Photo credit: Dave DiCello

17-803 Empirical Methods

Bogdan Vasilescu, Institute for Software Research

Formulating Research Questions

Thursday, February 4, 2021

Outline for Today

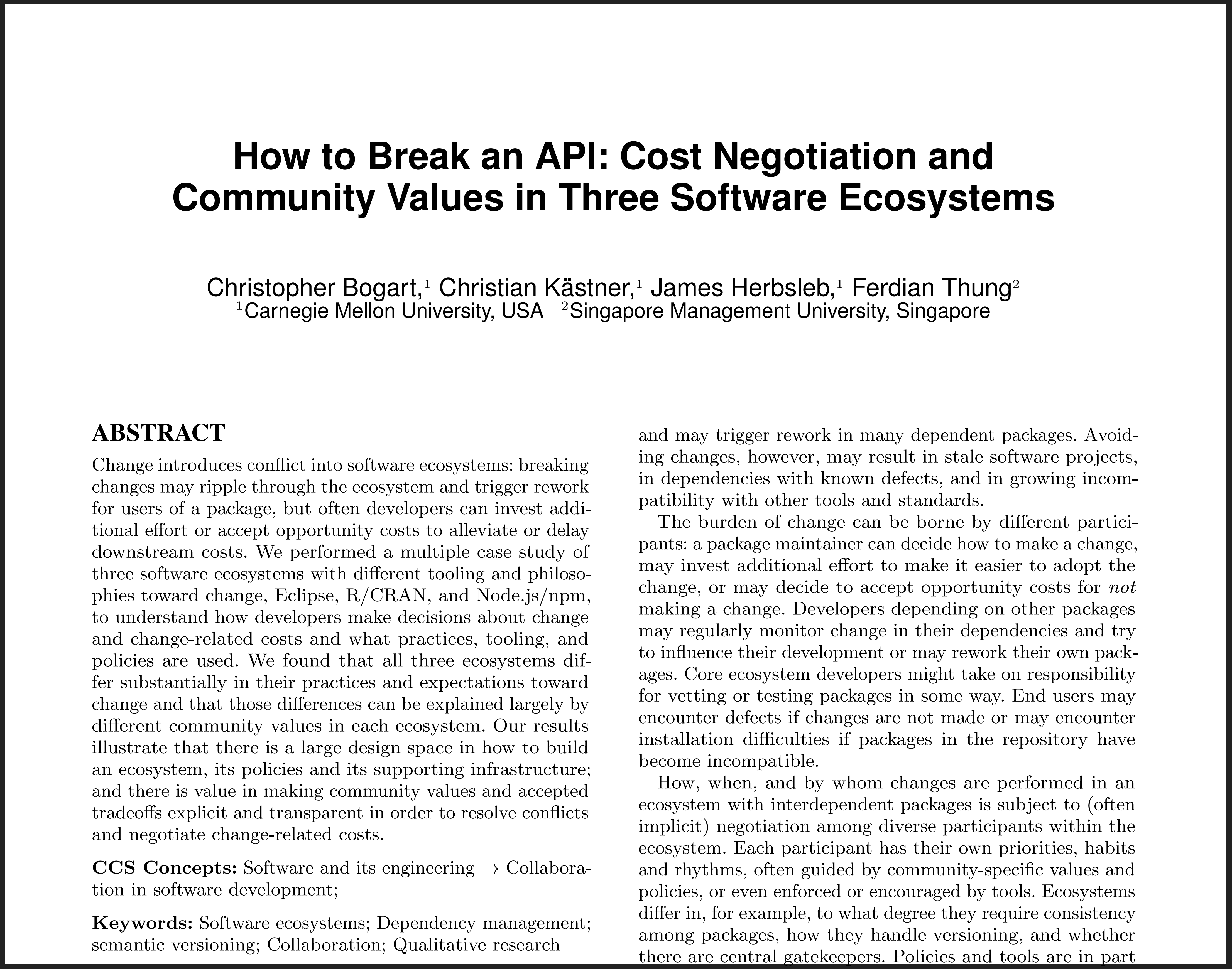
- Get to know each other
- Briefly discuss the two readings
- Formulating research questions
  - Activity

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

Homework Readings Discussion

Bogart, et al “How to Break an API”

- What is the point of this 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,[1] Christian Kästner,[1] James Herbsleb,[1] Ferdian Thung[2]
[1]Carnegie Mellon University, USA [2]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 accept opportunity costs to alleviate or delay downstream costs. We performed a multiple case study of three software ecosystems with different tooling and philosophies toward change, Eclipse, R/CRAN, and Node.js/npm, to understand how developers make decisions about change and change-related costs and what practices, tooling, and policies are used. We found that all three ecosystems differ substantially in their practices and expectations toward change and that those differences can be explained largely by different community values in each ecosystem. Our results illustrate that there is a large design space in how to build an ecosystem, its policies and its supporting infrastructure; and there is value in making community values and accepted tradeoffs explicit and transparent in order to resolve conflicts and negotiate change-related costs.

CCS Concepts: Software and its engineering → Collaboration in software development;

Keywords: Software ecosystems; Dependency management; semantic versioning; Collaboration; Qualitative research

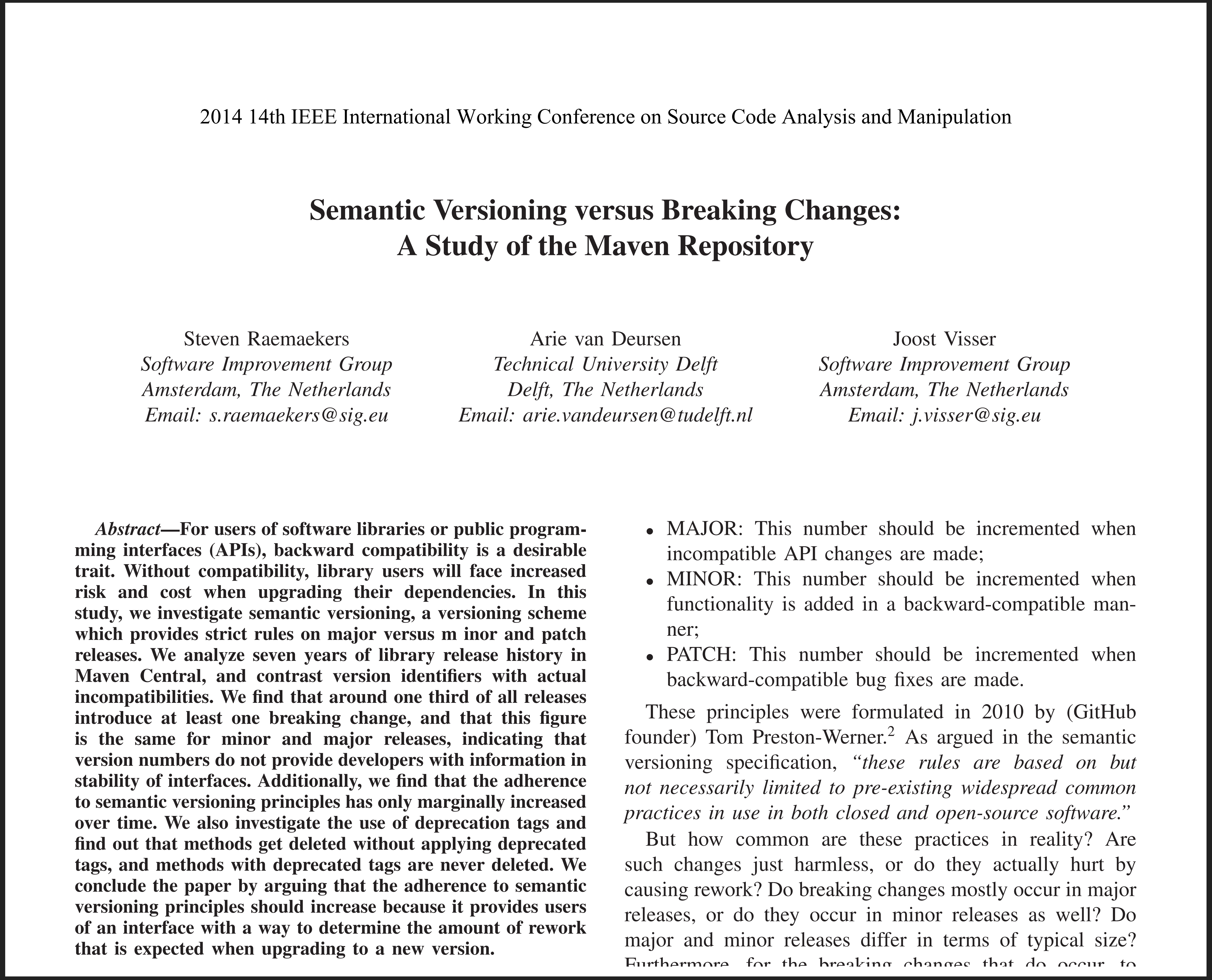
and may trigger rework in many dependent packages. Avoiding changes, however, may result in stale software projects, in dependencies with known defects, and in growing incompatibility with other tools and standards.

The burden of change can be borne by different participants: a package maintainer can decide how to make a change, may invest additional effort to make it easier to adopt the change, or may decide to accept opportunity costs for *not* making a change. Developers depending on other packages may regularly monitor change in their dependencies and try to influence their development or may rework their own packages. Core ecosystem developers might take on responsibility for vetting or testing packages in some way. End users may encounter defects if changes are not made or may encounter installation difficulties if packages in the repository have become incompatible.

How, when, and by whom changes are performed in an ecosystem with interdependent packages is subject to (often implicit) negotiation among diverse participants within the ecosystem. Each participant has their own priorities, habits and rhythms, often guided by community-specific values and policies, or even enforced or encouraged by tools. Ecosystems differ in, for example, to what degree they require consistency among packages, how they handle versioning, and whether there are central gatekeepers. Policies and tools are in part

Raemaekers, et al "Semantic Versioning versus Breaking Changes"

- What is the point of this 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?



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 m inor 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 in 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.

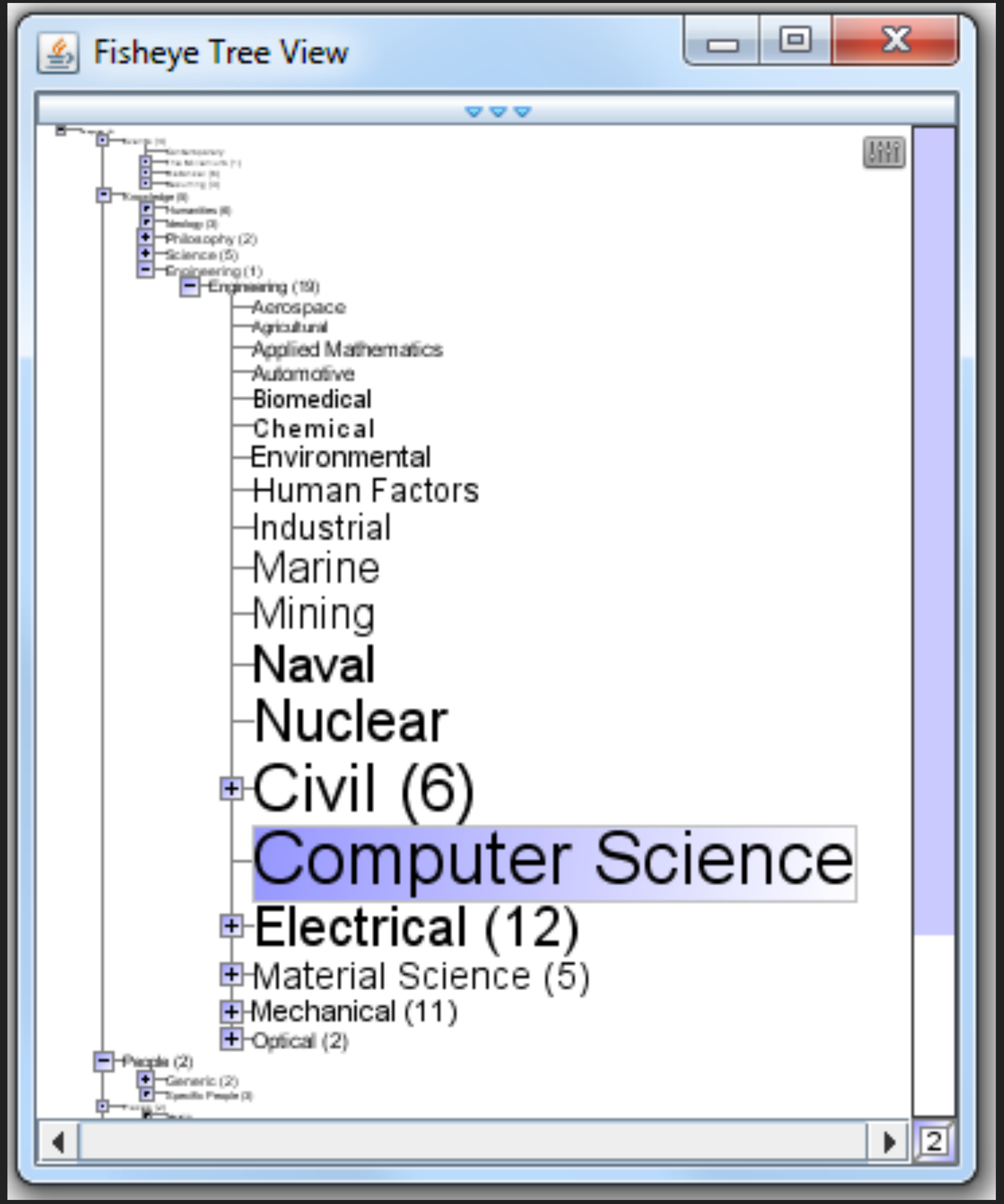
These principles were formulated in 2010 by (GitHub founder) Tom Preston-Werner.[2] As argued in the semantic versioning specification, *"these rules are based on but not necessarily limited to pre-existing widespread common practices in use in both closed and open-source software."*

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

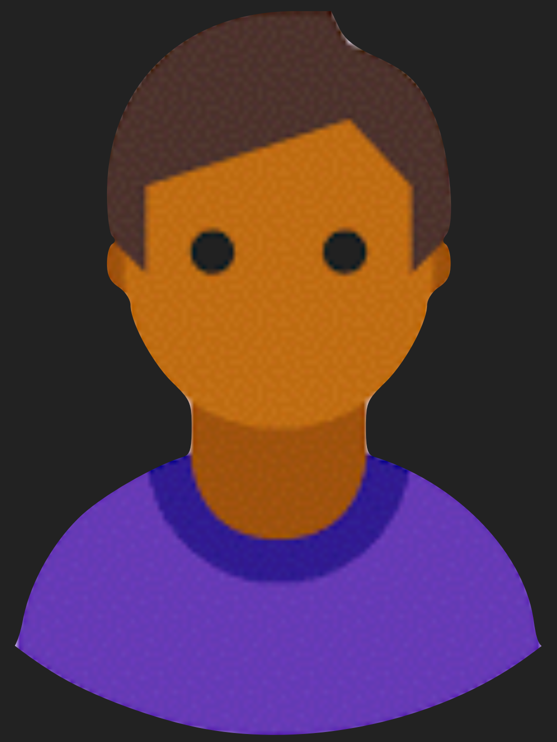
I. Formulating Research Questions

Meet Jane

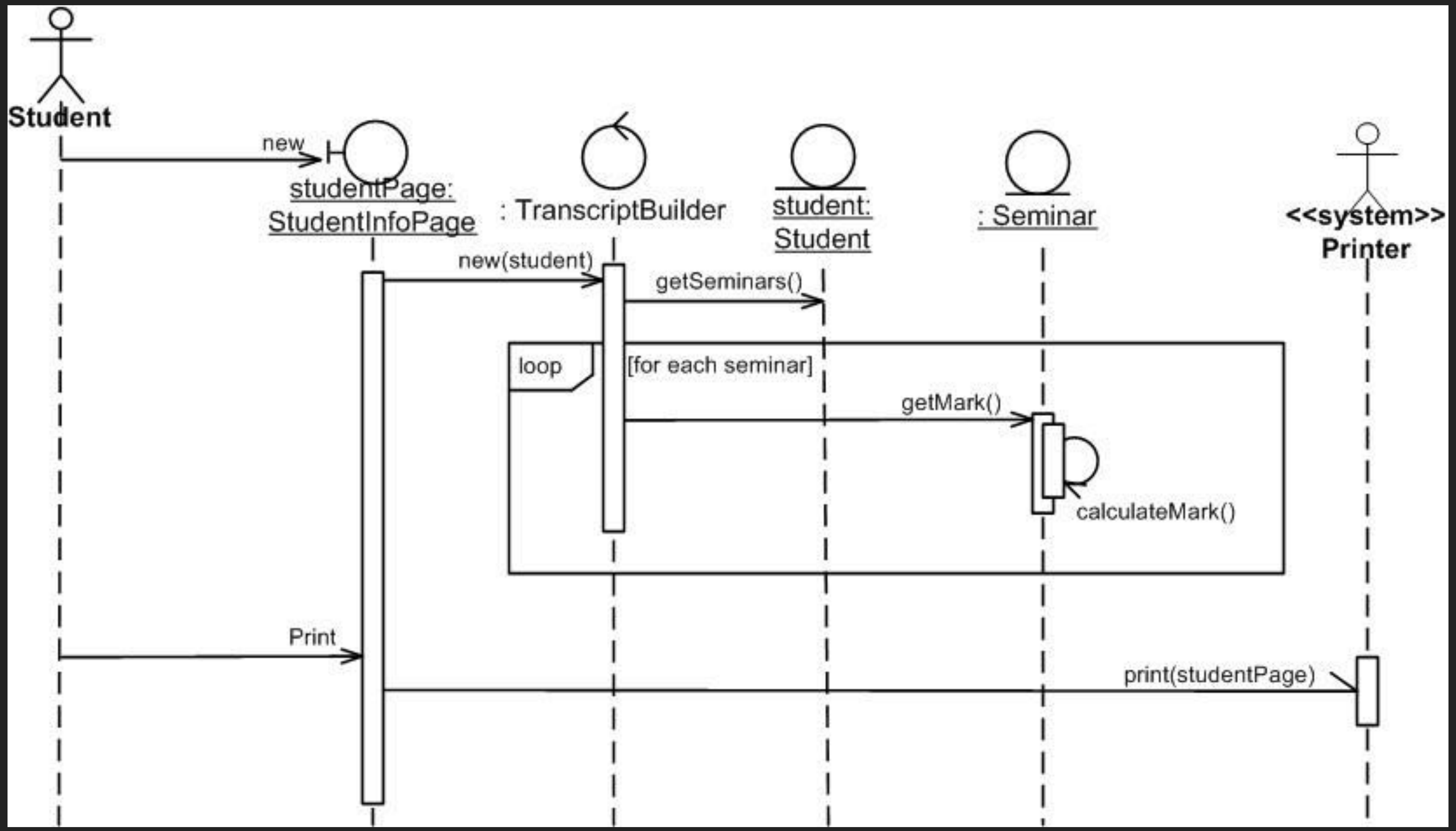
- Jane's intuition is that the fisheye-view file navigator is more efficient for file navigation than a than traditional file navigator.
  - File navigation requires a lot of scrolling and many clicks to find files.
  - "Fisheye-views" display information in a compact format that could potentially reduce 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



Meet Joe



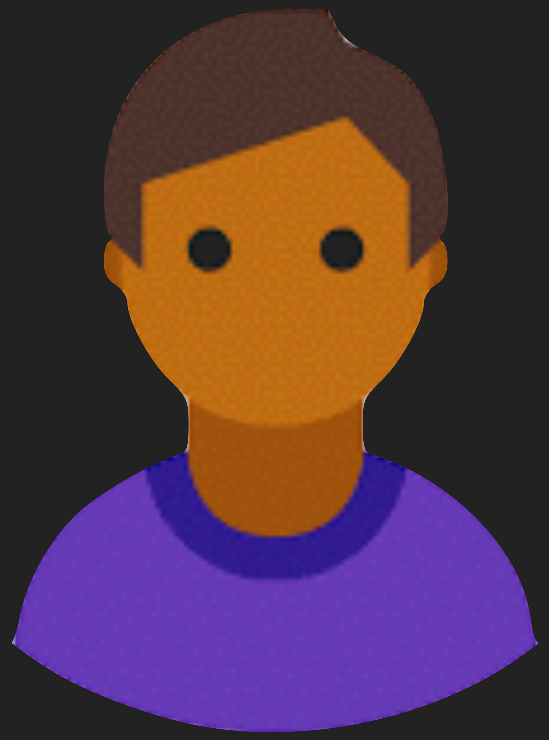
- Joe is interested in how developers in industry use (or not) UML diagrams during software design.
  - His professors recommended UML.
  - His EvilCorp internship indicates that UML is rarely used.
- Joe's research goals:
  - Explore how widely UML diagrams are used in industry.
  - Explore how these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



From Problems To Research Questions

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?"

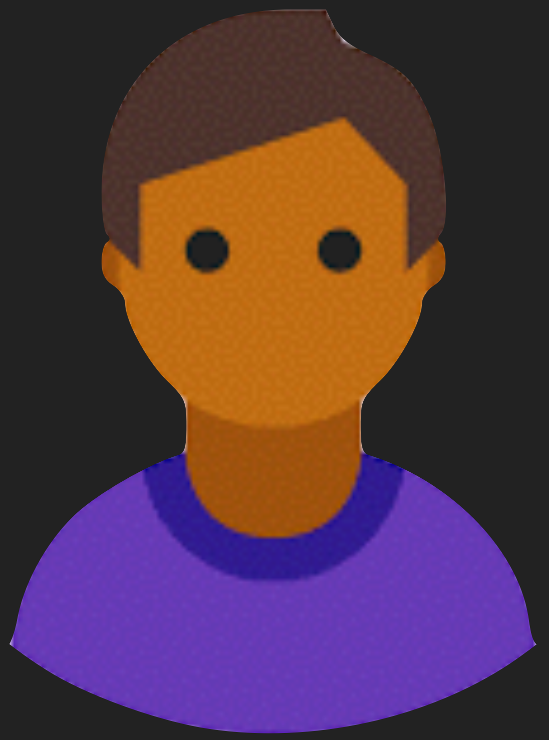
The Most Obvious Question Is Not Always the Best Choice for a Starting Point

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 normal 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).

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

Positivist ?

Constructivist ?

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

Positivist

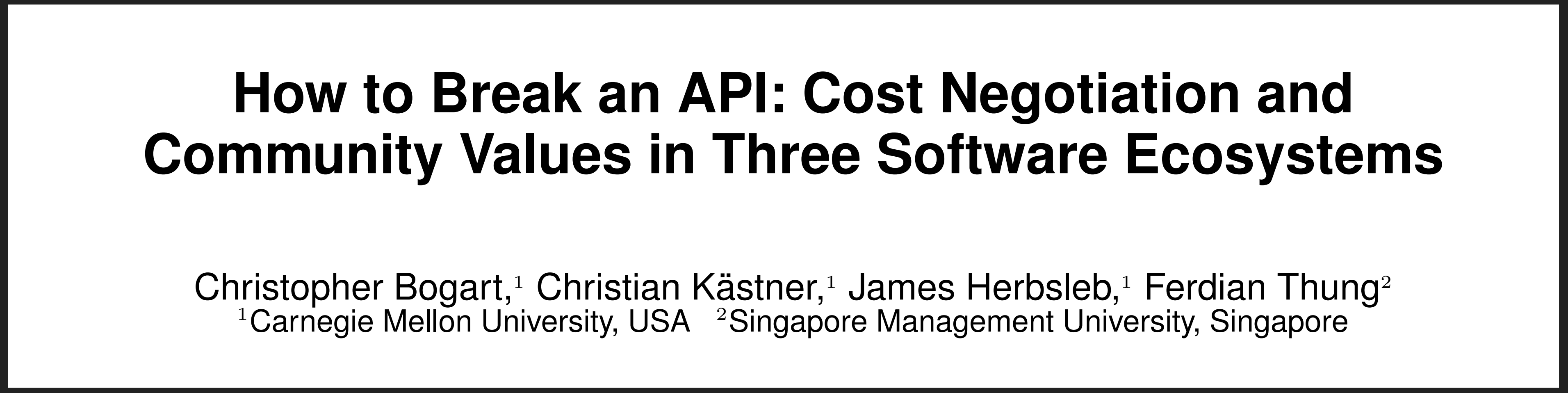
- Controlled experiments in laboratory conditions are the only source of trustworthy evidence.
  - to prove that A causes B is to manipulate A in a controlled setting, and measure the effect on B.

Constructivist

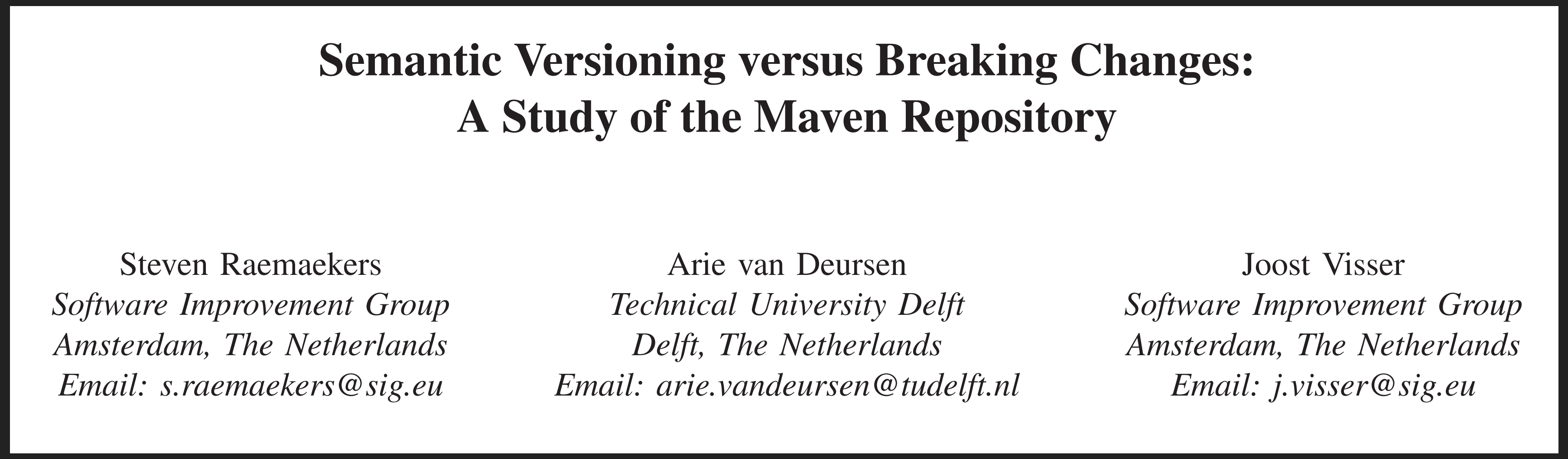
- "Lab experiments are useless, they ignore the messy complexity of real software projects."
  - Field work instead!
- Judgments about "improvements" to file navigation are subjective.
- Contextual factors such as distractions have a major impact.

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.

Discussion: What kind of questions did Bogart and Raemaekers ask?


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

Christopher Bogart,[1] Christian Kästner,[1] James Herbsleb,[1] Ferdian Thung[2]
[1]Carnegie Mellon University, USA [2]Singapore Management University, Singapore


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*

- How do developers make decisions about whether and when to perform breaking changes? How do they mitigate or delay costs for other developers?
- How do developers react to and manage change in their dependencies?
- How do policies, tooling, and community values influence decision making?

- How are semantic versioning principles applied in practice in the Maven repository?
- Has the adherence to semantic versioning principles increased over time?
- How are dependencies to newer versions updated? What are factors causing systems not to include the latest versions of dependencies?
- How are deprecation tags applied to methods in the Maven repository?

Activity: in Breakouts, 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 female students? Has this changed since online learning?
- How often does this software fail and in what ways?

Activity by Peggy Storey

Next Week:
The Role of Theory

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.