

Advances in Automated Program Repair *and* a Call to Arms

Westley Weimer
University of Virginia

Andreas Zeller, SSBSE Keynote 2011



For The Next Hour

- Automated Program Repair
- Historical Context
- Mistakes
- Opportunities

Speculative Fiction

What if large, trusted
companies paid strangers
to find and fix their
normal and critical bugs?

Microsoft Security Response Center

[HOME](#)[WHAT WE DO](#)[REPORT A VULNERABILITY](#)

Microsoft Security Bounty Programs

[Print](#)[Email](#)[Share](#)

Friends, hackers, researchers! We're looking for you to help us make some of our most popular products better? And earn money doing so? Step right up...

Microsoft is now offering direct cash payments in exchange for reporting certain types of vulnerabilities and exploitation techniques.

In 2002, Microsoft pioneered the Trustworthy Computing initiative to emphasize our commitment to doing what we believe best helps protect people's computing experience. In the years since, we introduced the Security Development Lifecycle (SDL) process to build more secure technologies. We also championed Coordinated Vulnerability Disclosure (CVD), formed industry collaboration programs such as MAPP and MSVR, and created the BlueHat Prize to encourage research into defensive technologies. Our new bounty programs add fresh depth and flexibility to our existing community outreach programs. Having these bounty programs provides a way to harness the collective intelligence and capabilities of security researchers to help further protect customers.

The following programs will launch on June 26, 2013:

1. **Mitigation Bypass Bounty.** Microsoft will pay up to \$100,000 USD for truly novel exploitation techniques against protections built into the latest version of our operating system (Windows 8.1 Preview). Learning about new exploitation techniques earlier helps Microsoft improve security by leaps, instead of capturing one vulnerability at a time as a traditional bug bounty alone would. *TIMEFRAME: ONGOING*
2. **BlueHat Bonus for Defense.** Additionally, Microsoft will pay up to \$50,000 USD for defensive ideas that accompany a qualifying Mitigation Bypass submission. Doing so highlights our continued support of defensive technologies and provides a way for the research community to help protect more than a billion computer systems worldwide. *TIMEFRAME: ONGOING (in conjunction with the Mitigation Bypass Bounty).*
3. **Internet Explorer 11 Preview Bug Bounty.** Microsoft will pay up to \$11,000 USD for

Microsoft Security Response Center

Personal Business

Email

Forgot?

Password

Forgot?

Log In

Sign Up



Buy ▾

Sell ▾

Transfer ▾

For Security Researchers

Bug Bounty Wall of Fame

For Customers: Reporting Suspicious Emails

Customers who think they have received a Phishing email, please learn more about phishing at https://cms.paypal.com/us/cgi-bin/marketingweb?cmd=_render-content&content_ID=security/hot_security_topics, or forward it to: spoof@paypal.com

For Customers: Reporting All Other Concerns

Customers who have issues with their PayPal Account, please visit: https://www.paypal.com/cgi-bin/helpscr?cmd=_help&t=escalateTab

For Professional Researchers: Bug Bounty Program

Our team of dedicated security professionals works vigilantly to help keep customer information secure. We recognize the important role that security researchers and our user community play in also helping to keep PayPal and our customers secure. If you discover a site or product vulnerability please notify us using the guidelines below.

Program Terms

Please note that your participation in the Bug Bounty Program is voluntary and subject to the terms and conditions set forth on this page ("Program Terms"). By submitting a site or product vulnerability to PayPal, Inc. ("PayPal") you acknowledge that you have read and agreed to these Program Terms.

These Program Terms supplement the terms of PayPal User Agreement, the PayPal Acceptable Use Policy, and any other agreement in which you have entered with PayPal (collectively "PayPal Agreements"). The terms of those PayPal Agreements will apply to your use of, and participation in, the Bug Bounty Program as if fully set forth herein. If there is any inconsistency exists between the terms of the PayPal Agreements and these Program Terms, these Program Terms will control, but only with regard to the Bug Bounty Program.

You can jump to particular sections of these Program Terms by using the following links:

[Responsible Disclosure Policy](#)

[Eligibility Requirements](#)

Microsoft has committed support of defensive technologies and provided a way for the research community to help protect more than a billion computer systems worldwide.
TIMEFRAME: ONGOING (in conjunction with the Mitigation Bypass Bounty).

Microsoft Security Response Center

Personal Business

Email

Forgot?

Password

Forgot?

Log In

Sign Up



Buy ▾

Sell ▾

Transfer ▾

AT&T Bug Bounty Program

For
Cu
co

For
Cu
Fo
Ou
us

Pro
Pl
as
Th
Pa
for
reg
Yo

Intro

Rewards

Report Bug

Hall of Fame

PRINT

EMAIL

Intro

Guidelines
Exclusions

Terms & Conditions

Already a Member?

Sign In

or [Join Now](#)

Welcome to the AT&T Bug Bounty Program! This program encourages and rewards contributions by developers and security researchers who help make AT&T's online environment more secure. Through this program AT&T provides monetary rewards and/or public recognition for security vulnerabilities responsibly disclosed to us.

The following explains the details of the program. To immediately start submitting your AT&T security bugs, please visit the [Bug Bounty submittal](#) page.

Guidelines

The AT&T Bug Bounty Program applies to security vulnerabilities found within AT&T's public-facing online environment. This includes, but not limited to, websites, exposed APIs, and mobile applications.

A security bug is an error, flaw, mistake, failure, or fault in a computer program or system that impacts the security of a device, system, network, or data. Any security bug may be considered for this program; however, it must be a new, previously unreported, vulnerability in order to be eligible for reward or recognition. Typically the in-scope submissions will include high impact bugs; however, any vulnerability at any severity might be rewarded.

Bugs which directly or indirectly affect the confidentiality or integrity of user data or privacy are prime candidates for reward. Any

Microsoft Security Response Center

[Personal](#)[Business](#)[Email](#)[Forgot?](#)[Password](#)[Forgot?](#)[Log In](#)[Sign Up](#)[Buy](#)[Sell](#)[Transfer](#)

For AT&T Bug Bounty Program

chat/bounty/

[bug bounties](#)[Sign Up](#)

Email or Phone

Password

 Keep me logged in[Forgot your password?](#)[Log](#)

Info



Thanks



Report Vulnerability

Information

If you are a security researcher, please review our responsible disclosure policy before reporting any vulnerabilities. If you are not a security researcher, visit the [Facebook Security Page](#) for assistance.

If you believe you have found a security vulnerability on Facebook, we encourage you to let us know right away. We will investigate all legitimate reports and do our best to quickly fix the problem.

Responsible Disclosure Policy

If you give us a reasonable time to respond to your report before making any information public and make a good faith effort to avoid privacy violations, destruction of data and interruption or degradation of our service during your research, we will not bring any lawsuit against you or ask law enforcement to investigate you.

Bug Bounty Info

To show our appreciation for our security researchers, we offer a monetary bounty for certain qualifying security bugs. Here is how it works:

Eligibility

To qualify for a bounty, you must:

- Adhere to our Responsible Disclosure Policy (above)
- Be the first person to responsibly disclose the bug
- Report a bug that could compromise the integrity of Facebook user data, circumvent the privacy protections of Facebook user data, or enable access to a system within the Facebook infrastructure, such as:
 - Cross-Site Scripting (XSS)

Microsoft Security Response Center

Personal

Business

Email

Forgot?

Password

Forgot?

Log In

Sign Up

PayPal™

Buy ▾

Sell ▾

Transfer ▾

For AT&T Bug Bounty Program

chat/bounty/

bug bounties

facebook

mozilla

About Us

Community Map

Our Projects

Get Involved



Bug Bounty Program

Introduction

The Mozilla Security Bug Bounty Program is designed to encourage security research in Mozilla software and to reward those who help us create the safest Internet clients in existence.

Many thanks to [Linspire](#) and [Mark Shuttleworth](#), who provided start-up funding for this endeavor.

General Bounty Guidelines

Mozilla will pay a bounty for certain client and service security bugs, as detailed below. All security bugs must follow the following general criteria to be eligible:

- Security bug must be original and previously unreported.

Microsoft Security Response Center

Personal

Business

Email

Forgot?

Password

Forgot?

Log In

Sign Up

PayPal™

Buy ▾

Sell ▾

Transfer ▾

For AT&T Bug Bounty Program

chat/bounty/

bug bounties

facebook

mozilla

About Us

Community Map

Our Projects

Get Involved

Info

Thanks

Report Vulnerability

Google | Application Security

Home Tools Vulnerability Reward Program Hall of Fame Research

Program Rules

We have long enjoyed a close relationship with the security research community. To honor all the cutting-edge external contributions that help us keep Google safe, we maintain a Vulnerability Reward Program for Google-owned web properties, running continuously since November 2010.

Services in scope

In principle, any Google-owned web service that handles reasonably sensitive user data is intended to be in scope. This includes virtually all the company's domains:

- *.google.com
- *.youtube.com
- *.blogger.com
- *.orkut.com

The program has four key exclusions:

- Non-web applications are generally not in scope. We make special exceptions for Google Wallet and Google Chrome. The Chrome reward pro-

Microsoft Security Response Center

PERSONAL BUSINESS

Email

Forgot?

Password

Forgot?

Log in

Sign up

PayPal™

Buy ▾

Sell ▾

Transfer ▾

(Raise hand if true)

I have used software produced by Microsoft, PayPal, AT&T, Facebook, Mozilla, Google or YouTube.

In principle, any Google-owned web service that handles reasonably sensitive user data is intended to be in scope. This includes virtually all the following domains:

- *.google.com
- *.youtube.com
- *.blogger.com
- *.orkut.com

The program has four key exclusions:

- Non-web applications are generally not in scope. We make special exceptions for Google Wallet and Google Chrome. The Chrome reward pro-

Client Reward Guidelines

The bounty for valid critical client security bugs will be \$3000 (US) cash reward and a Mozilla T-shirt. The bounty will be awarded for [sec-critical](#) and [sec-high](#) security bugs that meet the following criteria:

- Security bug is present in the most recent main development (i.e., Aurora, Beta or EarlyBird, and nightly mozilla-central releases) or released versions of Firefox, Thunderbird, Firefox for Android or in Mozilla services which could compromise users of those products, as released by Mozilla Corporation.
- Security bugs in or caused by additional 3rd-party software (e.g. plugins, extensions) are excluded from the Bug Bounty program.

More information about this program can be found in the [Client Security Bug Bounty Program FAQ](#).

Web Application and Services Reward Guidelines

The bounty for valid web applications or services related security bugs, we are giving a range starting at \$500 (US) for high severity and, in some cases, may pay up to \$3000 (US) for extraordinary or critical vulnerabilities. We will also include a Mozilla T-shirt. The bounty will be awarded for [sec-critical](#) and [sec-high](#) security bugs that meet the following criteria:

- Security bug is present in the web properties outlined in the [Web Application Security Bounty FAQ](#)
- Security bug is on the list of sites which part of the bounty. See the [eligible bugs](#) section of the [Web Application Security Bounty FAQ](#) for the list of sites which is included under the bounty.

Client Reward Guidelines

The bounty for valid critical client security bugs will be \$2000 (US) cash reward and a Mozilla T-shirt

Estimated payout ranges⁺ (in USD) for in-scope vulnerabilities are as follows:

Vulnerability	.paypal.com and PayPal subsidiary websites	Partner sites (www.paypal-__.com)
Remote Code Execution	Up to \$10,000	\$1,500
SQL Injection	Up to \$5,000	\$1,000
Authentication Bypass	Up to \$3,000	\$1,000
Cross-Site Scripting (XSS)	\$750	\$100
Information Disclosure of Sensitive Data	\$750	\$100
Clickjacking [#]	\$750	0
Cross-Site Request Forgery (CSRF) [#]	\$750	0

sec-high security bugs that meet the following criteria:

- Security bug is present in the web properties outlined in the [Web Application Security Bounty FAQ](#)
- Security bug is on the list of sites which part of the bounty. See the [eligible bugs](#) section of the [Web Application Security Bounty FAQ](#) for the list of sites which is included under the bounty.

Client Reward Guidelines

The bounty for valid critical client security bugs will be \$2000 (US) cash reward and a Mozilla T-shirt

Estimated payout ranges⁺ (in USD) for in-scope vulnerabilities are as follows:
will acknowledge your contribution on that page.

Reward amounts

Rewards for qualifying bugs range from \$100 to \$20,000. The following table outlines the usual rewards chosen for the most common class

	accounts.google.com	Other highly sensitive services [1]	Normal Google applications	Non-integrated lower priority
Remote code execution	\$20,000	\$20,000	\$20,000	\$1,337 - \$5,000
SQL injection or equivalent	\$10,000	\$10,000	\$10,000	\$1,337 - \$5,000
Significant authentication bypass or information leak	\$10,000	\$7,500	\$5,000	\$500
Typical XSS	\$7,500	\$5,000	\$3,133.7	\$100
XSRF, XSS and other common web flaws	\$500 - \$3,133.7	\$500 - \$1,337	\$500	\$100

[1] This category includes products such as Google Search (<https://www.google.com>), Google Wallet (<https://wallet.google.com>), Google Mail (<https://mail.google.com>), Google Code Hosting (<code.google.com>), Chrome Web Store (<https://chrome.google.com>), and Google Play

- Security bug is present in the web properties outlined in the [Web Application Security Bounty FAQ](#)
- Security bug is on the list of sites which part of the bounty. See the [eligible bugs](#) section of the [Web Application Security Bounty FAQ](#) for the list of sites which is included under the bounty.

Client Reward Guidelines

The bounty for valid critical client security bugs will be \$2000 (US) cash reward and a Mozilla T-shirt.

Estimated payout ranges⁺ (in USD) for in-scope vulnerabilities are as follows:
will acknowledge your contribution on that page.

Reward amounts

Rewards for qualifying bugs range from \$100 to \$20,000. The following table outlines the usual rewards chosen for the most common class when it is reported, different bounties will be awarded:

Bounty value	Pre-release bounty value	Type of bug
\$1000	\$2000	A bug which allows someone intercepting Tarsnap traffic to decrypt Tarsnap users' data.
\$500	\$1000	A bug which allows the Tarsnap service to decrypt Tarsnap users' data.
\$500	\$1000	A bug which causes data corruption or loss.
\$100	\$200 	A bug which causes Tarsnap to crash (without corrupting data or losing any data other than an archive currently being written).
\$50	\$100	Any other non-harmless bugs in Tarsnap.
\$20	\$40	Build breakage on a platform where a previous Tarsnap release worked.
\$10	\$20 	"Harmless" bugs, e.g., cosmetic errors in Tarsnap output or mistakes in source code comments.
\$1	\$2	Cosmetic errors in the Tarsnap source code or website, e.g., typos in website text or source code comments. Style errors in Tarsnap code qualify here, but usually not style errors in upstream code (e.g., libarchive).

Client Reward Guidelines

The bounty for valid critical client security bugs will be \$2000 (US) cash reward and a Mozilla T-shirt.

Estimated payout ranges⁺ (in USD) for in-scope vulnerabilities are as follows:
will acknowledge your contribution on that page.

Even though only 38% of the submissions were true positives (harmless, minor or major):

“Worth the money? Every penny.”

\$20	\$40	Build breakage on a platform where a previous Tarsnap release worked.
\$10	\$20	"Harmless" bugs, e.g., cosmetic errors in Tarsnap output or mistakes in source code comments.
\$1	\$2	Cosmetic errors in the Tarsnap source code or website, e.g., typos in website text or source code comments. Style errors in Tarsnap code qualify here, but usually not style errors in upstream code (e.g., libarchive).

Client Reward Guidelines

The bounty for valid critical client security bugs will be \$2000 (US) cash reward and a Mozilla T-shirt.

Estimated payout ranges⁺ (in USD) for in-scope vulnerabilities are as follows:
will acknowledge your contribution on that page.

"We get hundreds of reports every day. Many of our best reports come from people whose English isn't great - though this can be challenging, it's something we work with just fine and we have paid out over \$1 million to hundreds of reporters."

- Matt Jones, Facebook Software Engineer

\$20	\$40	Build breakage on a platform where a previous Tarsnap release worked.
\$10	\$20 	"Harmless" bugs, e.g., cosmetic errors in Tarsnap output or mistakes in source code comments.
\$1	\$2	Cosmetic errors in the Tarsnap source code or website, e.g., typos in website text or source code comments. Style errors in Tarsnap code qualify here, but usually not style errors in upstream code (e.g., libarchive).

customers.

The following programs will launch on June 26, 2013:

1. **Mitigation Bypass Bounty.** Microsoft will pay up to \$100,000 USD for truly novel exploitation techniques against protections built into the latest version of our operating system (Windows 8.1 Preview). Learning about new exploitation techniques earlier helps Microsoft improve security by leaps, instead of capturing one vulnerability at a time as a traditional bug bounty alone would. *TIMEFRAME: ONGOING*
2. **BlueHat Bonus for Defense.** Additionally, Microsoft will pay up to \$50,000 USD for defensive ideas that accompany a qualifying Mitigation Bypass submission. Doing so highlights our continued support of defensive technologies and provides a way for the research community to help protect more than a billion computer systems worldwide. *TIMEFRAME: ONGOING (in conjunction with the Mitigation Bypass Bounty).*
3. **Internet Explorer 11 Preview Bug Bounty.** Microsoft will pay up to \$11,000 USD for critical vulnerabilities that affect Internet Explorer 11 Preview on the latest version of Windows (Windows 8.1 Preview). The entry period for this program will be the first 30 days of the Internet Explorer 11 beta period (June 26 to July 26, 2013). Learning about critical vulnerabilities in Internet Explorer as early as possible during the public preview will help Microsoft make the newest version of the browser more secure. *TIMEFRAME: 30 DAYS*

Want to know more?

A vision of the ~~future~~ present

Finding, fixing and ignoring bugs are all so expensive that it is **now** economical to pay untrusted strangers to submit candidate defect reports and patches.

A Modest Proposal

Automatically find and fix defects (rather than, or in addition to, paying strangers).

Outline

- Automated Program Repair
- The State of the Art
 - Scalability and Recent Growth
- GenProg Lessons Learned (the fun part)
- Challenges & Opportunities
 - Test Suite Quality and Oracles
 - Reproducible Research & Benchmarks
 - Large Human Studies

Historical Context



“We are moving to a new era where software systems are open, evolving and not owned by a single organization. Self-* systems are not just a nice new way to deal with software, but a necessity for the coming systems. The big new challenge of self-healing systems is to guarantee stability and convergence: we need to be able to master our systems even **without knowing in advance what will happen** to them.”

- Mauro Pezzè, Milano Bicocca / Lugano

Historical Context

- <= 1975 “Software fault tolerance”
 - Respond with minimal disruption to an unexpected software failure. Often uses isolation, mirrored fail-over, transaction logging, etc.
- ~1998: “Repairing one type of security bug”
 - [Cowan, Pu, Maier, Walpole, Bakke, Beattie, Grier, Wagle, Zhang, Hinton. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. USENIX Security 1998.]
- ~2002: “Self-healing (adaptive) systems”
 - Diversity, redundancy, system monitoring, models
 - [Garlan, Kramer, Wolf (eds). First Workshop on Self-Healing Systems, 2002.]

Why not just restart?

- Imagine two types of problems:
 - **Non-deterministic** (e.g., environmental): A network link goes down, `send()` raises an exception
 - **Deterministic** (e.g., algorithmic): The first line of `main()` dereferences a null pointer
- Failure-transparent or transactional approaches usually restart the same code
 - What if there is a deterministic bug in that code?

Checkpoint and Restart



[Lowell, Chandra, Chen: Exploring Failure Transparency and the Limits of Generic Recovery. OSDI 2000.]

Groundhog Day



[Lowell, Chandra, Chen: Exploring Failure Transparency and the Limits of Generic Recovery. OSDI 2000.]

Early “Proto” Program Repair Work

- 1999: **Delta debugging** [Zeller: Yesterday, My Program Worked. Today, It Does Not. Why? ESEC / FSE 1999.]
- 2001: **Search-based software engineering** [Harman, Jones. Search based software engineering. Information and Software Technology, 43(14) 2001]
- 2003: **Data structure repair**
 - Run-time approach based on constraints [Demsky, Rinard: Automatic detection and repair of errors in data structures. OOPSLA 2003.]
- 2006: **Repairing safety policy violations**
 - Static approach using formal FSM specifications [Weimer: Patches as better bug reports. GPCE 2006.]
- 2008: **Genetic programming proposal** [Arcuri: On the automation of fixing software bugs. ICSE Companion 2008.]

General Automated Program Repair

- Given a program ...
 - Source code, assembly code, binary code
- ... and evidence of a bug ...
 - Passing and failing test cases, implicit specifications and crashes, preconditions and invariants, normal and anomalous runs
- ... fix that bug.
 - A textual patch, a dynamic jump to new code, run-time modifications to variables

How could that work?

- Many faults can be **localized** to a small area
 - [Jones, Harrold. Empirical evaluation of the Tarantula automatic fault-localization technique. ASE 2005.]
 - [Qi, Mao, Lei, Wang. Using Automated Program Repair for Evaluating the Effectiveness of Fault Localization Techniques. ISSTA 2013.]
- Many defects can be fixed with **small changes**
 - [Park, Kim, Ray, Bae: An empirical study of supplementary bug fixes. MSR 2012.]
- Programs can be **robust** to such changes
 - “Only attackers and bugs care about unspecified, untested behavior.”
 - [Schulte, Fry, Fast, Weimer, Forrest: Software Mutational Robustness. J. GPEM 2013.]

Scalability and Recent Growth



2009: A Banner Year

GenProg

Genetic programming evolves source code until it passes the rest of a test suite. [Weimer, Nguyen, Le Goues, Forrest: Automatically finding patches using genetic programming. ICSE May 2009.]

ClearView

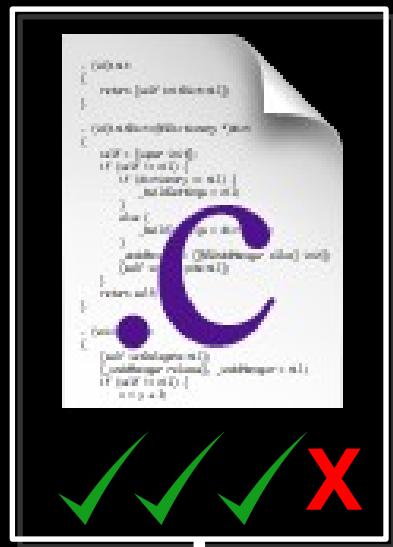
Detects normal workload invariants and anomalies, deploying binary repairs to restore invariants.

[Perkins, Kim, Larsen, Amarasinghe, Bachrach, Carbin, Pacheco, Sherwood, Sidiropoulos, Sullivan, Wong, Zibin, Ernst, Rinard: Automatically patching errors in deployed software. SOSP Oct 2009.]

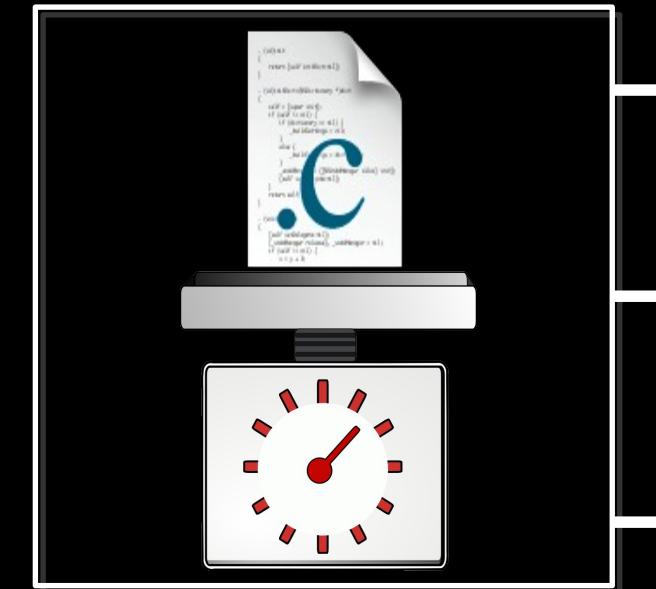
PACHIKA

Summarizes test executions to behavior models, generating fixes based on the differences. [Dallmeier, Zeller, Meyer: Generating Fixes from Object Behavior Anomalies. ASE Nov 2009.]

GenProg



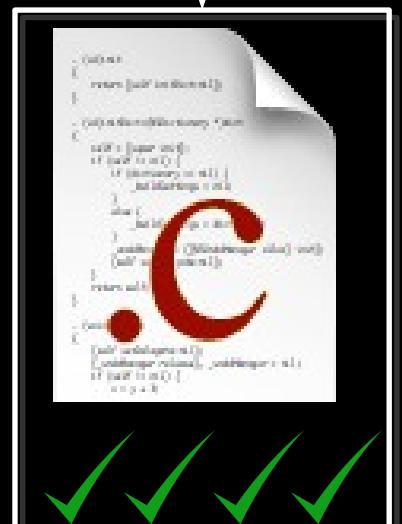
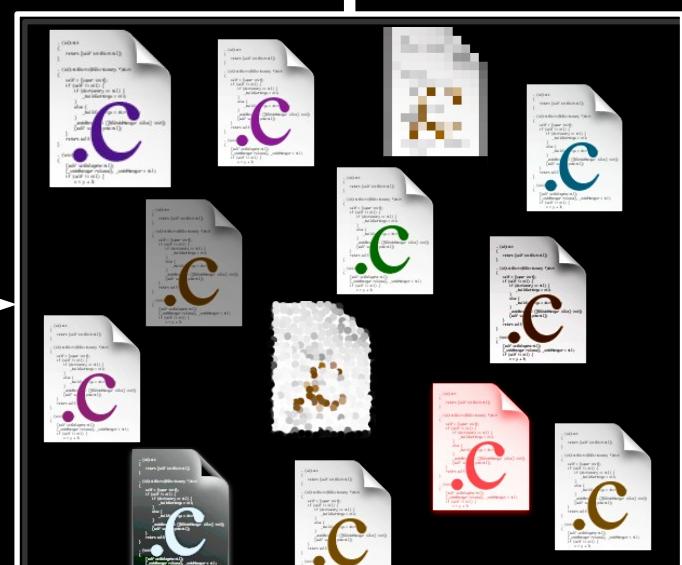
EVALUATE FITNESS



DISCARD



ACCEPT



2009 In A Nutshell

- Given a **program** and **tests** (or a workload)
 - Normal observations: ABC or ABCD
- A **problem** is detected
 - Failing observations: ABX C
- The difference yields **candidate repairs**
 - { “Don't do X”, “Always do D” }
- One repair **passes all tests**
 - Report “Don't do X” as the patch

Two Broad Repair Approaches

- Single Repair or “Correct by Construction”
 - Careful consideration (constraint solving, invariant reasoning, lockset analysis, type systems, etc.) of the problem produces a single good repair.
- Generate-and-Validate
 - Various techniques (mutation, genetic programming, invariant reasoning, etc.) produce multiple candidate repairs.
 - Each candidate is evaluated and a valid repair is returned.

Name	Subjects	Tests	Bugs	Notes
AFix	2 Mloc	–	8	Concurrency, guarantees
ARC	–	–	–	Concurrency, SBSE
ARMOR	6 progs.	–	3 + –	Identifies workarounds
Axis	13 progs.	–	–	Concurrency, guarantees, Petri nets
AutoFix-E	21 Kloc	650	42	Contracts, guarantees
CASC	1 Kloc	–	5	Co-evolves tests and programs
ClearView	Firefox	57	9	Red Team quality evaluation
Coker Hafiz	15 Mloc	–	7 / –	Integer bugs only, guarantees
Debroy Wong	76 Kloc	22,500	135	Mutation, fault localization focus
Demsky <i>et al.</i>	3 progs.	–	–	Data struct consistency, Red Team
FINCH	13 tasks	–	–	Evolves unrestricted bytecode
GenProg	5 Mloc	10,000	105	Human-competitive, SBSE
Gopinath <i>et al.</i>	2 methods.	–	20	Heap specs, SAT
Jolt	5 progs.	–	8	Escape infinite loops at run-time
Juzi	7 progs.	–	20 + –	Data struct consistency, models
PACHIKA	110 Kloc	2,700	26	Differences in behavior models
PAR	480 Kloc	25,000	119	Human-based patches, quality study
SemFix	12 Kloc	250	90	Symex, constraints, synthesis
Sidiropoulos <i>et al.</i>	17 progs.	–	17	Buffer overflows

Name	Subjects	Tests	Bugs	Notes
AFix	2 Mloc	–	8	Concurrency, guarantees
ARC	–	–	–	Concurrency, SBSE
ARMOR	6 progs.	–	3 + –	Identifies workarounds
Axis	13 progs.	–	–	Concurrency, guarantees, Petri nets
AutoFix-E	21 Kloc	650	42	Contracts, guarantees
CASC	1 Kloc	–	5	Co-evolves tests and programs
ClearView	Firefox	57	9	Red Team quality evaluation
Coker Hafiz	15 Mloc	–	7 / –	Integer bugs only, guarantees
Debroy Wong	76 Kloc	22,500	135	Mutation, fault localization focus
Demsky <i>et al.</i>	3 progs.	–	–	Data struct consistency, Red Team
FINCH	13 tasks	–	–	Evolves unrestricted bytecode
GenProg	5 Mloc	10,000	105	Human-competitive, SBSE
Gopinath <i>et al.</i>	2 methods.	–	20	Heap specs, SAT
Jolt	5 progs.	–	8	Escape infinite loops at run-time
Juzi	7 progs.	–	20 + –	Data struct consistency, models
PACHIKA	110 Kloc	2,700	26	Differences in behavior models
PAR	480 Kloc	25,000	119	Human-based patches, quality study
SemFix	12 Kloc	250	90	Symex, constraints, synthesis
Sidiropoulos <i>et al.</i>	17 progs.	–	17	Buffer overflows

Name	Subjects	Tests	Bugs	Notes
AFix	2 Mloc	–	8	Concurrency, guarantees
ARC	–	–	–	Concurrency, SBSE
ARMOR	6 progs.	–	3 + –	Identifies workarounds
Axis	13 progs.	–	–	Concurrency, guarantees, Petri nets
AutoFix-E	21 Kloc	650	42	Contracts, guarantees
CASC	1 Kloc	–	5	Co-evolves tests and programs
ClearView	Firefox	57	9	Red Team quality evaluation
Coker Hafiz	15 Mloc	–	7 / –	Integer bugs only, guarantees
Debroy Wong	76 Kloc	22,500	135	Mutation, fault localization focus
Demsky <i>et al.</i>	3 progs.	–	–	Data struct consistency, Red Team
FINCH	13 tasks	–	–	Evolves unrestricted bytecode
GenProg	5 Mloc	10,000	105	Human-competitive, SBSE
Gopinath <i>et al.</i>	2 methods.	–	20	Heap specs, SAT
Jolt	5 progs.	–	8	Escape infinite loops at run-time
Juzi	7 progs.	–	20 + –	Data struct consistency, models
PACHIKA	110 Kloc	2,700	26	Differences in behavior models
PAR	480 Kloc	25,000	119	Human-based patches, quality study
SemFix	12 Kloc	250	90	Symex, constraints, synthesis
Sidiropoulos <i>et al.</i>	17 progs.	–	17	Buffer overflows

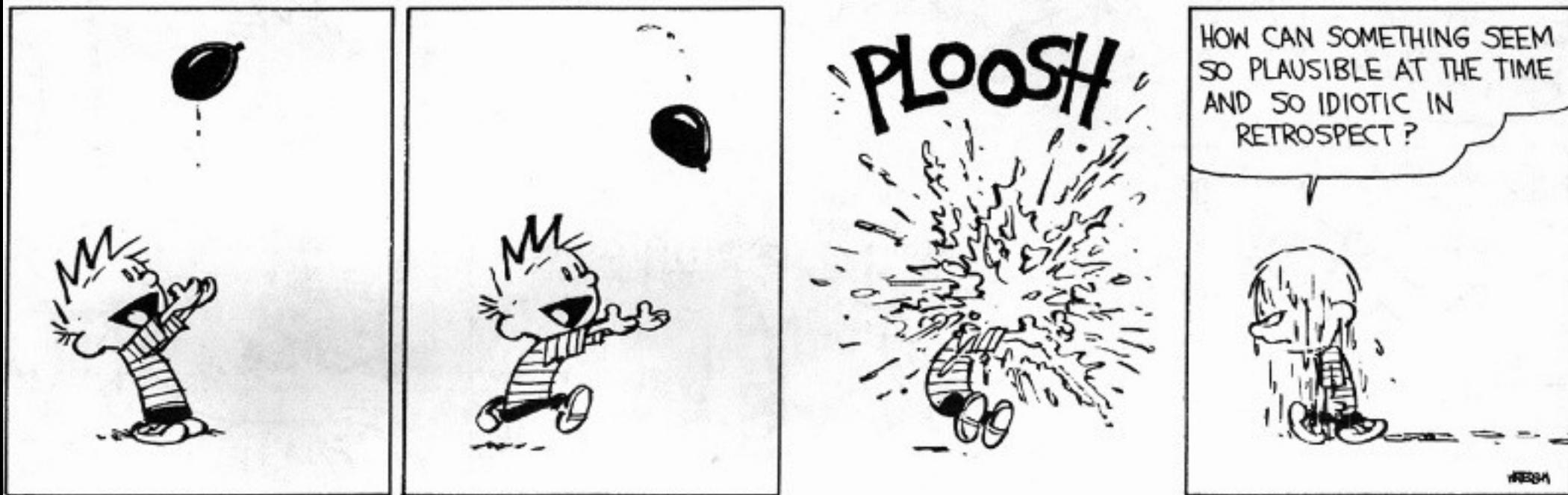
Name	Subjects	Tests	Bugs	Notes
AFix	2 Mloc	–	8	Concurrency, guarantees
ARC	–	–	–	Concurrency, SBSE
ARMOR	6 progs.	–	3 + –	Identifies workarounds
Axis	13 progs.	–	–	Concurrency, guarantees, Petri nets
AutoFix-E	21 Kloc	650	42	Contracts, guarantees
CASC	1 Kloc	–	5	Co-evolves tests and programs
ClearView	Firefox	57	9	Red Team quality evaluation
Coker Hafiz	15 Mloc	–	7 / –	Integer bugs only, guarantees
Debroy Wong	76 Kloc	22,500	135	Mutation, fault localization focus
Demsky <i>et al.</i>	3 progs.	–	–	Data struct consistency, Red Team
FINCH	13 tasks	–	–	Evolves unrestricted bytecode
GenProg	5 Mloc	10,000	105	Human-competitive, SBSE
Gopinath <i>et al.</i>	2 methods.	–	20	Heap specs, SAT
Jolt	5 progs.	–	8	Escape infinite loops at run-time
Juzi	7 progs.	–	20 + –	Data struct consistency, models
PACHIKA	110 Kloc	2,700	26	Differences in behavior models
PAR	480 Kloc	25,000	119	Human-based patches, quality study
SemFix	12 Kloc	250	90	Symex, constraints, synthesis
Sidiropoulos <i>et al.</i>	17 progs.	–	17	Buffer overflows

Name	Subjects	Tests	Bugs	Notes
AFix	2 Mloc	–	8	Concurrency, guarantees
ARC	–	–	–	Concurrency, SBSE
ARMOR	6 progs.	–	3 + –	Identifies workarounds
Axis	13 progs.	–	–	Concurrency, guarantees, Petri nets
AutoFix-E	21 Kloc	650	42	Contracts, guarantees
CASC	1 Kloc	–	5	Co-evolves tests and programs
ClearView	Firefox	57	9	Red Team quality evaluation
Coker Hafiz	15 Mloc	–	7 / –	Integer bugs only, guarantees
Debroy Wong	76 Kloc	22,500	135	Mutation, fault localization focus
Demsky <i>et al.</i>	3 progs.	–	–	Data struct consistency, Red Team
FINCH	13 tasks	–	–	Evolves unrestricted bytecode
GenProg	5 Mloc	10,000	105	Human-cooperative, SBSE
Gopinath <i>et al.</i>	2 methods.	–	20	Heap specs, SAT
Jolt	5 progs.	–	8	Escape infinite loops at run-time
Juzi	7 progs.	–	20 + –	Data struct consistency, models
PACHIKA	110 Kloc	2,700	26	Differences in behavior models
PAR	480 Kloc	25,000	119	Human-based patches, quality study
SemFix	12 Kloc	250	90	Symex, constraints, synthesis
Sidiropoulos <i>et al.</i>	17 progs.	–	17	Buffer overflows

State of the Art

- 2009: 15 papers on auto program repair
 - (Manual search/review of ACM Digital Library)
- 2011: Dagstuhl on Self-Repairing Programs
- 2012: 30 papers on auto program repair
 - At least 20+ different approaches, 3+ best paper awards, etc.
- 2013: ICSE has a “Program Repair” session
- So now let's talk about the *seamy underbelly*.

Lessons Learned



Lessons Learned: Test Quality

- Automated program repair is a whiny child:
 - “You only said I had *get into* the bathtub, you didn't say I had to wash.”

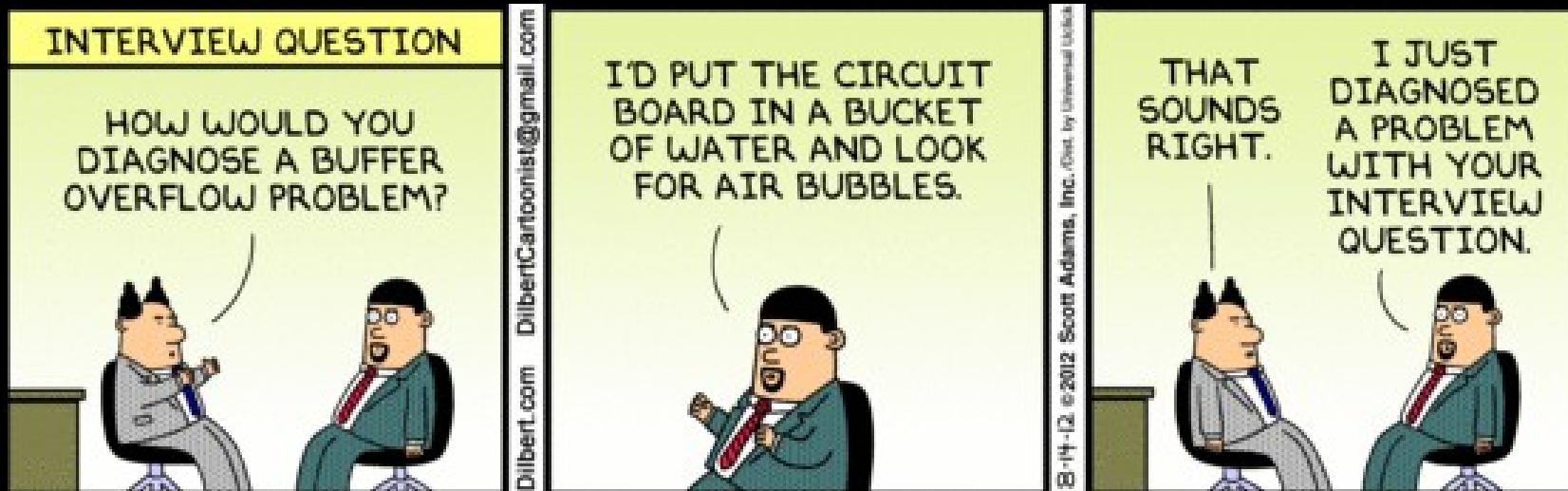


Lessons Learned: Test Quality

- Automated program repair is a whiny child:
 - “You only said I had *get into* the bathtub, you didn't say I had to wash.”
- GenProg Day 1: gcd, nullhttpd
 - 5 tests for nullhttpd (GET index.html, etc.)
 - 1 bug (POST → remote exploit)
 - GenProg's fix: remove POST functionality
 - (Adding a 6th test yields a high-quality repair.)

Lessons Learned: Test Quality (2)

- MIT Lincoln Labs test of GenProg: sort
 - Tests: “the output of sort is in sorted order”
 - GenProg's fix: “always output the empty set”
 - (More tests yield a higher quality repair.)



Lessons Learned: Test Framework

- GenProg: binary / assembly repairs
 - Tests: “compare your-output.txt to trusted-output.txt”
 - GenProg's fix: “delete trusted-output.txt, output nothing”
- “Garbage In, Garbage Out”



Lessons Learned: Integration

- Integrating GenProg with a real program's test suite is non-trivial
- Example: spawning a child process
 - `system("run test cmd 1 ..."); wait();`
 - `wait()` returns the error status
 - Can fail because the OS ran out of memory or because the child process ran out of memory
 - Unix answer: bit shifting and masking!

Lessons Learned: Integration (2)

- We had instances where PHP's test harness and GenProg's test harness wrapper disagreed on this bit shifting
 - GenProg's fix: “always segfault, which will mistakenly register as 'test passed' due to mis-communicated bit shifting”
- Think of deployment at a company:
 - Whose “fault” or “responsibility” is this?

Lessons Learned: Integration (3)

- GenProg has to be able to compile candidate patches
 - Just run “make”, right?
- Some programs, such as language interpreters, bootstrap or self-host.
 - We expected and handled infinite loops in tests
 - We did not expect infinite loops in compilation

Lessons Learned: Sandboxing

- GenProg has created ...
 - Programs that kill the parent shell
 - Programs that “sleep forever” to avoid CPU-usage tests for infinite loops
 - Programs that allocate memory in an infinite loop, causing the Linux OOM killer to randomly kill GenProg
 - Programs that email developers so often that Amazon EC2 gave us the “we think you’re a spammer” warning

Lessons Learned: Poor Tests

- Large open source programs have tests like:
 - Pass if today is less than December 31, 2012

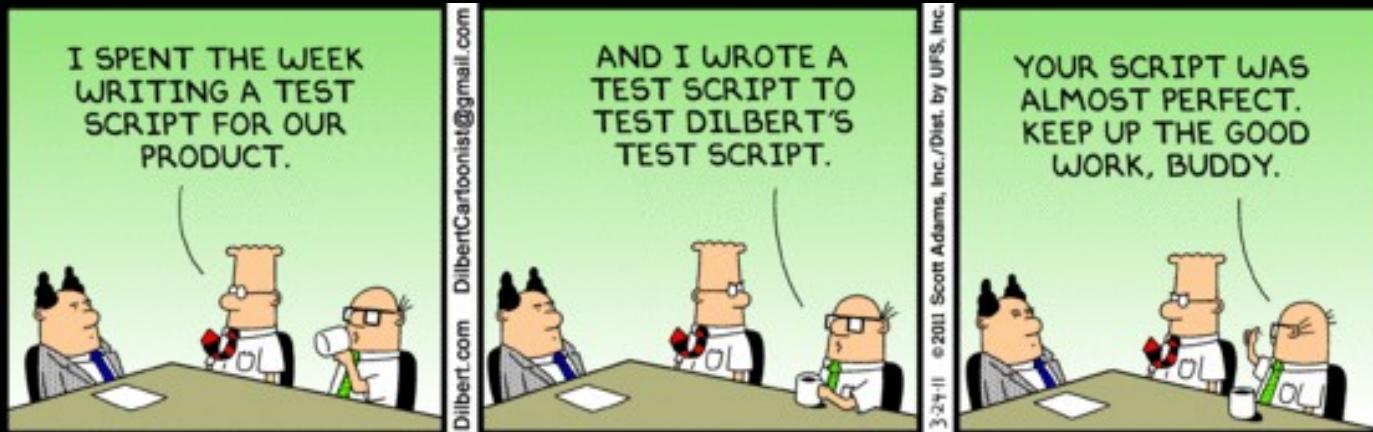


Lessons Learned: Poor Tests

- Large open source programs have tests like:
 - Pass if today is less than December 31, 2012
 - Check that the modification times of files in this directory are equal to my hard-coded values
 - Generate a random ID with prefix “999”, check to see if result starts with “9996” (dev typo)

Lessons Learned: Sanity

- Our earliest concession to reality was the addition of a “sanity check” to GenProg:
 - Does the program actually compile? Pass all non-bug tests? Fail all bug tests?
- A large fraction of our early reproduction difficulties were caught at this stage.



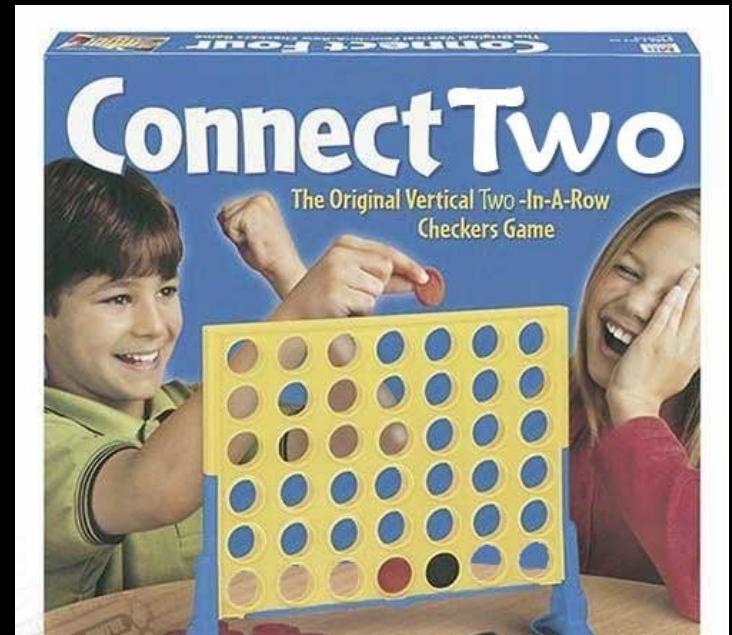
A Call To Arms



Challenges and Opportunities

- Test Suite Quality & Oracles
- Benchmarking & Reproducible Research
- Human Studies

Challenge: Test Suite Quality and Oracles



“A generated repair is the ultimate diagnosis in automated debugging - it tells the programmer where to fix the bug, what to fix, and how to fix it as to minimize the risk of new errors. A good repair depends on a good specification, though; and maybe the advent of good repair tools will entice programmers in improving their specifications in the first place.”

- Andreas Zeller, Saarland University

Test Suite Quality & Oracles

- $\text{Repair_Quality} = \min(\text{Technique}, \text{Test Suite})$
- Currently, we trust the test suppliers
- What if we spent time on writing good specifications instead of on debugging?
- Charge: **measure** the suites we are using or **generate** high-quality suites to use
- Analogy: Formal Verification
 - Difficulty depends on more than program size

Test Data Generation

- We have all agreed to believe that we can create **high-coverage test inputs**



Test Data Generation

- We have all agreed to believe that we can **create high-coverage test inputs**
 - DART, CREST, CUTE, KLEE, AUSTIN, SAGE, PEX ...
 - Randomized, search-based, constraint-based, concrete and symbolic execution, ...
 - [Cadar, Sen: Symbolic execution for software testing: three decades later. Commun. ACM 56(2), 2013.]

Test Data Generation

- We have all agreed to believe that we can **create high-coverage test inputs**
 - DART, CREST, CUTE, KLEE, AUSTIN, SAGE, PEX ...
 - Randomized, search-based, constraint-based, concrete and symbolic execution, ...
 - [Cedar, Sen: Symbolic execution for software testing: three decades later. Commun. ACM 56(2), 2013.]
- “And if it crashes on that input, that's bad.”

Test Oracle Generation

- What should the program be doing?
- **μTEST** [Fraser, Zeller: Mutation-Driven Generation of Unit Tests and Oracles. IEEE Trans. Software Eng. 38(2), 2012]
 - Great combination: Daikon + mutation analysis
 - Generate a set of candidate invariants
 - Running the program removes non-invariants
 - Retain only the useful ones: those killed by mutants
- [Staats, Gay, Heimdahl: Automated oracle creation support, or: How I learned to stop worrying about fault propagation and love mutation testing. ICSE 2012.]
- [Nguyen, Kapur, Weimer, Forrest: Using dynamic analysis to discover polynomial and array invariants. ICSE 2012.]

Specification Mining

- Given a program (and possibly an indicative workload), **generate partial-correctness specifications** that describe proper behavior.
[Ammons, Bodík, Larus: Mining specifications. POPL 2002.]
 - “Learn the rules of English grammar by reading student essays.”
- Problem: **common** behavior need not be **correct** behavior.
- Mining is most useful when the program deviates from the specification.

Spec Mining ≈ Oracle Generation

- Probabilistic FSM Learning
- Normal vs. Exceptional Paths, Code Quality Metrics [Le Goues, Weimer: Measuring Code Quality to Improve Specification Mining. IEEE Trans. Software Eng. 38(1), 2012.]
- Symbolic Automata + Abstract Domains [Peleg, Shoham, Yahav, Yang: Symbolic Automata for Static Specification Mining. SAS 2013.]
- Interprocedural static analysis and anomaly detection [Wasylkowski, Zeller, Lindig: Detecting object usage anomalies. ESEC/FSE 2007.]
- Word equations and quantifiers [Ganesh, Minnes, Solar-Lezama, Rinard: Word Equations with Length Constraints: What's Decidable? Haifa Verification, 2012.]

A Reasonable Goal

- Perhaps we wanted a Large Step in semantics
 - Inputs → Inputs + full-correctness test oracles
- I propose an intermediate step
 - Test inputs plus **partial**-correctness test oracles
- Research program: combine a subset of
 - Invariant generation
 - Mutation testing
 - **Specification mining**

Challenge:

Benchmarking



“One of the challenges will be to identify the situations when and where automated program repair can be applied. I don't expect that program repair will work for every bug in the universe (otherwise thousands of developers will become unemployed), but if we can identify the areas where it works in advance there is lots of potential.”

- Thomas Zimmermann, Microsoft

Benchmarking

- Reproducible research, results that generalize
- “Benchmarks set standards for innovation, and can encourage or stifle it.” [Blackburn *et al.*: The DaCapo benchmarks: Java benchmarking development and analysis. OOPSLA 2006.]
- We desire:
 - Latitudinal studies: many bugs and programs
 - Longitudinal studies: many bugs in one program
 - Comparative studies: many tools on the same bugs

Test Guidelines

- Test desiderata, from a program repair perspective:
 - Can the empty program pass it?
 - Can an infinite loop pass it?
 - Can an always-segfault program pass it?
- “if it completes in 10 seconds then pass”
- “if not grep(output,bad_string) then pass”

Number of the 15 papers
presented at SSBSE 2012 that
used the **same evaluation**
subject as another SSBSE
2012 paper:

?

Number of the 15 papers
presented at SSBSE 2012 that
used the **same evaluation**
subject as another SSBSE
2012 paper:

Zero.

Commonalities

- Many papers are on entirely new areas
- But, from titles alone ...
 - 2 studied threads or concurrency
 - 2 studied randomness
 - 5 studied testing
- It's not impossible to imagine *one* benchmark in common.

SSBSE 2012



Charge

- As reviewers, **acknowledge** benchmark creation as a scientific contribution
- As researchers, **create** benchmarks
- It does not have to be a sacrifice:
 - Siemens benchmarks paper >600 citations
 - DaCapo benchmarks paper >600 citations
 - PARSEC benchmark paper >1000 citations

Challenge: Human Studies



One Way To Turn Good Into Great

With all papers considered, those with user evaluations do *not* have higher citation counts overall. However, when attention is restricted to highly-cited works, user evaluations are relevant: for example, among the top quartile of papers by citation count, papers with user evaluations are cited 40% more often than papers without. Highly-selective conferences accept a larger proportion of papers with user evaluations than do less-selective conferences.

(3,000+ papers from ASE, ESEC/FSE, ICSE, ISSTA, OOPSLA, etc., 2000-2010)

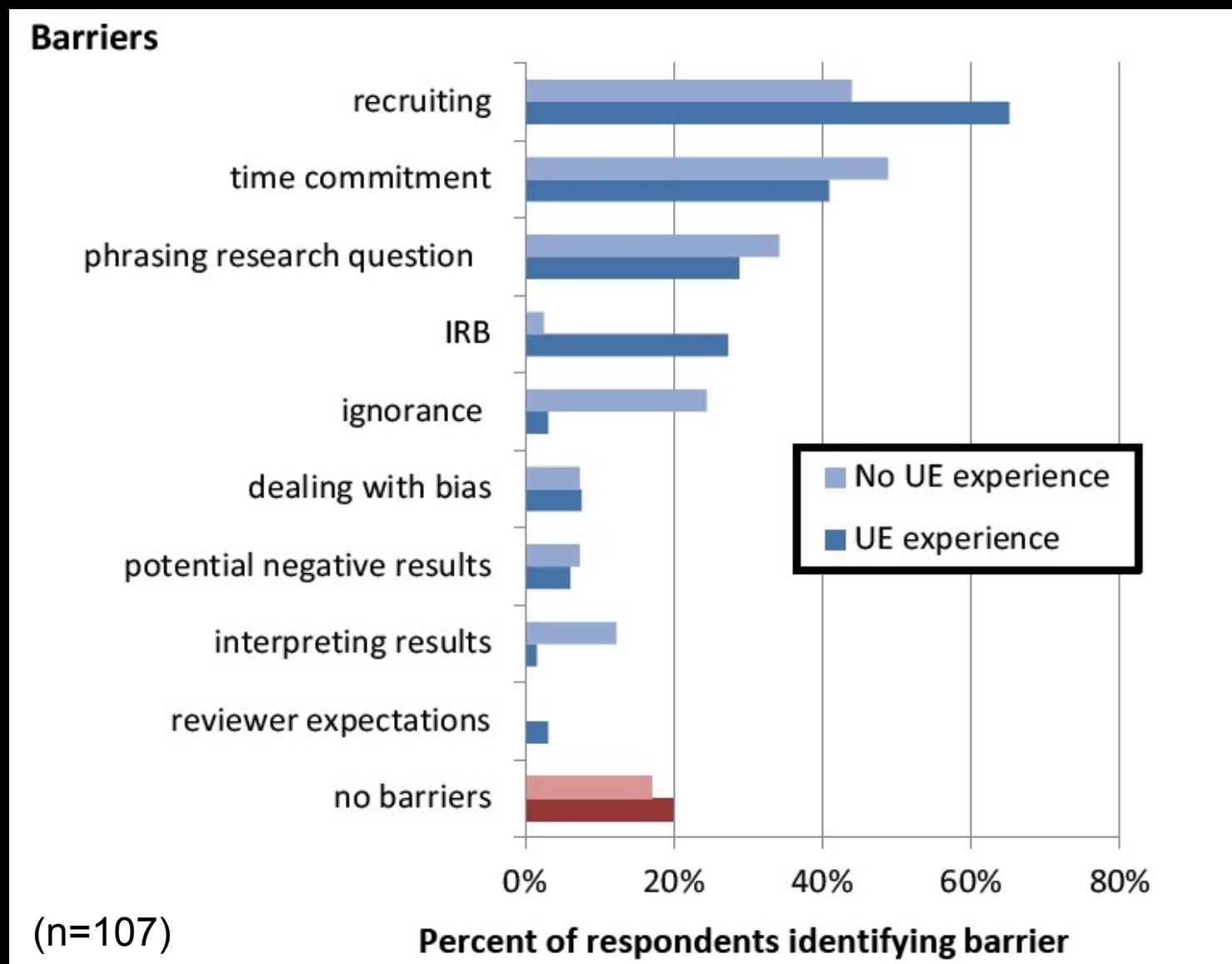
Number of the 15 papers
presented at SSBSE 2012 that
included a **human study**:

?

Number of the 15 papers
presented at SSBSE 2012 that
included a **human study**:

Zero.

Why Not Have a User Evaluation?



Hope

- Is an automated repair of high quality?
 - [Kim, Nam, Song, Kim: Automatic patch generation learned from human-written patches. ICSE 2013.]
- From 2000-2010, the number of human studies grew 500% at top SE conferences [Buse, Sadowski, Weimer: Benefits and barriers of user evaluation in software engineering research. OOPSLA 2011.]
- Two new sources of participants are available
 - Massive Open Online Courses (MOOCs)
 - Amazon's Mechanical Turk (crowdsourcing market)

One Source: MOOCs

- Popular: Udacity, Coursera, edX, ...
- Laurie Williams, Alex Orso, Andreas Zeller, Westley Weimer, Alex Aiken, John Regehr, ...
- **Simple:** course is unrelated
 - I asked my MOOC students to participate in a human study and received 5,000+ responses (over 1,000 of which had 5+ years in industry) for \$0
- **Complex:** course uses your new tool
 - [Fast, Lee, Aiken, Koller, Smith. Crowd-scale Interactive Formal Reasoning and Analytics. UIST 2013.]

One Source: Mechanical Turk

All HITs
1-10 of 2342 Results

Sort by: HITs Available (most first) Show all details | Hide all details 1 2 3 4 5 > Next >> Last

Categorization		View a HIT in this group	
Requester:	Hillary Roulette	HIT Expiration Date:	Aug 18, 2013 (3 days 6 hours) Reward: \$0.02
		Time Allotted:	60 minutes HITs Available: 63839
Search: Keywords on Google.com (US)			
Requester:	CrowdSource	HIT Expiration Date:	Aug 15, 2014 (52 weeks) Reward: \$0.08
		Time Allotted:	16 minutes HITs Available: 14994
Extract purchased items from a shopping receipt			
Requester:	Jon Breig	HIT Expiration Date:	Aug 22, 2013 (6 days 23 hours) Reward: \$0.06
		Time Allotted:	2 hours HITs Available: 8234
Classify Arabic Tweets Dialects (No Qualification)			
Requester:	Chris Callison-Burch	HIT Expiration Date:	Aug 22, 2013 (6 days 21 hours) Reward: \$0.05
		Time Allotted:	60 minutes HITs Available: 7267
Basic Caption Requirements Review			
Requester:	Redwood	HIT Expiration Date:	Aug 15, 2014 (52 weeks) Reward: \$0.01
		Time Allotted:	15 minutes HITs Available: 5351

MTurk Has Programmers

[Evaluating Source Code, #8](#)

[View a HIT in this group](#)

Requester: [CS Researcher](#) **HIT Expiration Date:** Aug 15, 2013 (8 hours 53 minutes) **Reward:** \$0.20
Time Allotted: 60 minutes **HITs Available:** 7

Description: Given a programming task, determine if three Java source code snippets are relevant to the task.

Keywords: [Java](#), [programming](#), [source](#), [code](#), [study](#), [survey](#), [computer](#), [science](#), [quick](#)

Qualifications Required:

Java Knowledge Qualification is not less than 99

HIT approval rate (%) is not less than 90

[Fix a Java bug](#)

[View a HIT in this group](#)

Requester: [ipam hkust](#) **HIT Expiration Date:** Aug 20, 2013 (5 days 1 hour) **Reward:** \$3.00
Time Allotted: 30 minutes **HITs Available:** 3

Description: Given a piece of buggy source code, find the bug and fix it.

Keywords: [debug](#), [Java](#), [programming](#)

Qualifications Required:

Java Programming has been granted

[Evaluating Source Code, #7](#)

[View a HIT in this group](#)

Using MTurk

- Register, link your credit card, say you have \$100 for HITs (Human Intelligence Tasks)
- Write a little boilerplate text:

Note: Only 1 HIT will be accepted per worker for this group (Problem 8).

Do the following:

1. Consider the programming task and compose an example input and expected output. For example, if the task is to compute the square of an integer, the input and output could be 3 and 9, respectively. (*Accuracy is required for payment*)
2. For each of the three Java code snippets: (*Completion of all parts is required for payment*)
 3. Read the Java code and determine if it is *relevant* to the programming task. Justify your response.
 4. Determine if the source code *solves* the programming task. Justify your response.

Programming Task: Given the String representation of a file name, trim off the extension

1. Provide input and expected output for the programming task (if the input requires multiple pieces,

1, 2, 3, 4)

Using MTurk (2)

- Make a simple webpage that records user selections or responses
- Include a survey at the end, and print out a randomly generated completion code
- Amazon workers use the code when asking for the money: you only give money to **accurate workers!**

Snippet

```
registry['include-output'] = ('EffDirectives', 'include_output')
registry['comment'] = ('EffDirectives', 'comment')

en.directives['chapter-list'] = 'chapter-list'
en.directives['item-list'] = 'item-list'
en.directives['index'] = 'index'
en.directives['include-code'] = 'include-code'
en.directives['include-output'] = 'include-output'
en.directives['comment'] = 'comment'
```

History

▼ Snippet 1

1 2 3 4 5

Zeno's Paradox

- Many MTurk workers will try to **game the system.**
 - 100 participants → 50 are usable
- However, the average fill time for 100 30-minute CS tasks at \$2 each is **only a few hours.**
- [Kittur, Chi, Suh. Crowdsourcing user studies with Mechanical Turk. CHI, 2008.]
- [Snow, O'Connor, Jurafsky, Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. EMNLP, 2008.]

Conclusion

- Industry is already paying untrusted strangers
- **Automated Program Repair** is a hot research area with rapid growth in the last few years
 - (Lesson: integrating with existing tests is hard.)
- Challenges & Opportunities:
 - **Test Suites and Oracles** (spec mining)
 - **Benchmarking** (reproducible)
 - **Human Studies** (crowdsourcing)