remained an accurate requirements document even through deployment, and**
- **it anticipated functionality that the client did not know she needed until later.**

**All this, even though Dan knew nothing about the application domain.**

# Ignorance Hiding -3

**The paper makes the point that *ignorance hiding* can be used to hide the requirement engineer's ignorance of the client's domain, by encapsulating that ignorance behind abstractions that can be taken as primitive.**

# Abstractions

**Buzz words**

- **nouns = types and objects**
- **verbs = functions and procedures**

# The Power of Abstractions

**These abstractions provide the tools Dan needed to produce a very good requirements document, the best he had ever written!**

# Practice

**Since the paper's publication, Orna and Dan have practiced ignorance hiding on a number of requirements engineering efforts.**

# RE Experiences -1

**Many times, though, there was not much ignorance to hide.**

- **Orna worked in her area of expertise, networking.**
- **Dan worked in his area of expertise, electronic publishing.**

# RE Experiences -2

**Both were quite satisfied with the general success of the method and would not use any other.**

**Orna, in industry, became known for her ability to get to the heart of requirements quickly and was in demand among several projects in the company for which she worked and in other companies.**

# Revealing Experience -1

**In 1995, Dan was called in as a consultant to help a start-up write requirements for a new multi-port Ethernet switching hub.**

**Dan protested that he knew nothing about networking and Ethernet beyond nearly daily use of telnet, ftp, and netfind.**

**At one point, earlier in his life, he worried that the ether in Ethernet cables might evaporate!**

# Dilbert's Ph.B.

# Revealing Experience -2

**The engineers in the start-up**

- **were almost exclusively hardware engineers**
- **were struggling 4 months to come up with a software requirements document and were getting nowhere fast**
- **knew the technology cold but not how to structure the software for it**

# Revealing Experience -3

**The engineers in the start-up (cont'd.)**

- **had not stated the requirements in full and were cycling with no convergence between requirements gathering and software design**

- **had a much stronger understanding of how to specify hardware, so that the hardware part of the project was on schedule but the software part was way behind schedule**

# Revealing Experience -4

**Dan asked each person to supply him with complete lists of the pieces of the system and of the features (operations) of each.**

**Dan read these and began to build abstractions.**

**Dan noticed lots and lots of inconsistencies.**

# Revealing Experience -5

**Dan asked lots and lots of questions and nudged for resolutions of all inconsistencies during a 2-hour meeting.**

**Dan worked for 4 more hours to produce a first draft specification that seemed to have electrified the engineers.**

# Revealing Experience -6

**That is, in 6 hours, Dan had put down in words and diagrams what the engineers had been trying to say in 4 months**

**Dan continued to work over 2 more months to produce a functional specification and an architectural specification that were carefully maintained to be consistent.**

# Ignorance is the Key -1

**While Dan was lecturing at CMU on ignorance hiding (mentioned earlier), one student, Jim Alstad, remarked that maybe the very fact that Dan knew so little about Orna's application area had been a significant factor in the success of the first experience.**

# Ignorance is the Key -2

**By being ignorant of the application area, Dan was able to avoid falling into the tacit assumption tarpit!**

**The 1995 experience seems to confirm the importance of the ignorance that ignorance hiding is so good at hiding.**

# Ignorance is the Key -3

**It was clear to Dan that the main problem preventing the engineers at the start-up from coming together to write a requirements document was that**

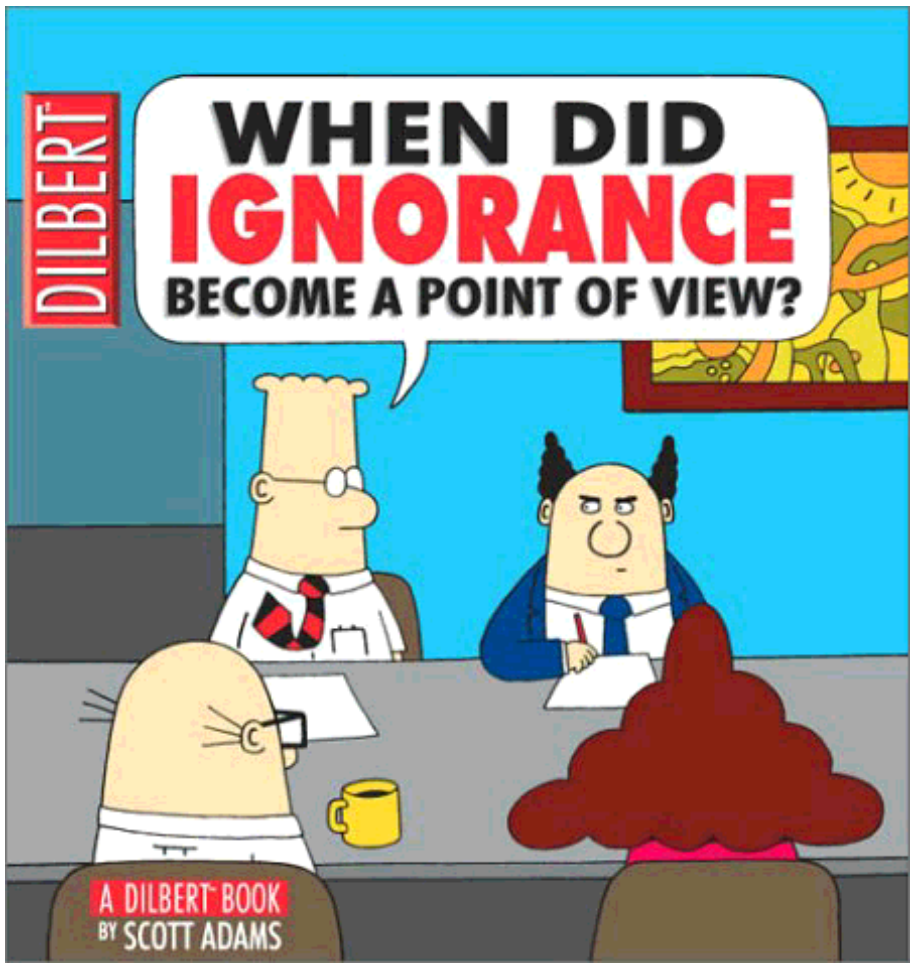
- **all were using the same vocabulary in slightly different ways,**
- **none was aware of any other's tacit assumptions, and**
- **each was wallowing deep in his own pit.**

# Ignorance is the Key -4

**Dan's lack of assumptions forced him**

- **to ferret out these assumptions and**
- **to regard the ever so slight differences in the uses of some terms as inconsistencies.**

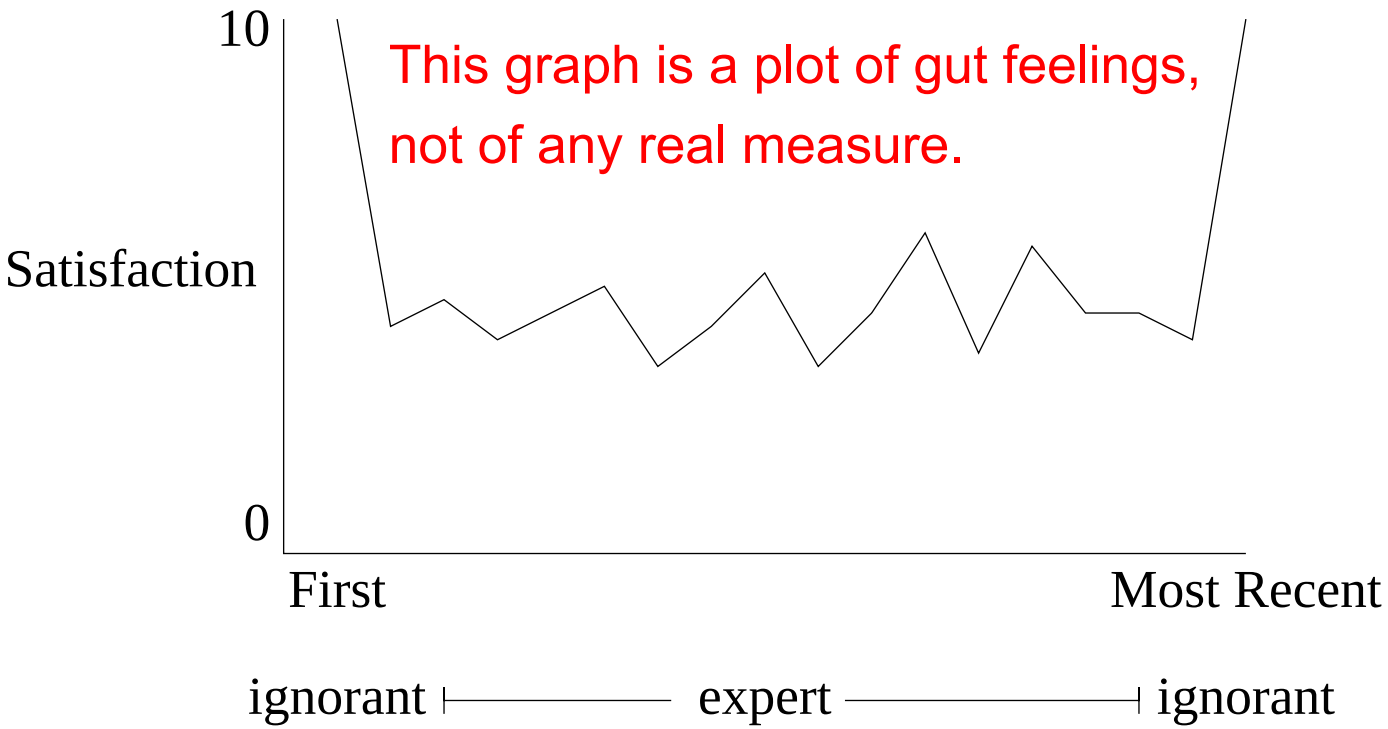
# Ignorance A Point of View?



# Retrospective -1

**Looking back over the history of applying ignorance hiding, Dan and Orna observed that the first and the most recent applications were the most successful in terms of their own satisfaction with the results.**

# Retrospective -2




History of Ignorance Hiding Experiences

# Retrospective -3

**In the first and most recent cases, the requirement engineer was ignorant of the domain.**

**In all other cases, the requirements engineer was in his or her field of expertise.**

# Need Ignorance

**Our conclusion is that every requirements engineering team requires a person who is ignorant in the application domain, the ignoramus of the team, who is not afraid to ask questions that show his or her ignorance, and who *will* ask questions about anything that is not entirely clear.**

# Still Need Experts

**We are not claiming that expertise is not needed.**

***Au contraire*, you cannot get the material in which to find inconsistencies without the experts.**

# Ignorance, Not Stupidity!

**We are not claiming that the ignoramus is stupid.**

***Au contraire*, he or she must be an expert in general software system structures and must be smart enough to catch inconsistencies in statements made by experts in fields other than his or her own.**

# Recommendations

**Each requirements engineering team needs**

- **at least one domain expert, usually supplied by the customer**
- **at least one smart ignoramus**

# Resumes of the Future

**Resumes of future software engineers will have a section proudly listing all areas of ignorance.**

**This is the only section of the resume that shrinks over time!**

**The software engineer will charge fees according to the degree of ignorance: the more ignorance, the higher the fee!**

# Meaning of "Ignoramus"

**Just to be clear, in the rest of this talk,**

**Not only is an ignoramus *not* stupid, he or she is a *competent professional* software engineer or requirements analyst.**

**The ignorance is simply of the *application domain* and *not* of computing and software engineering.**

**In fact, the *more* competent and professional, the *better* !**

# First Observations of Benefits of Ignorance

Getting back to the mainline talk!

Probably, the earliest observation of the benefits of ignorance was Burkinshaw's statement during the 1969 Second NATO Conference on Software Engineering:

> *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 don't stay ignorant too long, so ignorance becomes a rather precious resource. Suitable late entrants to the project are sometimes useful here.*

# Other Related Work

- Naggapan *et al* studied the impact of computer science educational background on requirements inspection effectiveness.
  - Inspectors who had a background that was unrelated to computing were significantly more effective in identifying defects.
- Kenzi *et al* conducted an exploratory study of the perceptions of requirements analysts of the role of domain ignorance in RE.

# Outline

# Context of the Study

In each experiment, subjects perform an RE task that generates things, such as requirement ideas for some *computer-based system (CBS)* for some *client*.

- The RE task that is done in an experiment is called a *generative task (GT)*. Example GTs are requirements elicitation and requirements document inspection.
- The unit generated by a GT is called a *desired generated unit (DGU)*. For the two example GTs, the DGUs are requirements ideas and defects in a requirements document.

# Context of the Study

- The CBS is situated in some *domain*, and at least one member of the client's organization is at least *aware of* and is often expert in this domain.
- Each member of the software development organization doing the RE activities has a different amount of *knowledge about the domain*. Each is either:
  - *Ignorant of the domain*, i.e., is a *domain ignorant (DI)*.
  - *Aware of the domain*, i.e., is a *domain aware (DA)*.
- Each of domain ignorance and domain awareness is a kind of *domain familiarity*.

# Research Questions

## Main Question

How does one form the most effective team, consisting of some mix of DIs and DAs, for a RE activity involving knowledge about the domain of the CBS whose requirements are being determined by the team?

## Elaborated Questions

- Does a mix of DIs and DAs perform a RE activity more effectively than only DAs?
- Do other factors impact the effectiveness of an individual in performing an RE activity?

# Hypothesis

## Main Hypothesis

A team consisting of a mix of DIs and DAs is more effective in an RE activity than is a team consisting of only DAs.

## Null Hypothesis

The mix of DIs and DAs in a team has no effect on the team's effectiveness in an RE activity.

# Outline

# Lessons Learned from Pilot Studies

1. Find a suitable problem domain.
2. Consider other factors (e.g. industrial experience) in analyzing the results.
3. Assess also the quality of the DGUs.
4. For many domains, so-called DIs turn out not to be real DIs, and so-called DAs turn out not to be real DAs.

# Lessons Learned from Pilot Studies

Lessons 1 and 4 taught us that we need a problem domain that *partitions* the set of subjects with precision into

- DAs
- DIs

with no one in between.

We thought very hard to find such a domain, bidirectional word processing:

- CSers from the Middle East are DAs.
- CSers from elsewhere are DIs.

# Outline

# Experiment Context

- *GT:* The first, idea-generation step in a brainstorming activity to generate requirement ideas for a CBS.
- *DGUs:* Requirement ideas
- *Domain:* Bidirectional word processing
- *Subjects:* Volunteer subjects were recruited from a "Software Requirements and Specification" course and from outside the course, but nevertheless in CS or a related discipline.
- *Teams:*
  - 3I: a team consisting of 3 DIs and 0 DAs,
  - 2I: a team consisting of 2 DIs and 1 DAs,
  - 1I: a team consisting of 1 DIs and 2 DAs,
  - 0I: a team consisting of 0 DIs and 3 DAs.

# Variables

- Independent Variables about a team
  - *Mix of Domain Familiarities*
  - *Creativity Level*
  - *RE Experience*
  - *Industrial Experience*
- Dependent Variable
  - *Effectiveness*

# Hypotheses

**$H_{11}$: The effectiveness of a team in requirements idea generation is affected by the team's mix of domain familiarities.**

$H_{10}$: The effectiveness of a team in requirements idea generation is not affected by the team's mix of domain familiarities.

**$H_{21}$: The effectiveness of a team in requirements idea generation is affected by the team's creativity level.**

$H_{20}$: The effectiveness of a team in requirements idea generation is not affected by the team's creativity level.

# Hypotheses

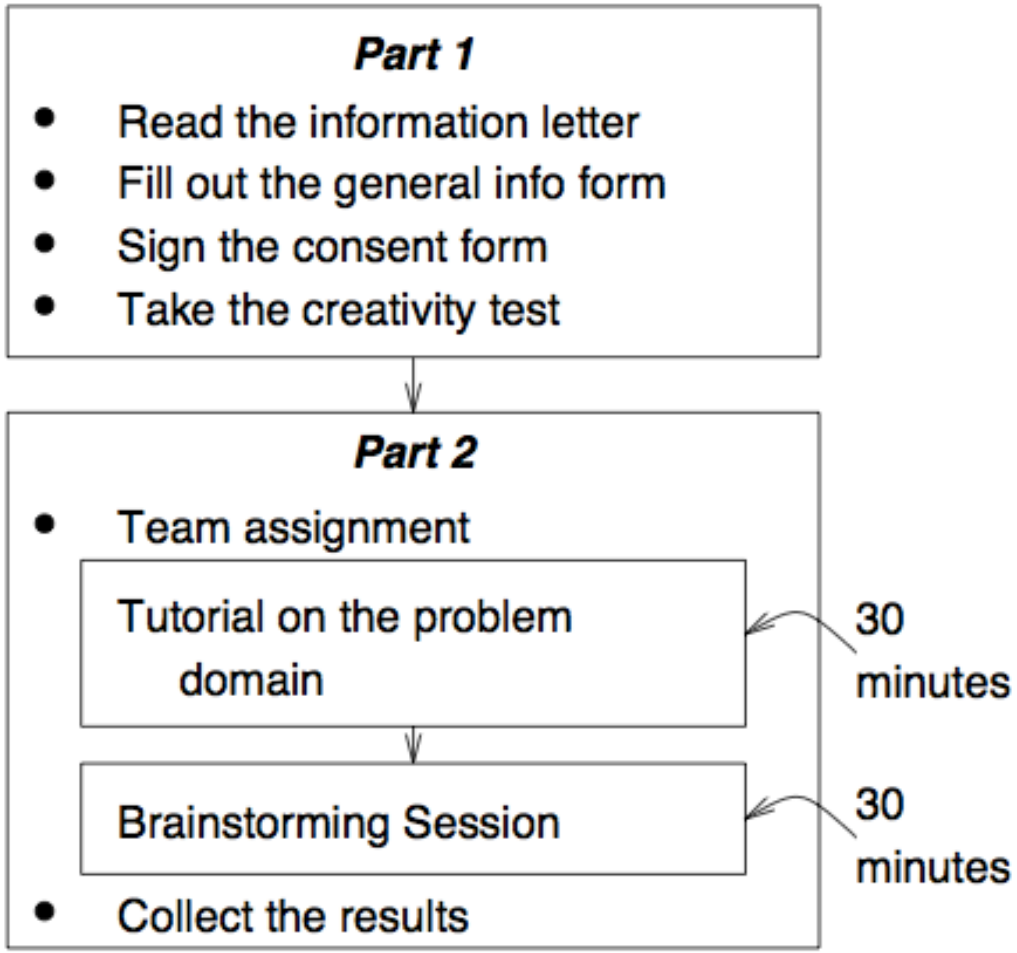
**$H_{31}$: The effectiveness of a team in requirements idea generation is affected by the team's RE experience.**

$H_{30}$: The effectiveness of a team in requirements idea generation is not affected by the team's RE experience.

**$H_{41}$: The effectiveness of a team in requirements idea generation is affected by the team's industrial experience.**

$H_{40}$: The effectiveness of a team in requirements idea generation is not affected by the team's industrial experience.

# Procedure


Part 1
Read the information letter
Fill out the general info form
Sign the consent form
Take the creativity test
Part 2
Team assignment
Tutorial on the problem domain
30 minutes
Brainstorming Session
30 minutes
Collect the results

# Evaluation of Generated Ideas

- The quantitative data is the number of raw ideas generated by each team, which is a good measure for the GT = brainstorming (because quantity is the *goal* of the first stage of brainstorming).
- To better compare the performance of the teams, Niknafs considered also the quality of their generated ideas.

# Quality of Generated Ideas

Based on the characteristics of a good requirement in the IEEE 830 Standard, each idea is classified according to three characteristics:

1. *Relevancy:* an idea is considered relevant if it has something to do with the domain.
2. *Feasibility:* an idea is considered feasible if it is relevant and it is correct, well presented, and implementable.
3. *Innovation:* an idea is considered innovative if it is feasible and it is not already implemented in an existing application for the domain known to the evaluator.

# Evaluation of Quality of Generated Ideas

- Berry and Niknafs evaluated the quality of the ideas since we were both experts in bidirectional word processing.
- To eliminate any bias in classifying an idea that might arise from the evaluator's knowing the domain familiarity mix of the team from which the idea came, Niknafs produced a list of all ideas generated by all teams, sorted using the first letters of each idea.
- Each domain-expert evaluator classified the ideas in the full list.
- After both evaluations were done, the each evaluator's classifications of each idea were transferred to the idea's occurrences in the individual team lists.

# Outline

# Results: Data About the Teams

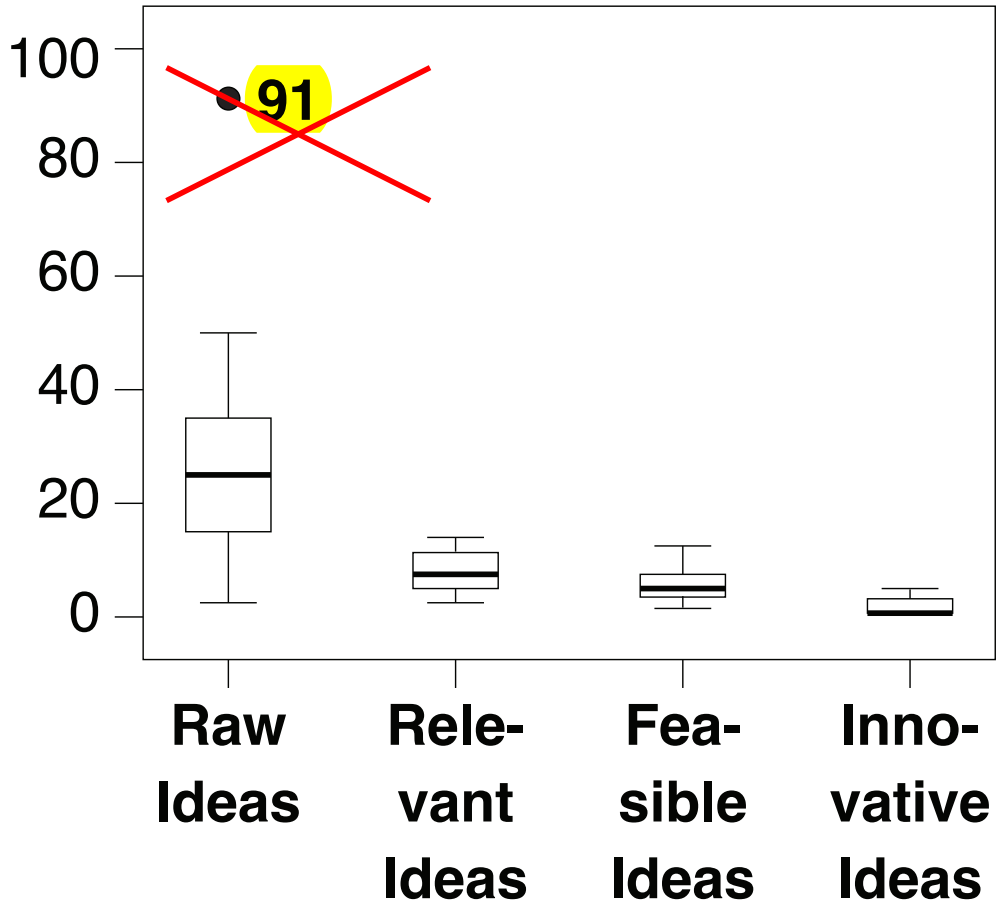
| ***Type of Teams*** | ***Number of Teams*** | ***Creativity*** | ***RE Experience*** | ***Industrial Experience*** |
|---|---|---|---|---|
| | | Mean | Mean | Mean |
| *3I* | 9 | 69.11 | 0.89 | 3.06 |
| *2I* | 4 | 71.75 | 0.75 | 3.33 |
| *1I* | 3 | 70.67 | 1.00 | 1.33 |
| *0I* | 3 | 71.33 | 1.00 | 2.00 |

We balanced teams by creativity, ignoring other variables.

Creativity & RE Experience turned out to have NO effect on effectiveness, but Industrial Experience DID, but in a surprising way!

# Outliers

- Boxplots were used to graphically expose any outliers.



# ANOVA Prerequisites

- The differences between the teams were determined by means of an analysis of variance (ANOVA).
- In order to be allowed to apply an ANOVA, the data must meet the three prerequisites for an ANOVA:
  1. *All dependent variables are normally distributed.*
  2. *All variances are homogeneous.*
  3. *All observations are independent.*

# ANOVA Prerequisites

- An ANOVA was applied to the dependent variables whose values met the prerequisites for an ANOVA; i.e. the numbers of generated raw, relevant, and feasible ideas.
- For innovative ideas, another, non-parametric test was used.

# ANOVA Results

| | Raw Ideas | | | | Relevant Ideas | | | | Feasible Ideas | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Effect | $F$ | $p$ | $f^2$ | $P$ | $F$ | $p$ | $f^2$ | $P$ | $F$ | $p$ | $f^2$ | $P$ |
| Mix of Domain Familiarities | .165 | .915 | .011 | .068 | 8.675 | **.032** | .319 | .816 | 13.486 | **.015** | .449 | .941 |
| Creativity | .921 | .469 | .048 | .146 | 3.918 | .114 | .159 | .459 | .984 | .449 | .051 | .153 |
| Industrial Experience | .563 | .609 | .031 | .107 | 10.089 | **.027** | .331 | .833 | 4.381 | .098 | .173 | .499 |
| RE Experience | .145 | .722 | .008 | .063 | .173 | .699 | .009 | .65 | .035 | .861 | .002 | .53 |

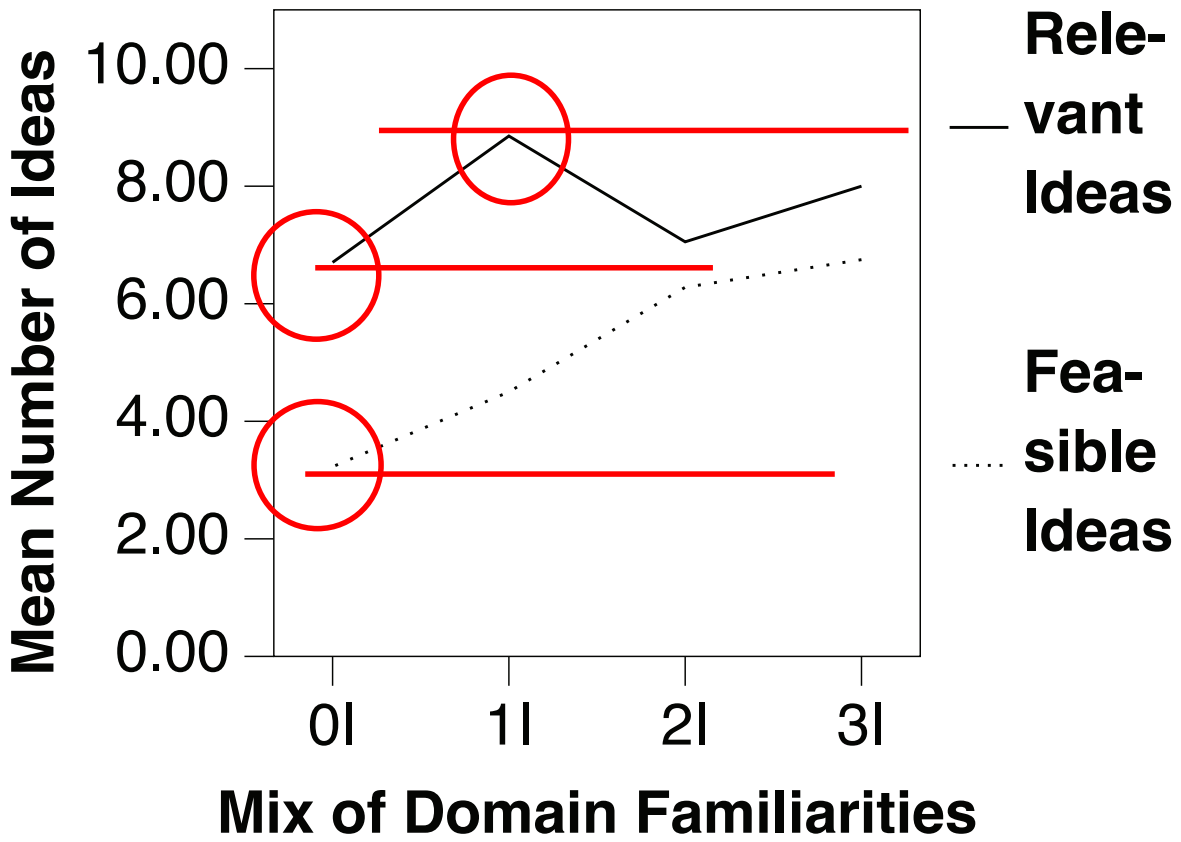
$F$ is $F$-test; $p$ is $p$-value of $F$-test; $f^2$ is Cohen effect size; $P$ is post-hoc power.

# Focused ANOVA Results

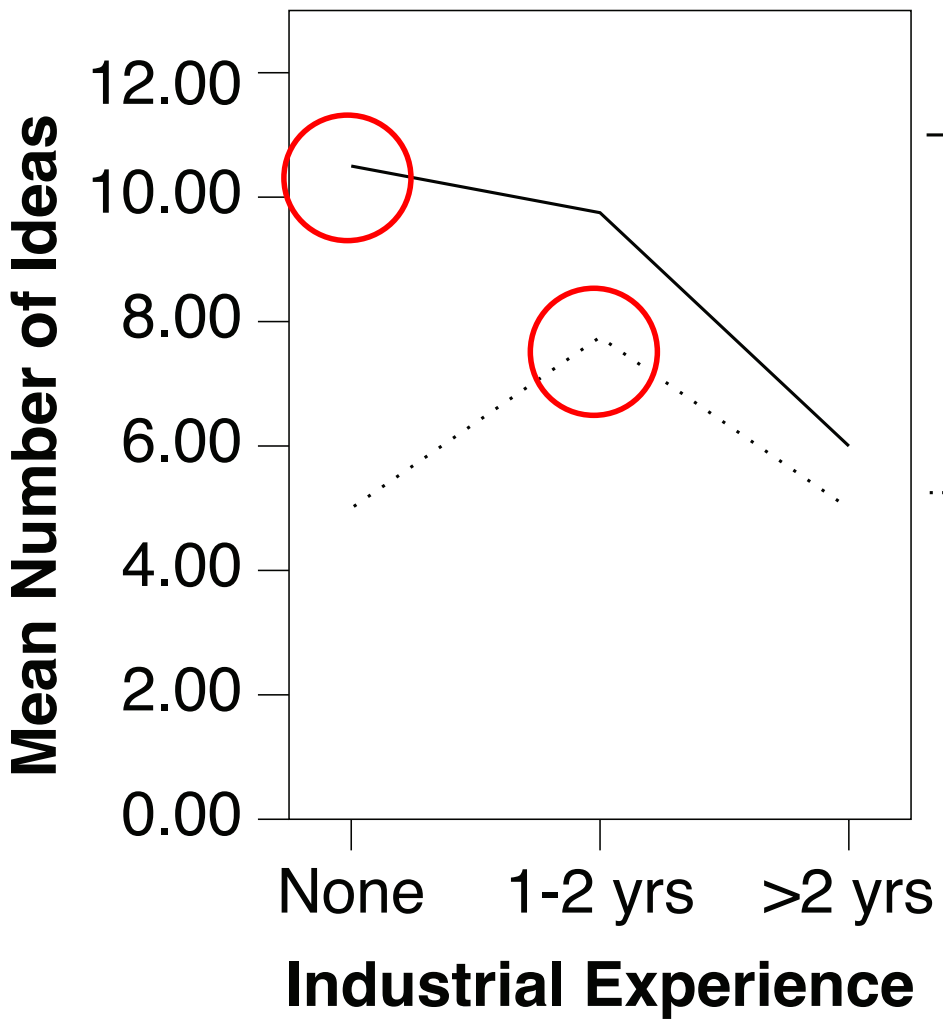
| | Relevant Ideas | | Feasible Ideas | |
|---|---|---|---|---|
| Effect | *p* | *P* | *p* | *P* |
| Mix of Domain Familiarities | **.032** | .816 | **.015** | .941 |
| Industrial Experience | **.027** | .833 | .098 | .499 |

*p* is *p*-value of *F*-test; *P* is post-hoc power.

# ANOVA Results: Impact of Domain Knowledge



# ANOVA Results: Impact of Industrial Experience




Too much industrial experience not helpful! Consistent with findings on ignorance!

# ANOVA Results: Non-Parametric Test on Innovative Ideas

| ***Effect*** | *Kruskal-Wallis* Significance |
|---|---|
| Mix of Domain Familiarities | .966 |
| Creativity | .996 |
| Industrial Experience | .240 |
| RE Experience | .749 |

These are p-values that need to be <.05 for significance.

So NONE are significant.

# Threats to Validity

- *Conclusion Validity:* Low Statistical Power: 20 teams would be enough to achieve statistical power of 0.80, but, the unequal number of teams in the mixes reduces statistical power.
- *Internal Validity:* Voluntary Subjects: All subjects were voluntary but were randomized to the extent possible while still getting the necessary mixes of domain familiarities among the teams.

# Threats to Validity

- *Construct Validity:* Confounding Constructs: Sometimes the value of an independent variable affects the results more than the presence or absence of the variable would.
- *External Validity:* Population Validity: The experiment used student subjects instead of professional analysts, although the students are mostly co-op and work one term per year.

# Conclusion About Hypotheses

- Hypothesis $H_{11}$ is strongly accepted:
  ***The effectiveness of a team in requirements idea generation is affected by the team's mix of domain familiarities.***
- Hypothesis $H_{20}$ is weakly accepted:
  ***The effectiveness of a team in requirements idea generation is not affected by the team's creativity level.***

# Conclusion About Hypotheses

- Hypothesis $H_{30}$ is accepted:
  ***The effectiveness of a team in requirements idea generation is not affected by the team's RE experience.***
- Hypothesis $H_{41}$ is accepted:
  ***The effectiveness of a team in requirements idea generation is affected by the team's industrial experience.***

# Main Result

From these results, considering the threats, the main hypothesis, that

***A team consisting of mix of DIs and DAs is more effective in requirements idea generation than a team consisting of only DAs***,

appears to be weakly supported.

# Expected Application of the Results

Help RE managers in forming teams that are performing knowledge-intensive RE activities, by

- providing a list of RE activities for which domain ignorance is at least helpful and
- providing advice on the best mix of DIs and DAs for any RE activity.

# New Experiments

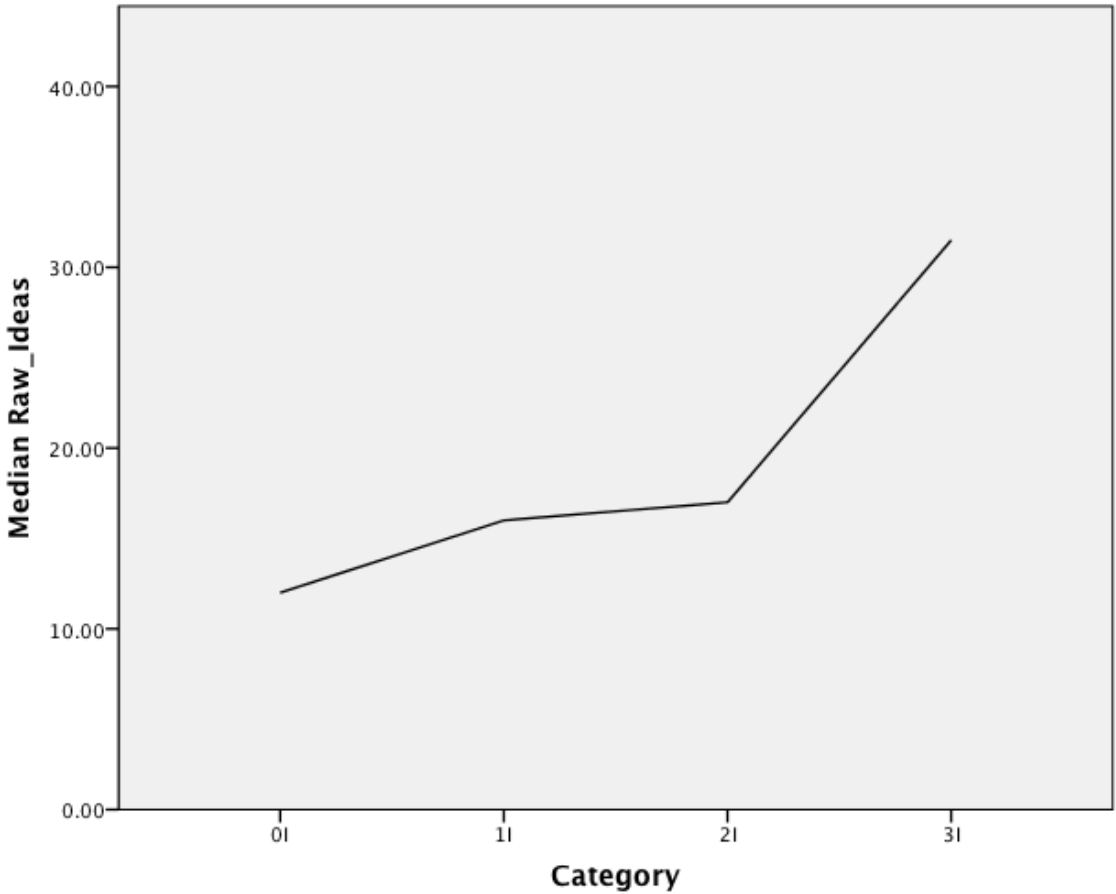
**Niknafs has done more repetitions of the experiment aimed at getting more teams of each mix and balancing the number of teams with the various mixes.**

**He ended up with 10 teams per mix, for a total of 40 teams.**

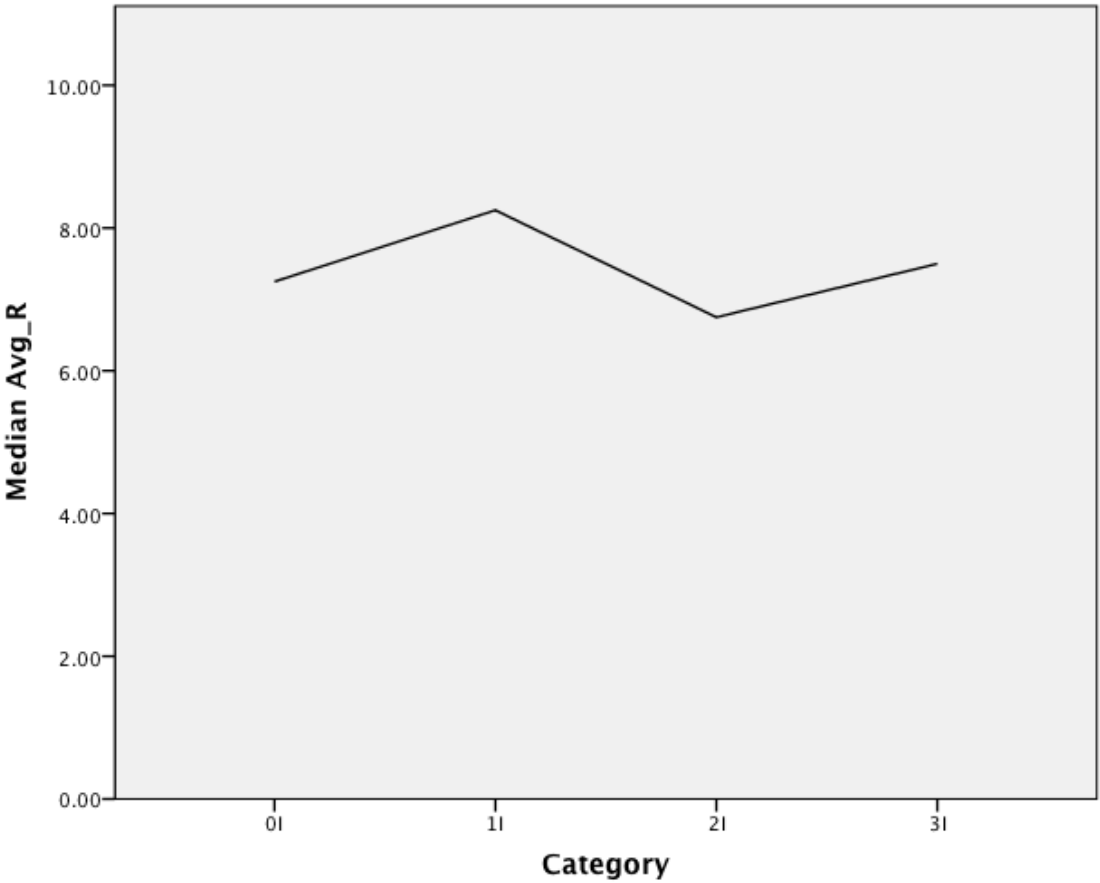
**Here are plots of the median numbers of ideas of all kinds per team against team mixes.**

If time permits for full treatment of new experiments, go to seminar slides page 29.

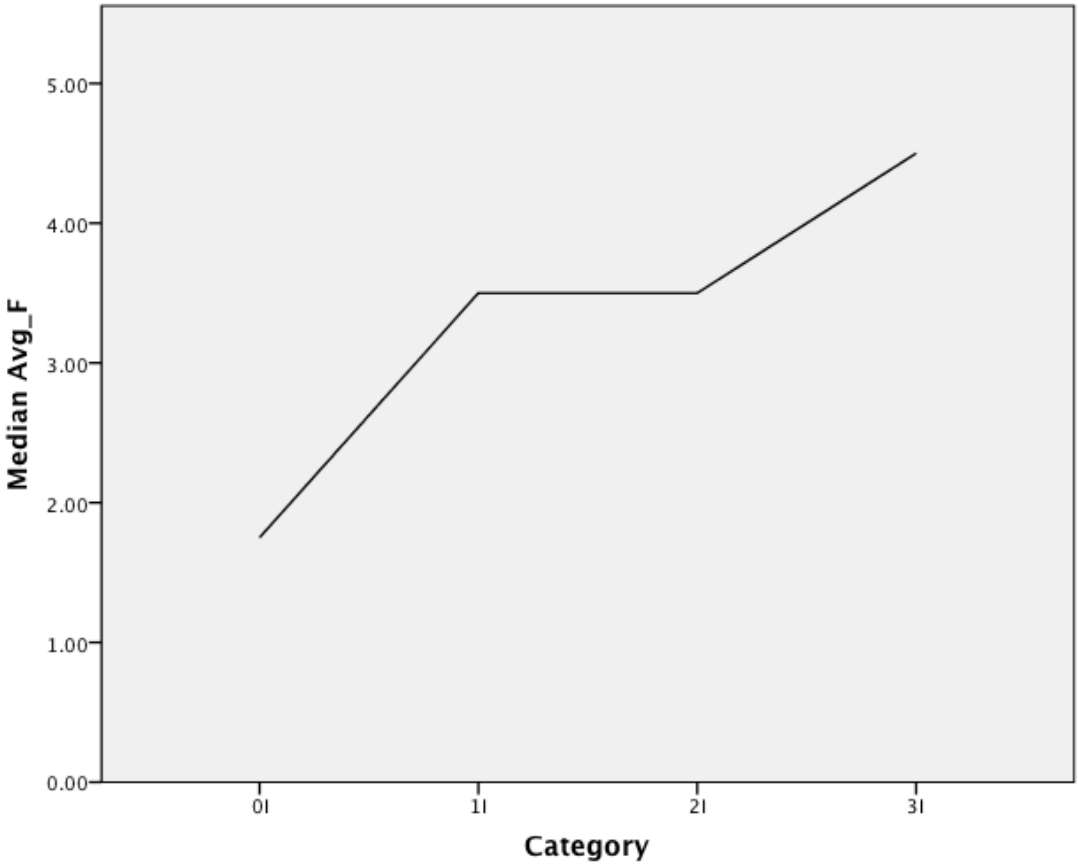
If you have time for case study go to defense slides page 36



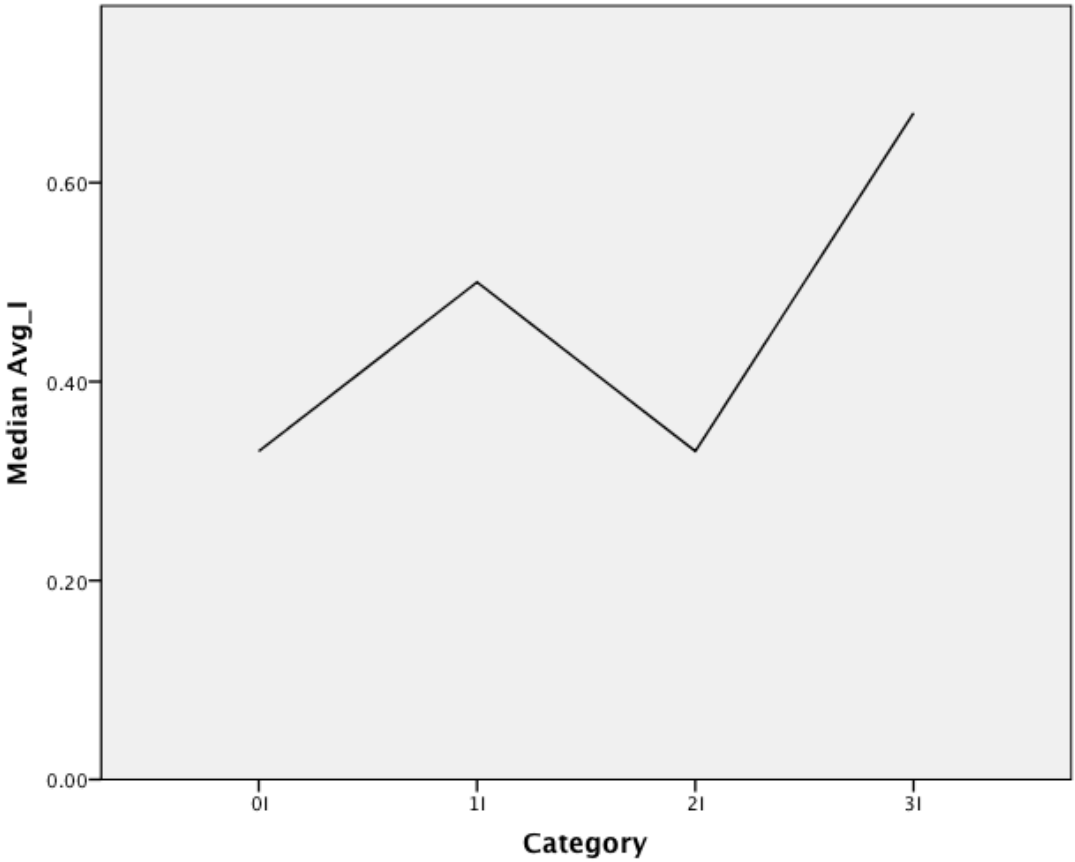

Median Number of Raw Ideas as a function of Mix




Median Number of Relevant Ideas as a function of Mix




Median Number of Feasible Ideas as a function of Mix




Median Number of Innovative Ideas as a function of Mix

# Preliminary Results

**The results so far seem to be:**

**The more ignorants in a team, the more ideas of most kinds the team generates.**

**A team with only ignorants generates more ideas of most kinds than a team with a mix of ignorants and awares, and still more than a team of only awares.**

# Preliminary Results, Cont'd

**The previous results suggested that a team with a mix generated more ideas of most kinds than a team of only ignorants or only awares.**

**That this would be the case was the main hypothesis.**

**The new results are a disappointing surprise.**

# However!!!

**ANOVAs show also the more experience of most kinds a team has the more ideas of most kinds it generates.**

**But, this is what we have been assuming all along: technical competence coupled with domain ignorance, in the smart ignoramus, leads to generation of more and better ideas.**

# Smart Ignoramuses are Needed

**Ignorance alone is not enough.**

**That is, someone who doesn't know the domain, but is highly technically competent applies thinking patterns from other disciplines to a new domain ...**

**and comes up with creative new ideas in that new domain.**

# Conclusion & Future Work

**So maybe the results are good after all!**

**Niknafs is now doing the deeper analysis.**

If you have time for case study go to defense slides page 36.

If there is at least 10 minutes left, do this, if not, skip up to "Mathematicians as Ignoramuses" to finish off the talk!

# Role of Domain Ignorance in Software Development

UNIVERSITY OF WATERLOO

uwaterloo.ca

Gaurav Mehrotra

Master's Thesis Presentation

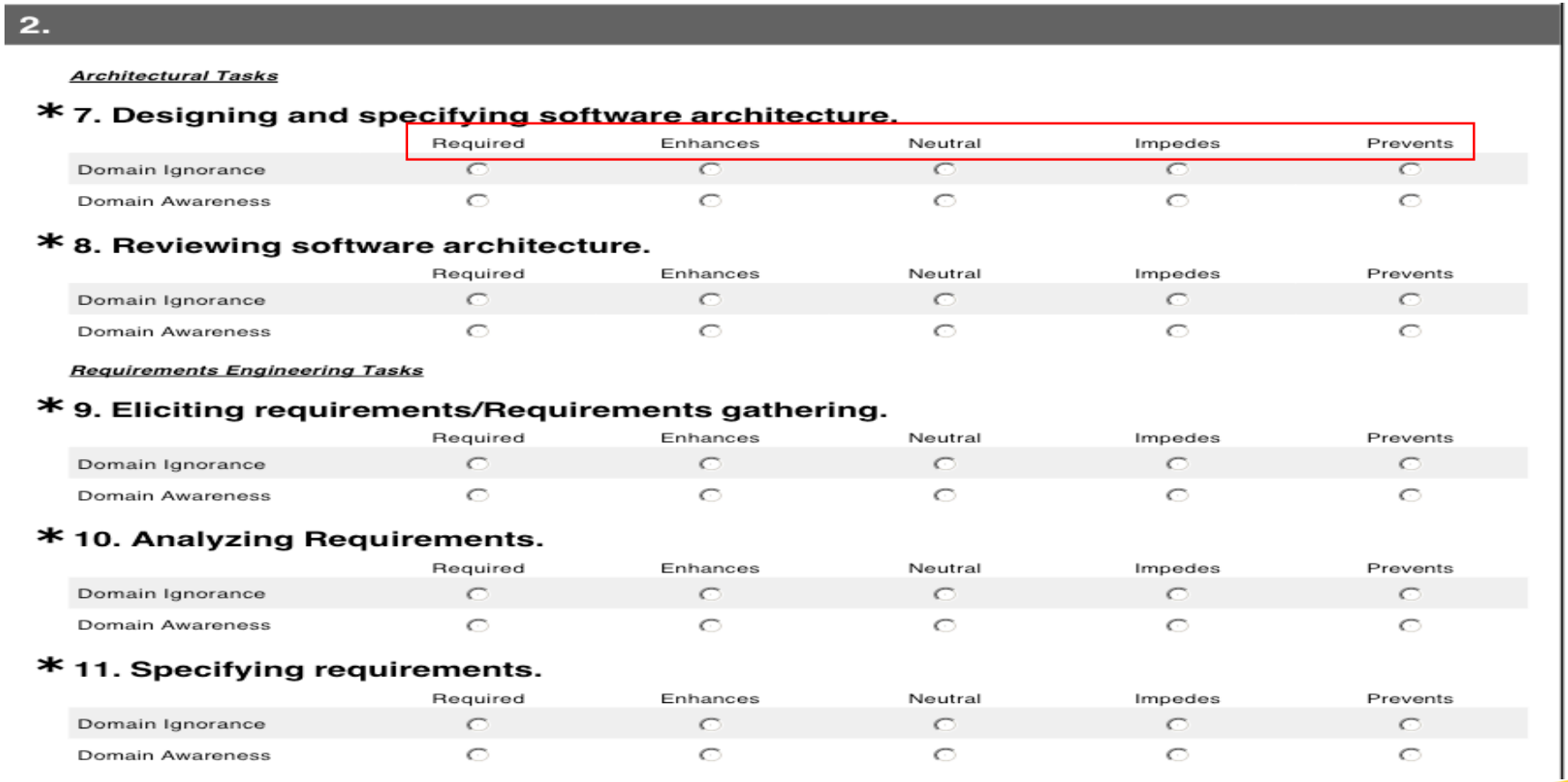
Daniel M. Berry, Supervisor

# Problem Statement

This research aims to answer two important questions:

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

# Empirical Study Design



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 | ○ | ○ | ○ | ○ | ○ |

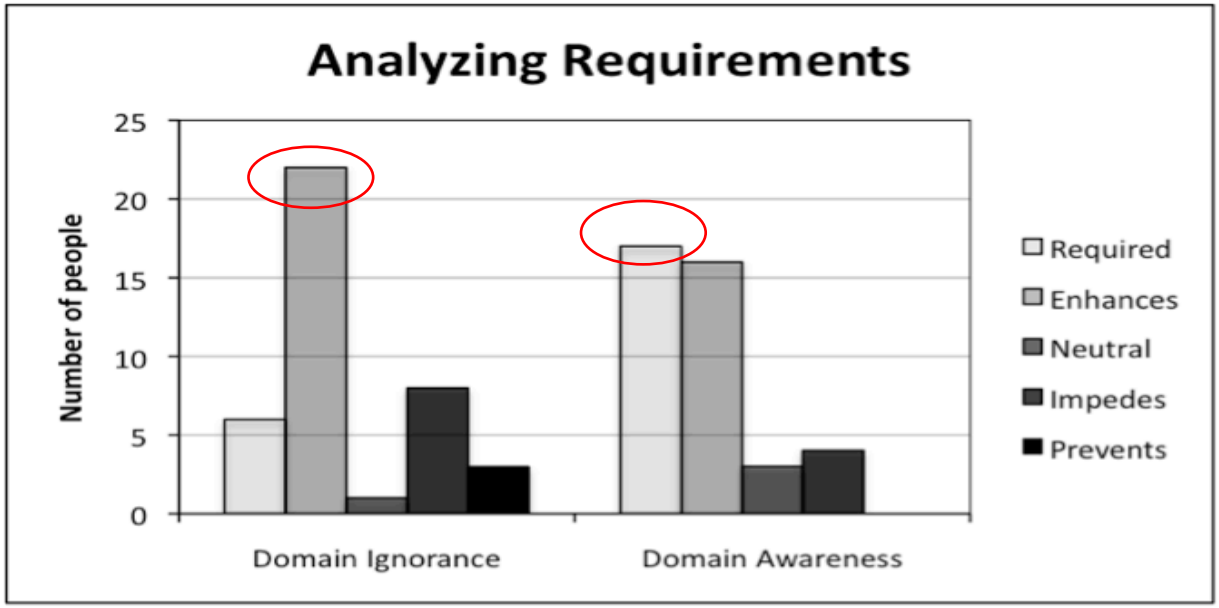
* 10. Analyzing Requirements.

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

* 11. Specifying requirements.

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

# How Results Were Obtained




Mode (Domain Ignorance) = “Enhances”

Mode (Domain Awareness) = “Required”

Zip through the next 8 slides, up to first highlighted slide, which is about the Hypothesis.

# Domain Ignorance Helps:

**(Bold face means that also domain awareness helps the activity.)**

- **requirements gathering**,
- analyzing requirements,
- identifying project risks,
- creating high-level software design,
- **user interface design**,

# Domain Ignorance Helps:

- **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**,

# Domain Ignorance Helps:

**(Italics means that domain ignorance enhances, while domain awareness is required for the activity.)**

- *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.

# Domain Ignorance Hinders:

(Bold face means that Dagenais *et al* report that the activity was done by newbies with smooth immigrations.)

- designing and specifying software architecture,
- **reviewing software architecture**,
- specifying requirements,
- validating requirements,
- reusing and managing requirements,
- inspecting code,

# Domain Ignorance Hinders:

- managing builds of a software,
- deployment planning,
- risk planning/monitoring and control,
- creating low level software design,
- preventing security threats,
- identifying design and implementation rationale,
- **fixing bugs**,

# Domain Ignorance Hinders:

- developing unit test cases
- developing white box test cases,
- developing integration test cases,
- determining source of a bug,
- test planning for a release,
- developing system/performance test cases,
- **manually executing test cases**, and
- providing technical support to users.

# Domain Ignorance Not Affect:

(Bold face means that Dagenais *et al* report that the activity was done by newbies with smooth immigrations.)

- learning processes/practices/technology used,
- **source/version control tasks**,
- coding simple features,
- other code oriented tasks,
- automating test cases,

# Domain Ignorance Not Affect:

- reviewing trace information,
- attending courses/trainings,
- attending formal project meetings,
- attending code/project walkthroughs,
- **compiling project code**, and
- **installing and configuring development environment**.

# Recall Berry's Hypothesis

A newbie's immigration into a project is smoothest when he or she is assigned tasks that are helped by domain ignorance

- He or she is immediately useful while
- learning the domain proceeds more naturally and with less pressure

I decided to test the hypothesis by getting raw data from Dagenais et al and comparing those data with the results I got from the survey.

# Testing Hypothesis

- Dagenais *et al* studied the immigration of newbies into software development projects, with an aim to determine how to make these immigrations smoother.
- Transcripts containing information regarding the tasks a newbie was assigned during his initial days and the difficulties faced by him in doing those tasks were obtained from Ossher (one of the co-authors).

# Distillation of Activities Domain Ignorance Enhances

If the neutral activities are eliminated from the two pairs of lists,

- the positive plus smooth immigration lists are left with a majority of activities that domain ignorance is thought to enhance, and
- the negative plus non-smooth immigration lists are left with only activities that domain ignorance is thought to impede.

# Distillation of Activities Domain Ignorance Enhances

Therefore, 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.

# Why is support only marginal?

It is clear that in real life, there are many factors affecting smoothness of immigration, including personality.

Without doing a controlled experiment (which perhaps is not real life) we cannot isolate any factor.

So, the best we can say is that all other factors being equal, perhaps it's best to assign a newbie to an activity that domain ignorance helps.

After publishing "The Importance of Ignorance in RE",
I got e-mail from Martin Feather, a formal methodologist.

# Mathematicians as Ignoramuses

**Martin Feather of JPL on Importance of Ignorance Paper:**

**I have often wondered about the success stories of applications of formal methods. Should these successes be attributed to the formal methods themselves, or rather to the intelligence and capabilities of the proponents of those methods?**

# Mathematicians -1

**Typically, proponents of any not-yet-popularised approach must be skilled practitioners and evangelists to [bring the approach] to our attention. Formal methods proponents seem to have the additional characteristic of being particularly adept at getting to the heart of any problem, abstracting from extraneous details, carefully organizing their whole approach to problem solving, etc.**

# Mathematicians -2

**Surely, the involvement of such people would be beneficial to almost any project, whether or not they applied “formal methods.” Daniel Berry’s contribution to the February 1995 Controversy Corner, “The Importance of Ignorance in Requirements Engineering,” provides further explanation as to why this might be so.**

# Mathematicians -3

**In that column, Berry expounded upon the beneficial effects of involving a “smart ignoramus” in the process of requirements engineering. Berry argued that the “ignoramus” aspect (ignorance of the problem domain) was advantageous because it tended to lead to the elicitation of tacit assumptions.**

# Mathematicians -4

**He also recommended that “smart” comprise (at least) “information hiding, and strong typing ... attuned to spotting inconsistencies ... a good memory ... a good sense of language...,” so as to be able to effectively conduct the requirements process.**

# Mathematicians -5

**Formal methods people are usually mathematically inclined. They have, presumably, spent a good deal of time studying mathematics. This ensures they meet both of Berry's criteria. Mastery of a non-trivial amount of mathematics ensures their capacity and willingness to deal with abstractions, reason in a rigorous manner, etc., in other words to meet many of the characteristics of Berry's "smartness" criterium.**

# Mathematicians -6

**Further, during the time they spent studying mathematics, they were avoiding learning about non-mathematics problem domains, hence they are likely to also belong in Berry's "ignoramus" category. Thus a background in formal methods serves as a strong filter, letting through only those who would be an asset to requirements engineering.**

# Real Value of FMs

**Perhaps the real value of FMs is that they attract really good people, the formal methodologist, who is good at dealing with abstractions, who is good at modeling, etc., the smart ignoramus, into working on the development of your software.**

**Managers know that the success of a software development project depends more on personnel issues than on technological issues.**

# An Implication

**An attempt to train non-mathematically mature domain experts to apply formal methods in their domain is not likely to succeed.**

**Here, you have “dumb” experts, dumb in the sense of mathematically naive.**

**You need smart ignoramuses.**