



17-803 Empirical Methods

Bogdan Vasilescu, S3D

---

# Formulating Research Questions

# Tuesday Summary: All Methods Are Limited

This course teaches strategies to overcome weaknesses

# What This Class Is and Isn't About

- ▶ This class is not about software engineering.
  - ▶ In fact, this is more a **science class** than a class about any CS discipline.
  - ▶ But because I am most familiar with SE, you will see many readings from SE venues.
    - ▶ No prior knowledge of SE or SE research experience is needed to digest these.

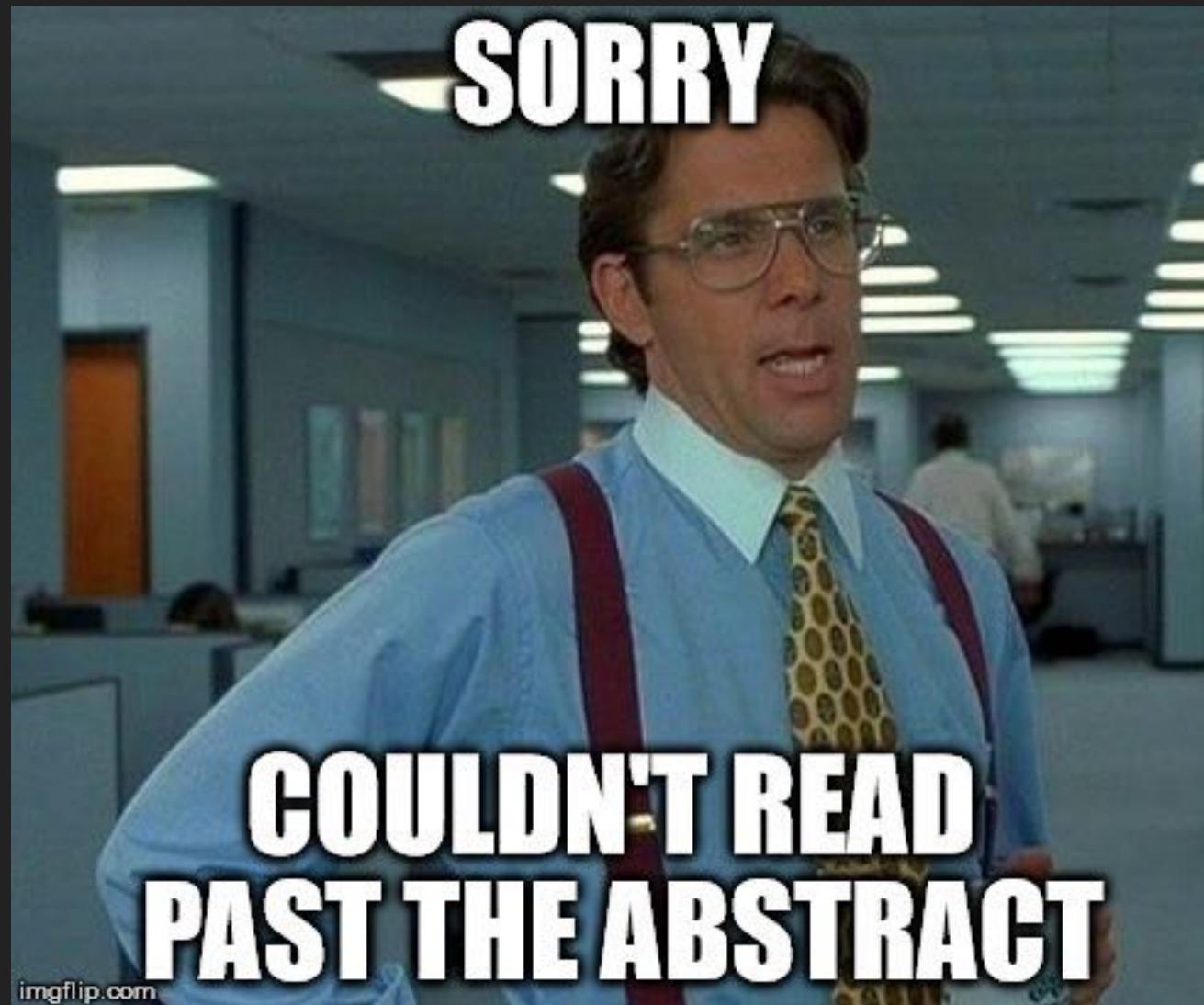
# What This Class Is and Isn't About

## ► This class is ~~secretly~~ about communication.

- You will practice:
  - Articulating what the problem is
  - Articulating why the problem is important
  - Articulating your vision and plan for solving the problem
  - Disseminating (your) results
    - Oral presentations throughout the semester
    - Blog posts?
    - A research project report at the end
- Effective (oral & written) communication is:
  - Necessary for successful research
  - Much harder than you think

# What This Class Is and Isn't About

- ▶ This class is ~~secretly~~ about peer review.
  - ▶ You will read and critique (including in writing) many research papers throughout the semester



# What This Class Is and Isn't About

- ▶ This class is ~~secretly~~ about developing a healthy dose of skepticism.
  - ▶ All fields of science use (the same) empirical methods
  - ▶ By learning about empirical research in CS you'll also get better at recognizing strengths and weaknesses in any scientific paper
    - ▶ Tune your scientific BS meter!

## Anecdotal evidence reliable? One man says “yes”.

A STUDY CONDUCTED YESTERDAY by a man on himself concluded that self-reported anecdotal evidence is, in fact, both reliable and relevant.

The landmark study, conducted by Mark Mattingly of Virginia Beach in his apartment, concluded with 100% accuracy that data collected from personal experience can disprove other data conducted by reputable scientific institutions, thereby proving once and for all that “statistics can’t be trusted”.

In a press release Mr. Mattingly took aim at his detractors saying that “...this study shows what I’ve been telling people on the internet for years: all your fancy evidence and statistics don’t mean nothing in the real world.”

A frequenter of internet forums, comment sections, and social media, Mr. Mattingly recounts that he was inspired to undertake the study when someone reportedly kept insisting that he provide evidence for his claims. “I think everyone’s entitled to an opinion, and that my opinion is worth just as much as anyone else’s” Mr. Mattingly said.

Academic types have criticised the study, and papers who are publishing it, saying that it lacks everything and makes no sense. When shown the study, Emeritus Professor James Albrecht of Carnegie Mellon University looked all confused and hopeless before making pining, guttural sounds.



Mr. Mattingly in his apartment looking all smug.

Mr. Mattingly has responded saying that this is just the first of many studies he intends to conduct, and that a meta-analysis of people who have opinions and anecdotal experiences independent of controls, methodological rigor, blinding and peer review are soon to be published, adding further weight to his initial findings.

Published Saturday 22 February 2014 by [yourlogicalfallacyis.com/anecdotal](http://yourlogicalfallacyis.com/anecdotal)

Photo: Weasello

# Outline for Today

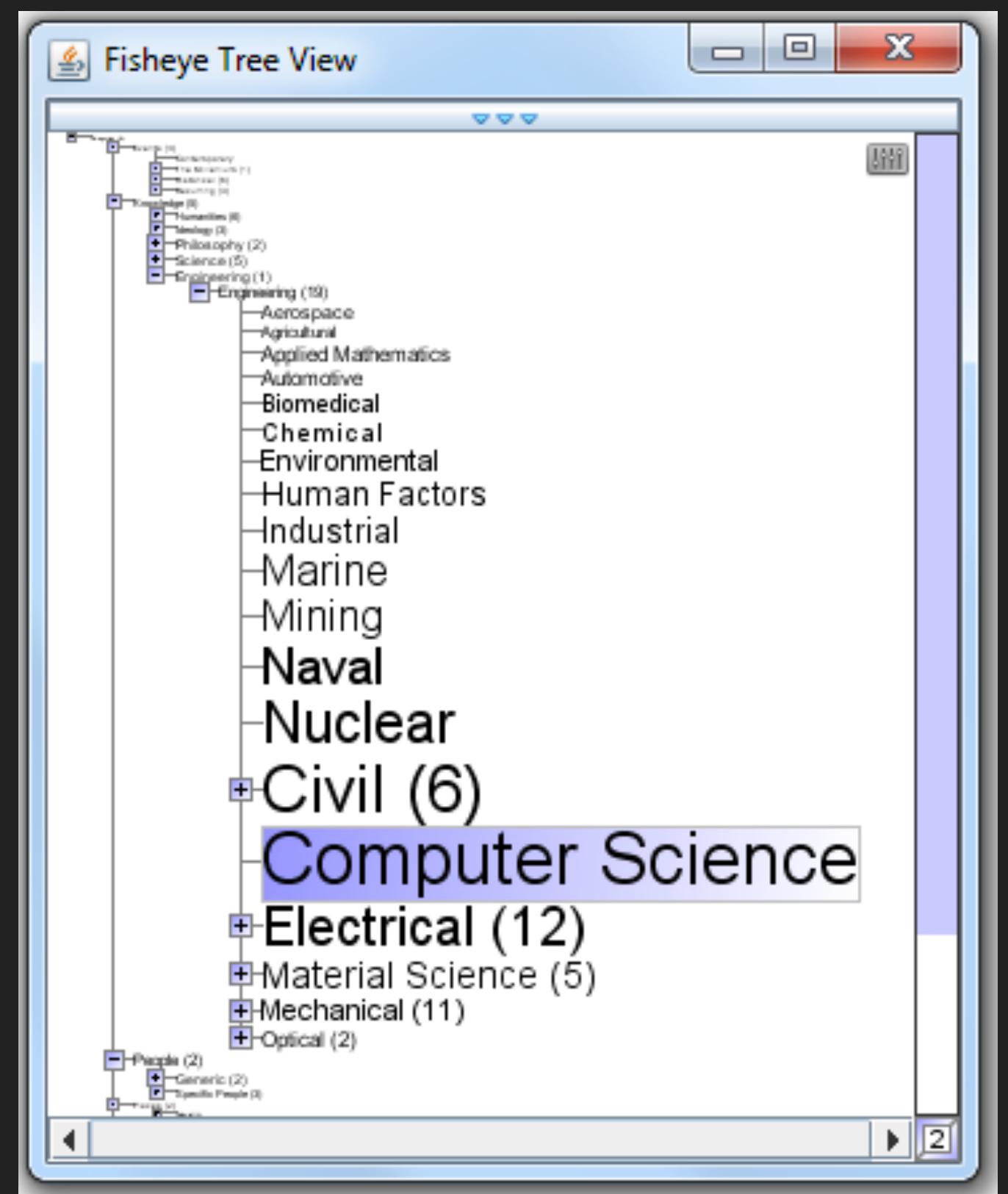
- ▶ Briefly introduce each other
- ▶ Formulating research questions

**Who are you?  
What is your research?  
What would make this course valuable to you?**

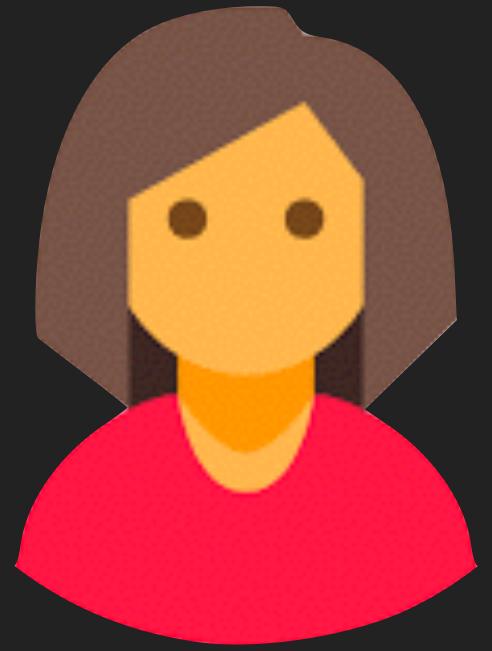
# Formulating Research Questions

# Meet Jane

- ▶ Jane's intuition is that the **fisheye-view file navigator** is more efficient for file navigation than a traditional file navigator.
  - ▶ File navigation requires a lot of scrolling and many clicks to find files.
  - ▶ "Fisheye-views" display information in a compact format that could potentially reduce the amount of scrolling required.
- ▶ Critics argue:
  - ▶ difficult to read
  - ▶ developers won't adopt
- ▶ Jane's research goal: collect evidence that supports or refutes her intuition



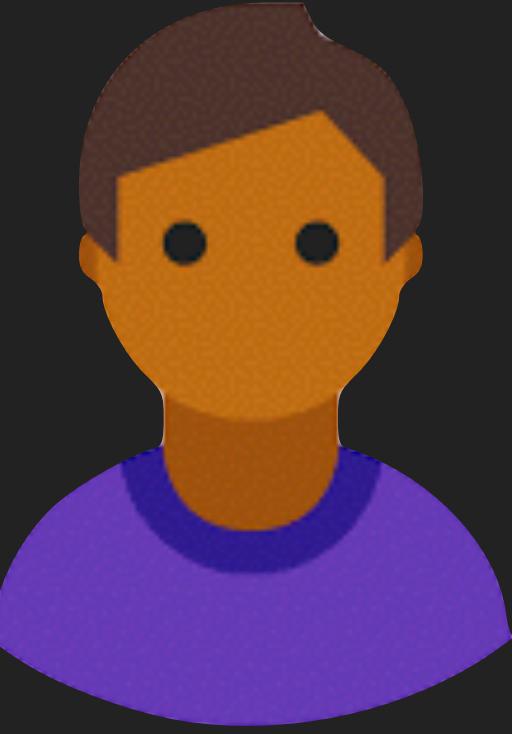
# From Problems To Research Questions



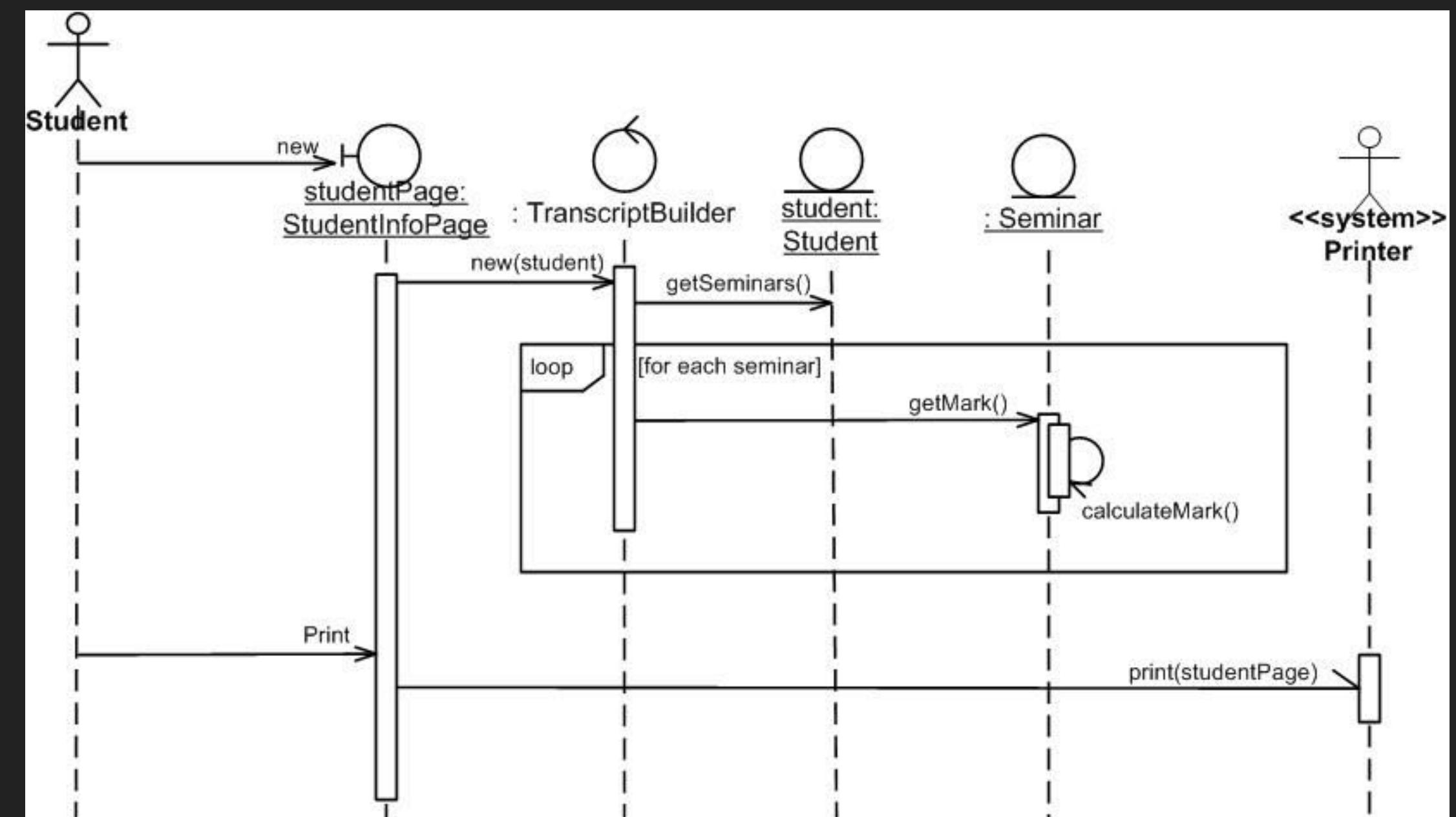
Jane

*Formulate Jane's research  
question(s): ...*

# Meet Joe



- ▶ Joe is interested in how developers in industry use (or not) **UML diagrams** during software design.
  - ▶ His professors recommended UML.
  - ▶ His EvilCorp internship indicates that UML is rarely used.
- ▶ Joe's research goals:
  - ▶ Explore how widely UML diagrams are used in industry.
  - ▶ Explore how these diagrams are used as collaborative shared artifacts during design.



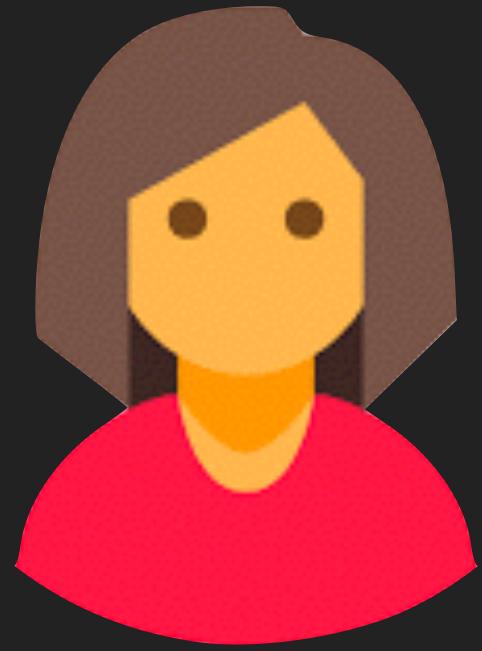
# From Problems To Research Questions



Joe

*Formulate Joe's research  
question(s): ...*

# Critique These Research Questions



Jane

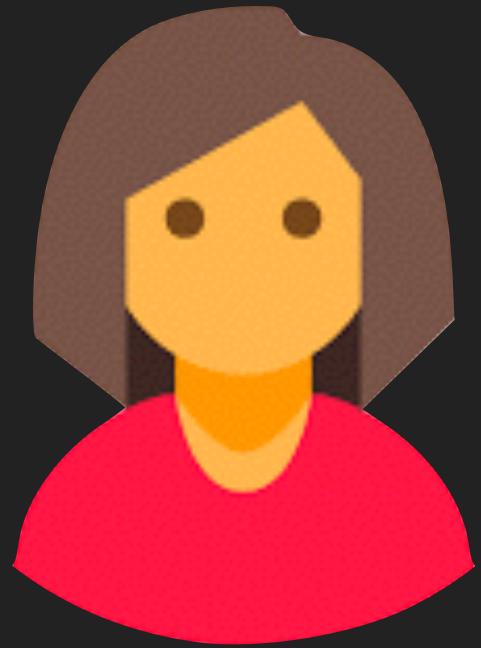
*"Is a fisheye-view file navigator more efficient than the traditional view for file navigation?"*



Joe

*"How widely are UML diagrams used as collaborative shared artifacts during design?"*

# The Most Obvious Question Is Not Always the Best Choice for a Starting Point

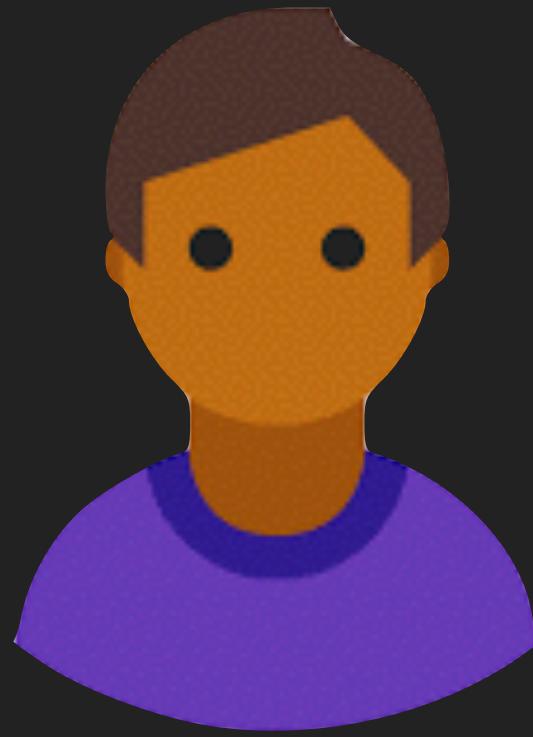


Jane

*"Is a fisheye-view file navigator more efficient than the traditional view for file navigation?"*

- ▶ Do we already know that some people (who?) need to do file navigation?
- ▶ What does file navigation mean exactly?
- ▶ Under what circumstances do these people do file navigation?
- ▶ Is efficiency (measured how?) a relevant goal for these people?

# The Most Obvious Question Is Not Always the Best Choice for a Starting Point



Joe

*"How widely are UML diagrams used as collaborative shared artifacts during design?"*

- ▶ What's a "collaborative shared artifact"?
- ▶ Can we reliably identify one?
- ▶ Can we reliably say which things are and aren't UML diagrams?

**Both questions are vague, because they make assumptions about the phenomena to be studied, and kinds of situation in which these phenomena occur.**

**Some possible (better) questions Jane and Joe could have asked**

# Exploratory Questions

## ► Existence questions

 “Is file navigation something that (certain types of programmers) actually do?”

 “Is efficiency actually a problem in file navigation?”

 “Do collaborative shared artifacts actually exist?”

## ► Description and Classification questions

 “How can we measure efficiency for file navigation?”

 “What are all the types of collaborative shared artifacts?”

## ► Descriptive-Comparative questions

 “How do fisheye views differ from conventional views?”

 “How do UML diagrams differ from other representations of design information?”

## Outcomes:

- Clearer understanding of the phenomena
- More precise definitions of the theoretical terms
- Evidence that we can measure them
- Evidence that the measures are valid

# Base-Rate Questions (Normal Patterns of Occurrence of Phenomena)

- ▶ Frequency and distribution questions

-  "How many distinct UML diagrams are created in software development projects in large software companies?"

- ▶ Descriptive-Process questions

-  "How do programmers navigate files using existing tools?"

## Outcomes:

- ▶ Basis for saying whether a particular situation is typical or unusual

# Relationship Questions

## ► Relationship questions

-  "Does efficiency in file navigation correlate with the programmer's familiarity with the programming environment?"
-  "Do managers' claims about how often they use UML correlate with the actual use of UML?"

## Outcomes:

- Establish that occurrence of one phenomenon is related to occurrence of another

# Causality Questions

## ► Causality questions

 "Do fisheye-views cause an improvement in efficiency for file navigation?"

## ► Causality-Comparative questions

 "Do fisheye-views cause programmers to be more efficient at file navigation than conventional views?"

## ► Causality-Comparative Interaction questions

 "Do fisheye-views cause programmers to be more efficient at file navigation than conventional views when programmers are distracted, but not otherwise?"

## Outcomes:

- Explain why a relationship holds by attempting to identify a cause and effect
- Understand how context affects a cause-effect relationship

# Design Questions

- ▶ Design questions

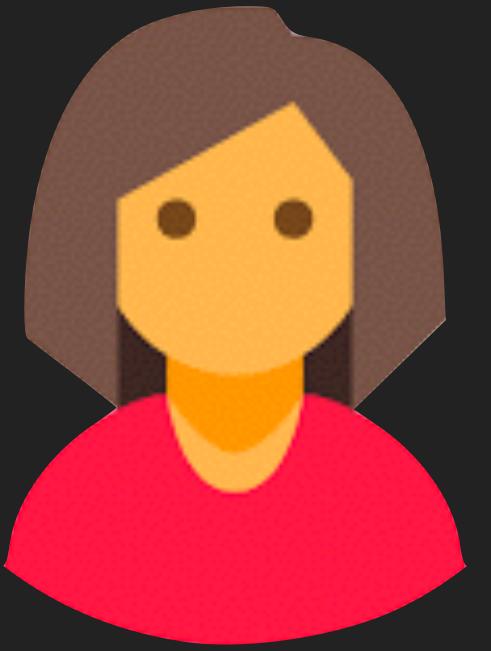
-  "What is an effective way for teams to represent design knowledge to improve coordination?"

## Outcomes:

- ▶ Design better procedures and tools for carrying out some activity
- ▶ Design suitable social or regulatory policies

A long term research program in an applied discipline (e.g., SE) typically involves a mix of both types of questions (knowledge and design).

# What Research Questions Did You Write Down?



Jane

...

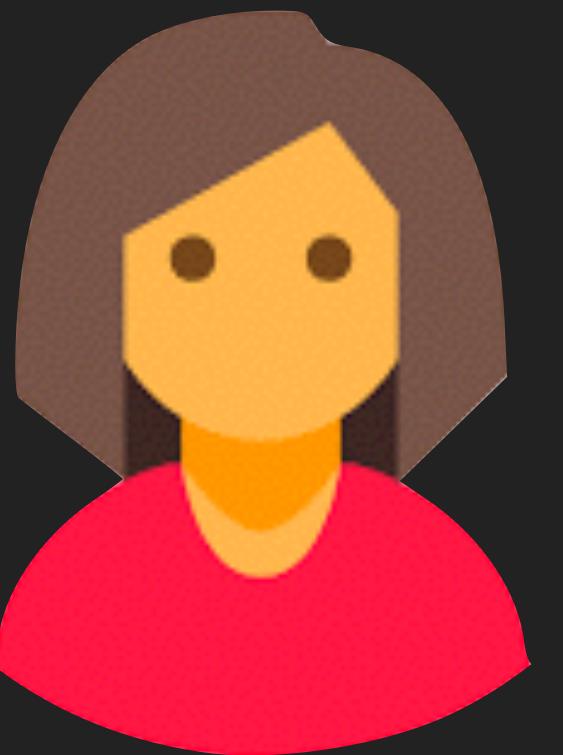


Joe

...

# Remember Last Lecture? (What Will You Accept as Valid Answers?)

Positivist ?



Constructivist ?

# Remember Last Lecture? (What Will You Accept as Valid Answers?)

Positivist



- ▶ Controlled experiments in laboratory conditions are the only source of trustworthy evidence.
  - ▶ to prove that A causes B is to manipulate A in a controlled setting, and measure the effect on B.

Constructivist



- ▶ "Lab experiments are useless, they ignore the messy complexity of real software projects."
  - ▶ Field work instead!
  - ▶ Judgments about "improvements" to file navigation are subjective.
  - ▶ Contextual factors such as distractions have a major impact.

**It is impossible to avoid some commitment to a particular stance, as you cannot conduct research, and certainly cannot judge its results, without some criteria for judging what constitutes valid knowledge.**

# 1-2-all Activity: Choose “Best” Method for Answering These Questions

- ▶ Why do [engineers] ignore [security warnings in their code]?
- ▶ Does [test driven development] improve [code quality]?
- ▶ Which [code review tool] reveals [more bugs]?
- ▶ Do the topics discussed in [online technical forums] deter the involvement of [women]? Has this changed since online learning?
- ▶ How often does [this software] [fail] and in what ways?

Adapted from an activity by Peggy Storey

# Activity: Compare two papers

- ▶ What is the point of the paper?
- ▶ What is the methodology?
  - ▶ Why this choice of method?
- ▶ Do you trust the results?
  - ▶ Why or why not?
  - ▶ What are the risks of being misled?
  - ▶ How do you evaluate a study with this type of methodology?
  - ▶ What does it tell about other ecosystems?

## How to Break an API: Cost Negotiation and Community Values in Three Software Ecosystems

Christopher Bogart,<sup>1</sup> Christian Kästner,<sup>1</sup> James Herbsleb,<sup>1</sup> Ferdian Thung<sup>2</sup>

<sup>1</sup>Carnegie Mellon University, USA <sup>2</sup>Singapore Management University, Singapore

### ABSTRACT

Change introduces conflict into software ecosystems: breaking changes may ripple through the ecosystem and trigger rework for users of a package, but often developers can invest additional effort or a downstream cost to understand how change-related policies are used to substantially change and that different communities illustrate that the cost of change in an ecosystem, it is not always clear what tradeoffs explicit and negotiate changes.

**CCS Concepts:** Semantic versioning in software

**Keywords:** Software versioning, semantic versioning

2014 14th IEEE International Working Conference on Source Code Analysis and Manipulation

## Semantic Versioning versus Breaking Changes: A Study of the Maven Repository

Steven Raemaekers  
*Software Improvement Group*  
Amsterdam, The Netherlands  
Email: s.raemaekers@sig.eu

Arie van Deursen  
*Technical University Delft*  
Delft, The Netherlands  
Email: arie.vandeursen@tudelft.nl

Joost Visser  
*Software Improvement Group*  
Amsterdam, The Netherlands  
Email: j.visser@sig.eu

*Abstract*—For users of software libraries or public programming interfaces (APIs), backward compatibility is a desirable trait. Without compatibility, library users will face increased risk and cost when upgrading their dependencies. In this study, we investigate semantic versioning, a versioning scheme which provides strict rules on major versus minor and patch releases. We analyze seven years of library release history in Maven Central, and contrast version identifiers with actual incompatibilities. We find that around one third of all releases introduce at least one breaking change, and that this figure is the same for minor and major releases, indicating that version numbers do not provide developers with information on the stability of interfaces. Additionally, we find that the adherence to semantic versioning principles has only marginally increased over time. We also investigate the use of deprecation tags and find out that methods get deleted without applying deprecated tags, and methods with deprecated tags are never deleted. We conclude the paper by arguing that the adherence to semantic versioning principles should increase because it provides users of an interface with a way to determine the amount of rework that is expected when upgrading to a new version.

- MAJOR: This number should be incremented when incompatible API changes are made;
- MINOR: This number should be incremented when functionality is added in a backward-compatible manner;
- PATCH: This number should be incremented when backward-compatible bug fixes are made.

These principles were formulated in 2010 by (GitHub founder) Tom Preston-Werner.<sup>2</sup> As argued in the semantic versioning specification, “these rules are based on but not necessarily limited to pre-existing widespread common practices in use in both closed and open-source software.”

But how common are these practices in reality? Are such changes just harmless, or do they actually hurt by causing rework? Do breaking changes mostly occur in major releases, or do they occur in minor releases as well? Do major and minor releases differ in terms of typical size? Furthermore, for the breaking changes that do occur, to

# Credits

- ▶ **Graphics:**
  - ▶ Dave DiCello photography (cover)
- ▶ **Content:**
  - ▶ Easterbrook, S., Singer, J., Storey, M. A., & Damian, D. (2008). Selecting empirical methods for software engineering research. In Guide to advanced empirical software engineering (pp. 285-311). Springer, London.