API Usability and Security FY 15 LENS

Sam Weber

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0002775

Schedule

Team members

Motivation

Project Progress

- Semi-structured interviews
- Prototypes
- User validation studies

Summary and Future Work

Project Funding and Team Members

Project co-funded by SEI and the National Science Foundation

- SEI: One-year funding, FY 15
- NSF: Three-year funding, FY 15-17. Award 1423054

Team Members

Dr. Sam Weber
Dr. Forrest Shull
Robert Seacord
David Keaton
Dr. Brad Myers
Dr. Jonathan Aldrich
Dr. Joshua Sunshine
Michael Coblenz

Summer Interns: Sophie Gairo, Paul Peng

Project Aims and Vision

Goal: To develop and empirically test *concrete* and *actionable* API design principles that lead to more secure code

Long-term vision: empirically test secure development practices

- Many decades of work on secure development practices, most of it based upon experience and reflections by smart people, but little, if any, information about validity and relative merit of different practices

"While design vulnerabilities are common they are not often tracked….With a lack of empirical results on security-related design flaws, research that provides security at the design level may not have any empirical support to validate against."

(Dr. Andrew Meneely)

Principle: Programmers and designers are people too

- Need to design APIs that foster more secure code

Why APIs?

APIs have a large impact upon system security

- C string library still major cause of problems
- See Wang et al, “Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization”

APIs are long-lasting

APIs are generally designed by a small number of more-experienced people

Core Idea: APIs represent boundaries between components. They define what entities exist, what operations they can do and can be done to them, and the responsibilities of the API users and implementers. Most importantly, they rely on programmers understanding each other’s responsibilities and possible actions.

Project Decisions and Methodology

In order to be *concrete*, we need to focus on a particular subset of API design decisions

First focus: **state management**

- How are system state changes controlled?
- Ex: immutable data structures, whose value is fixed when created

Methodology: iterative participatory design

- elicit designer's pain-points and ways of thinking, make trial attempts to address them, evaluate, then iterate

Steps:

1. Semi-structured interviews with experienced developers
2. Create prototypes addressing issues
3. Evaluate with user studies

Interview Results Summary

- Controlling where/when state is changed and by whom is a serious problem. **All** interviewees agreed!
  - Example: mistrust created between groups caused by third-party changing data-structure internals
- Programmers do use concepts like immutability
- Language features, like `const` don't satisfy programmer's needs:
  - Are all objects of a certain class immutable or just some?
  - What about objects which are logically immutable, but whose bits change (caches, self-optimizing data structures…)?
  - Viral nature of C++'s `const`
  - Objects that can be mutated by some parties but not others
  - …

Prototypes

Want to have *actionable* advice for API developers.

Designed three language extensions for Java, addressing common use-cases

- Two pertain to what data is immutable
  - Transitive vs non-transitive
- One pertains to when it is immutable
  - Mutable during set-up process, fixed afterwards

Current status: two implemented, one in-progress

Class-Based Immutability Extensions

Syntax simple: annotate class with “@Immutable”

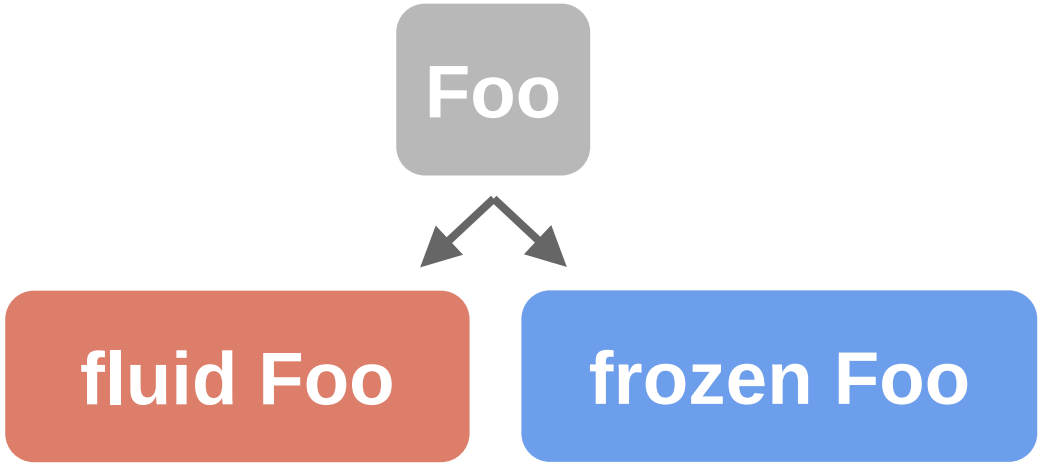
```
class AContainer {
        Date d; // Oops; needs to be immutable
        @Immutable AContainer () {
        }
}
```

Error message: Cannot declare AContainer() with annotation @Immutable because AContainer transitively contains d, which does not have a strong enough immutability annotation

Two variations: transitive vs non-transitive

- If object immutable, are objects that it refers to also immutable?
- Use cases for both. For example, if map from keys to values immutable, does that imply that the values themselves are unchangeable?

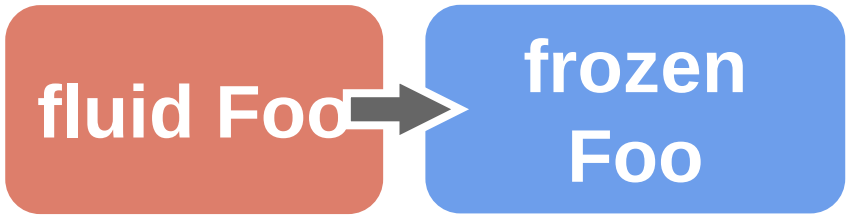
Immutability During Initialization



New type system

Objects can be mutable during initialization, immutable afterwards

(subtle rules about use of fluid objects)



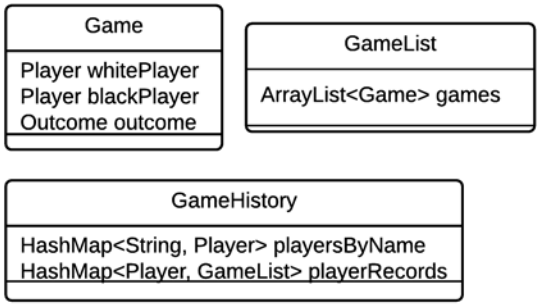

```
fluid Foo bar = new fluid Foo();
bar.baz = 0; // initialize the object
             // mutably
freeze bar;  // freeze the object to be
             // used immutably

rest_of_program(bar);
```

Evaluation

User-studies to show effectiveness of prototypes

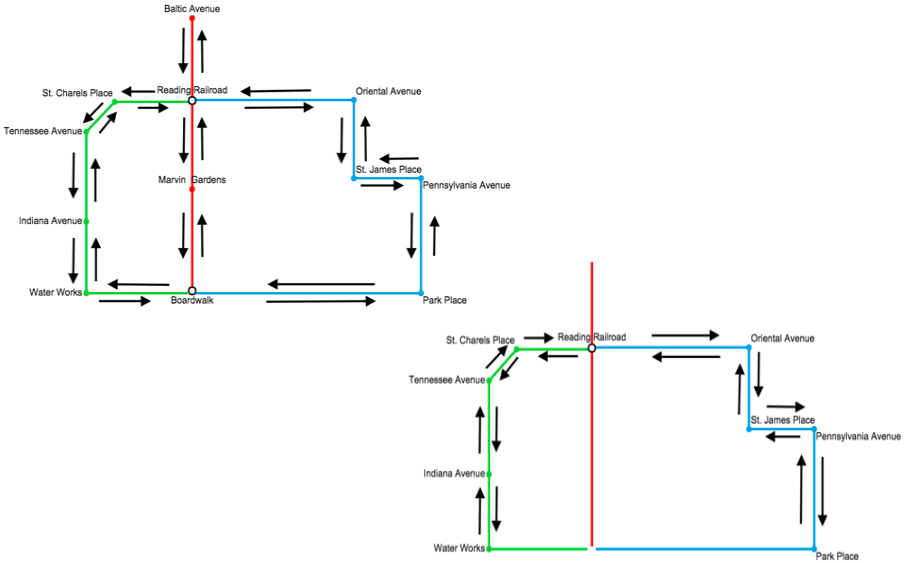
- Again, use participatory design techniques, including think-aloud protocols
  - Give developers tasks
  - Elicit how they would want to solve tasks, before telling them about features that we are testing
    - How do programmers conceptualize the problems?
  - Introduce features being tested
  - Have developers use features, and discuss out loud their thoughts, strategies and problems

Task Examples



Online game scenario

- Data structures to keep track of game state
- Programmers have to enhance/modify code



Bus routes

- Data structures to keep track of bus routes, including transfers between buses
- Create new routes without changing old ones

Project Summary

Goal: To develop and empirically test concrete and actionable API design principles that lead to more secure code

Initial Focus: State management

- How API-defined entities can change state, and by which parties

Project Plan

- Structured interviews of experienced developers
- Prototype solutions to address issues
- User evaluation studies of prototypes

Results and Future Work

- Identified design-space of state-change restrictions
- Interviews confirmed management of state to be serious problem, and current language features are not suitable
- Succeeded in showing that various kinds of immutability features are possible which better match developer needs
- Publications:
  - workshops: PLATEAU ‘15 (associated with SPLASH)
    LAW ‘14 (associated with ACSAC)
  - conferences: VL/HCC ‘15
    submission to ICSE ‘16

Future plans:

- Partners will continue ongoing work
- Partner emphasis will primarily shift towards usability