What is Software Engineering Research?

Welcome – Summer 2019

Jonathan Aldrich
Professor of Computer Science
Director, SE Ph.D. Program
Summer REUs in SE at CMU

- Work with CMU faculty and researchers
  - Contribute to new SE knowledge

- As part of a community
  - Mailing lists:
    reuse-students@cs.cmu.edu,
    software-group@cs.cmu.edu
  - Slack: reuse-2019
  - SE brown bag lunches
    - Wednesday at noon (usually in Newell Simon 4305)
  - ISR “Birthday Celebration” lunches – June 18 and July 16
  - Other activities

- Weekly seminar series – SSSG
  - Research (and meta-research) talks
  - Reading papers
Software Engineering at Carnegie Mellon

Goal: new knowledge
• about the engineered world (as-is)
• about how to improve that world

Software engineering is the branch of computer science that creates practical, cost-effective solutions to computing and information processing problems, preferentially by applying scientific knowledge, developing software systems in the service of mankind

- from “Software Engineering for the 21st Century: A basis for rethinking the curriculum” by the CMU SE Faculty (Mary Shaw, editor).
One Model of SE Research

Understanding a software engineering problem

- mining software repositories
- grounded theory
- observational study

Designing and prototyping an intervention

- tool
- process
- proof

Evaluating the intervention

- experiment
- case study

One path through the model: research on object protocols
Background: Protocols

- APIs often define **object protocols**
- Protocols restrict possible orderings of method calls
  - Violations result in error or undefined behavior

```java
package java.io;

class FileReader {
    int read() { … }
    …

    /** Closes the stream and releases any system resources associated with it. Once the stream has been closed, further read(), ready(), mark(), reset(), or skip() invocations will throw an IOException. Closing a previously closed stream has no effect. **/  
    void close() { … }
}
```

- We have developed a language, **PLaid**, that builds protocols into the type system and object model
  - now Obsidian does too!
Study 1: How Common Are Protocols?

• Approach: Quantitative Code Corpus Study
  • Used the Java standard library, plus many apps, frameworks
  • Tool identifies code pattern likely to indicate protocol
  • Weed out false positives via manual comparison to definition of what a protocol is
  • Categorized the protocols found

• Results
  • At least 7.2% of types define protocols
    • Compare: 2.5% of types define generics
  • At least 13.3% of classes use protocols
  • Identified 7 categories covering 98% of protocols

Study 2: Protocol Programming Barriers

- **Question:** What barriers do programmers face when using APIs with protocols?
- **Approach:** Observational lab study of professional programmers
  - Programmers did mined protocol tasks (from another study) while thinking aloud
  - We assigned programmer time to quotes they uttered
  - Using open coding, we categorized the quotes
- **Results:** Programmers spent 70-80% of their time asking 4 kinds of questions:
  
  A) What abstract state is the object in?
  B) What are the capabilities of object in state X?
  C) In what state(s) can I do operation Z?
  D) How do I transition from state X to state Y?

**Observations:**
- Primarily a qualitative study
- However, we did gather some quantitative data
Intervention: The **PLaidD** Language

```plaintext
state File {
    val String filename;
}
state ClosedFile = File with {
    method void open() [ClosedFile>>OpenFile];
}
state OpenFile = File with {
    private val CFile fileResource;

    method int read();
    method void close() [OpenFile>>ClosedFile];
}
```
Implementing Typestate Changes

```java
method void open() [ClosedFile>>OpenFile] {
    this <- OpenFile {
        fileResource = fopen(filename);
    }
}
```

Typestate change primitive – like Smalltalk `become`

Values must be specified for each new field

Side note: we’re now building on these ideas in the Obsidian language
Why Typestate in the Language?

- The world has state – so should programming languages
  - egg -> caterpillar -> butterfly; sleep -> work -> eat -> play; hungry <-> full
- Language influences thought [Sapir ‘29, Whorf ‘56, Boroditsky ‘09]
  - Language support encourages engineers to think about states
    - Better designs, better documentation, more effective reuse
- Improved library specification and verification
  - Typestates define when you can call read()
  - Make constraints that are only implicit today, explicit
- Expressive modeling
  - If a field is not needed, it does not exist
  - Methods can be overridden for each state
- Simpler reasoning
  - Without state: fileResource non-null if File is open, null if closed
  - With state: fileResource always non-null
    - But only exists in the FileOpen state
Theory: Plaid’s Type System is Safe

• Typestate checks should ensure protocol objects are accessed safely
• Formal model of language, type system
• **Theorem:** a well-typed program won’t call a method from the wrong state

\[
\begin{align*}
\text{(STnew)} & \quad \frac{\text{fields}(C) = \overline{T}, f}{\Delta \vdash x : T \rightarrow \Delta'} \quad \Delta \vdash \text{new } C(\overline{x}) : \text{full(Object)} \quad C \rightarrow \Delta' \\
\text{(STupdate)} & \quad \frac{k \in \{\text{full, shared}\}, \text{fields}(C) = \overline{T}, f, C <: D}{\Delta \vdash x_2 : T \rightarrow \Delta', x_1 : k(D) \quad E} \quad \Delta \vdash x_1 \leftarrow C(\overline{x_2}) : \text{Void} \rightarrow \Delta \downarrow', x_1 : k(D) \quad C
\end{align*}
\]

Two typing rules in a formal model of Plaid


This paper also explored *gradual typestate*, leading to our gradual verification project!
Implementation: Plaid can be Efficient

- Typestate requires changing behavior at run time
  - How can we make this object model efficient?
- New compilation approach
  - Associate state-based metadata with each object
  - Update methods following metadata on state change
- Prototyped in JavaScript
  - Performance comparison to native JS and to naïve Plaid compiler

Study 3: Effect of Protocol Documentation

• We wanted to know if Plaid can help programmers program more effectively with protocols
  • But that’s a hard question to measure directly, due to learning effects, tool quality, etc.

• Proxy: Plaid’s design enables new forms of javadoc-like documentation. Does the documentation help?
  • plaiddoc: shows state space, organizes methods by state, shows state pre- and post-conditions

• Approach: controlled laboratory experiment
  • 20-participant between-subjects study
  • Task: answer questions identified in study 2b, above

Study 3: Effect of Protocol Documentation

• We wanted to know if Plaid can help programmers program more effectively with protocols
  • But that’s a hard question to measure directly, due to learning effects, tool quality, etc.

• Proxy: Plaid’s design enables new forms of javadoc-like documentation. Does the documentation help?
  • plaiddoc: shows state space, organizes methods by state, shows state pre- and post-conditions

• Approach: controlled laboratory experiment
  • 20-participant between-subjects study
  • Task: answer questions identified in study 2b, above

• Results:
  • plaiddoc participants were 2.2x faster (p < 0.001)
  • javadoc participants were 7.9x more likely to make errors (p=0.002)
One Model of SE Research

Understanding a software engineering problem
- mining software repositories
- grounded theory
- observational study

Designing and prototyping an intervention
- tool
- process

Evaluating the intervention
- proof
- experiment
- case study

Study 1: how common are object protocols in Java?

Study 2: what barriers do programmers face when using object protocols?

Design a programming language based on typestate. Implement compiler and design documentation tool for that language.

Formal model of language and proof of type soundness

Study 3: Can the documentation tool help programmers overcome the barriers from Study 2?

We’re doing this now in Obsidian!
If you like the REU, what might be next?
CMU SE Ph.D. Alumni Careers (examples)

**Thomas LaToza**
Assistant Professor, George Mason
SE/HCI research on how humans interact with code and designing new ways to build software

**Chris Scaffidi**
Associate Professor, Oregon State
Research on helping end-users create software; directs OSU’s master’s in SE

**Josh Sunshine**
Systems Scientist, Carnegie Mellon
SE/PL research, especially on the usability of reusable software components

**Ciera Jaspan**
Tech Lead Manager, Google Engineering Productivity Research
Research on developer productivity at Google; regularly publishes at ICSE, OOPSLA

**Jason Tsay**
IBM Research AI
Research in AI Engineering: how to improve experiences of data scientists, developers who work in AI

**Owen Cheng**
Senior Software Engineer, Uber Advanced Technology Group
SE Research – Careers and Ph.D.

- University professor
  - Pursue your own research agenda – be your own boss!
  - Teach and mentor students in research
  - Even service tasks are rewarding – running programs, organizing conferences

- Industry researcher
  - Explore the most cutting edge ideas in a real environment
  - See those ideas have an immediate impact on products

- For either, you need a Ph.D. in CS…or SE!
  - Primary focus is research
  - Typically takes 5-6 years
    - Master’s degree is not a prerequisite
  - Tuition and stipend are provided
  - Just as fun as an REU in SE!

- CMU among the best places to study CS/SE
Ph.D. Applications

• Primary criterion: promise to do successful independent research
  • You will get practice this summer!

• Apply in December 2019 for Fall 2020 (etc.)

• Documentation: Recommendations, research & industry experience, statement, test scores, grades
Our SE PhD Curriculum

- Research (always ≥ 50% time)
  - Project work
  - Thesis

- Coursework
  - Core SE course
  - Core areas
    - Design/Engineering
    - Symbolic modeling and analysis
    - Behavioral science
    - Society/Bus./Policy
  - 2 Electives

- Practicum
  - Reflection on practice

- Skills
  - Speaking and writing

- Teaching
  - TA two courses

- Community engagement
  - Weekly research seminar
  - Volunteer service

- Experience
  - Prior industrial experience
  - Internships during the program
The Ph.D. is a New World

• Research is your #1 job!
  • Starts immediately when you arrive

• Course grades (mostly) don’t matter: learning does

• Nature of the work differs
  • You will be given ill-defined problems, and have to define them
  • Critical thinking and interpretation dominate fact-finding
  • Much of the feedback you get will be informal

• Challenging and fun
  • A chance to make a lasting contribution to scientific knowledge
  • One of the best periods in my life—also true for many students here!