Impact-Driven Research on Software Testing and Analysis Tools: *Getting Real*

Tao Xie

University of Illinois
at Urbana-Champaign

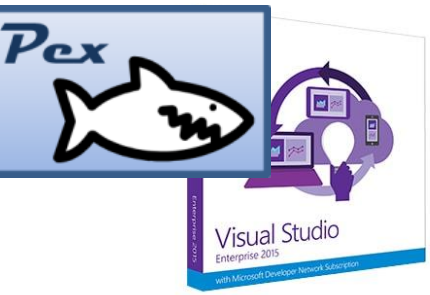
taoxie@illinois.edu

Successful Samples: Research ➔ Practice

pattern insight™

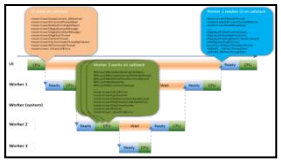
coverity™ a higher code™

SLAM



XIAO

MSRA

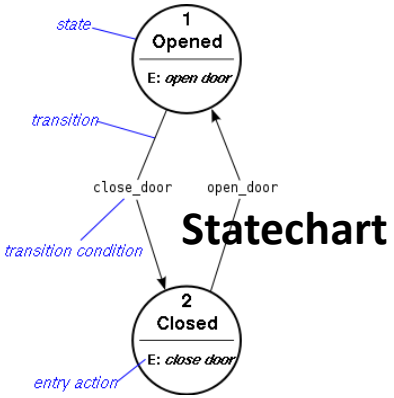


STACKMINE

MSRA

Google

Tricorder/Shipshape



Statechart

SPIN

…

CAV Awards

- 2016: Separation logic (Facebook Infer)
- 2013: UPPAAL
- 2012: PVS
- 2011: SLAM/SDV
- 2010: Cadence SMV
- …

http://i-cav.org/cav-award

https://www.research.ibm.com/haifa/conferences/hvc2016/award.shtml

ACM Software System Awards

- 2013: Coq
- 2012: LLVM
- 2007: Statemate
- 2006: Eiffel
- 2005: Boyer-Moore Theorem Prover
- 2001: SPIN
- …

http://awards.acm.org/software_system/year.cfm

NSF Workshop on Formal Methods

December 2012

- Goal: to identify the future directions in research in formal methods and its transition to industrial practice.
- The workshop brought together researchers and identified primary challenges in the field, both foundational, infrastructural, and in transitioning ideas from research labs to developer tools.

http://goto.ucsd.edu/~rjhala/NSFWorkshop/

NSF Workshop on Formal Methods

December 2012

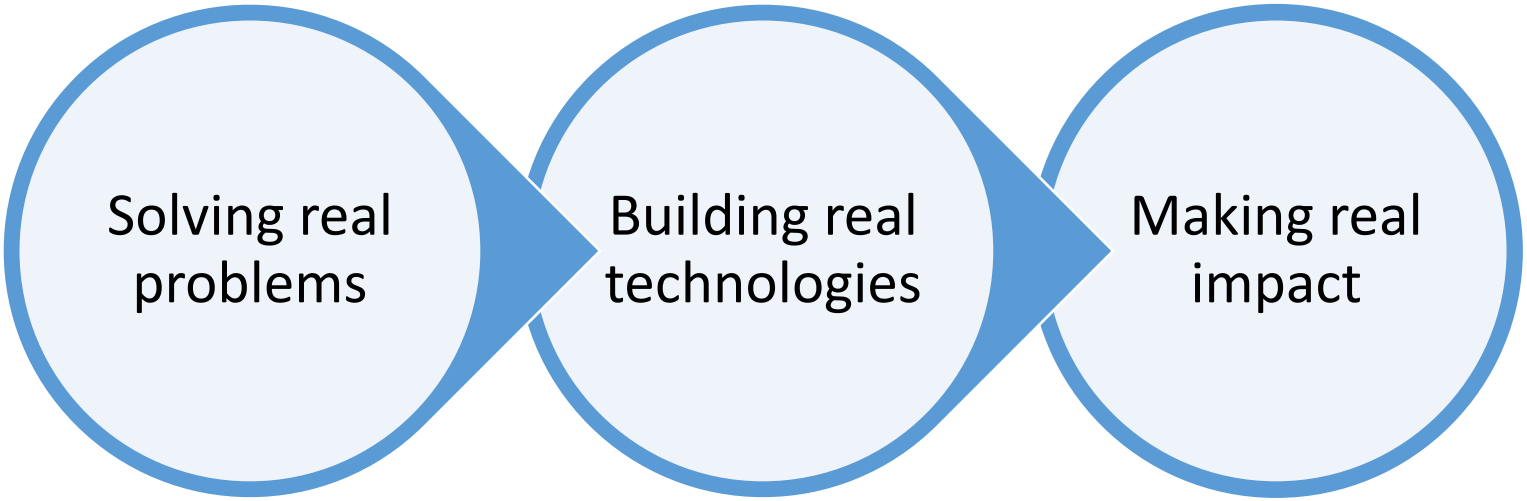
- Successes
- Unrealized Successes
- Obstacles
- Future Directions

http://goto.ucsd.edu/~rjhala/NSFWorkshop/static/Questions.md

http://goto.ucsd.edu/~rjhala/NSFWorkshop/static/survey.pdf

Getting Real to Produce Practice Impact

Software testing/analysis tools are naturally tied with software development practice



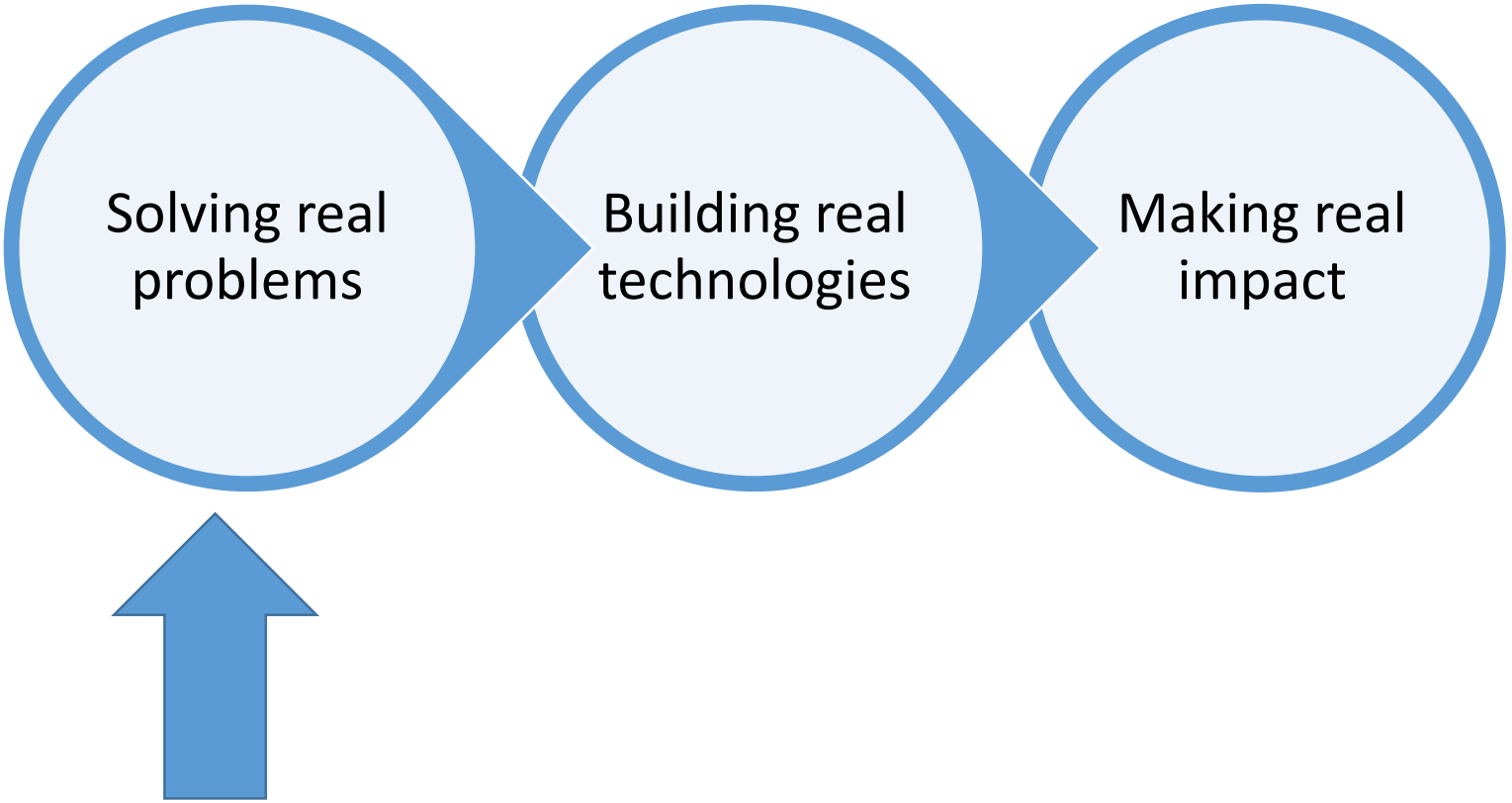
Industry Academia Collaboration

- Academia (research recognitions, e.g., papers) vs. Industry (company revenues)
- Academia (invention/innovation) vs. Industry (likely involving engineering efforts)
- Academia (long-term/fundamental research or out of box thinking) vs. Industry (short-term research or work)

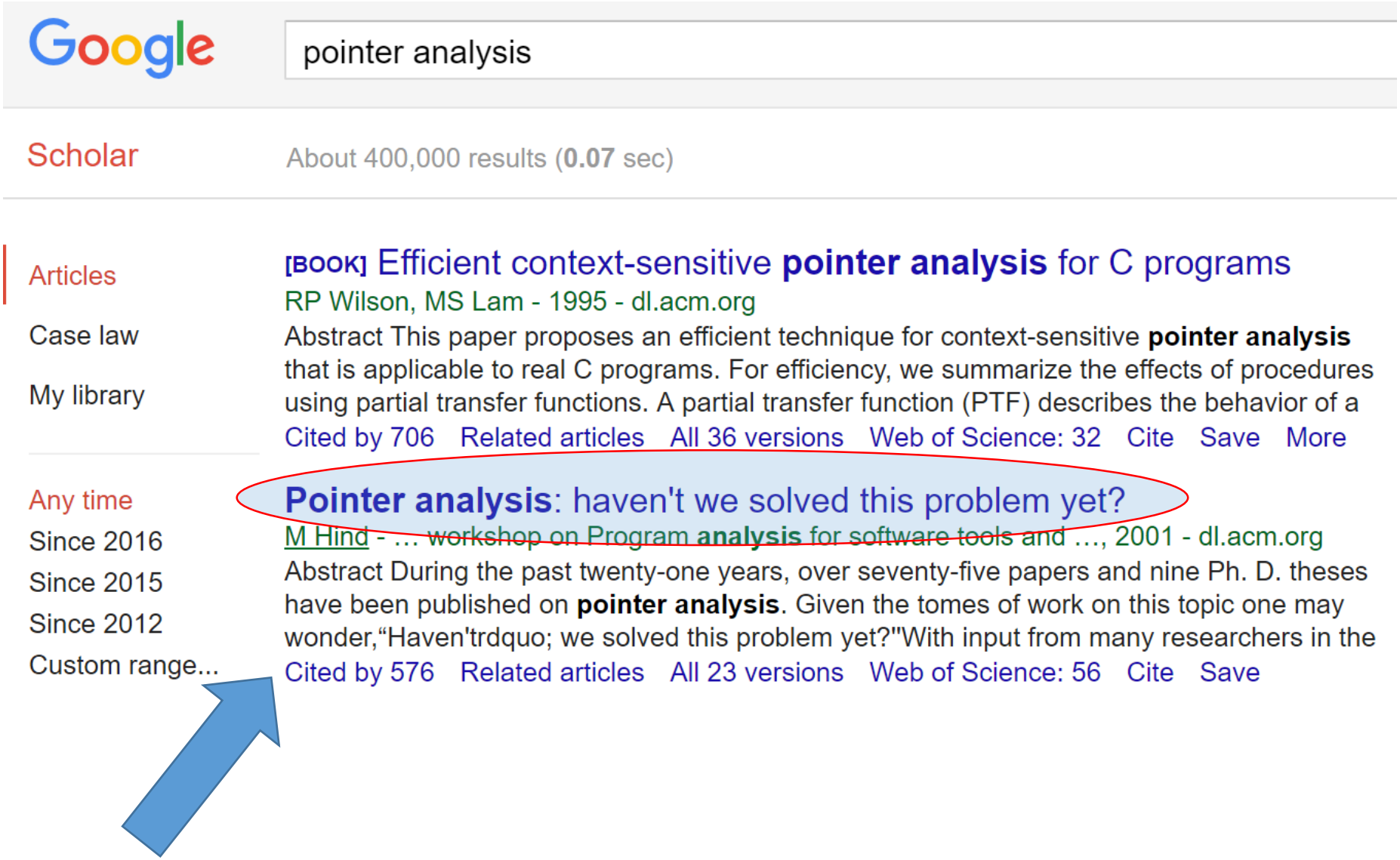
- Industry: problems, infrastructures, data, evaluation testbeds, …
- Academia: educating students, …

Getting Real to Produce Practice Impact

Software testing/analysis tools are naturally tied with software development practice



Google Scholar: “Pointer Analysis”



“Pointer Analysis: Haven’t We Solved This Problem Yet?” [Hind PASTE’01]

“During the past 21 years, over 75 papers and 9 Ph.D. theses have been published on pointer analysis. Given the tones of work on this topic one may wonder, “Haven't we solved this problem yet?'' With input from many researchers in the field, this paper describes issues related to pointer analysis and remaining open problems.”

Michael Hind. **Pointer analysis: haven't we solved this problem yet?.** In *Proc. ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering* (PASTE 2001)

Source©M. Hind

“Pointer Analysis: Haven’t We Solved This Problem Yet?” [Hind PASTE’01]

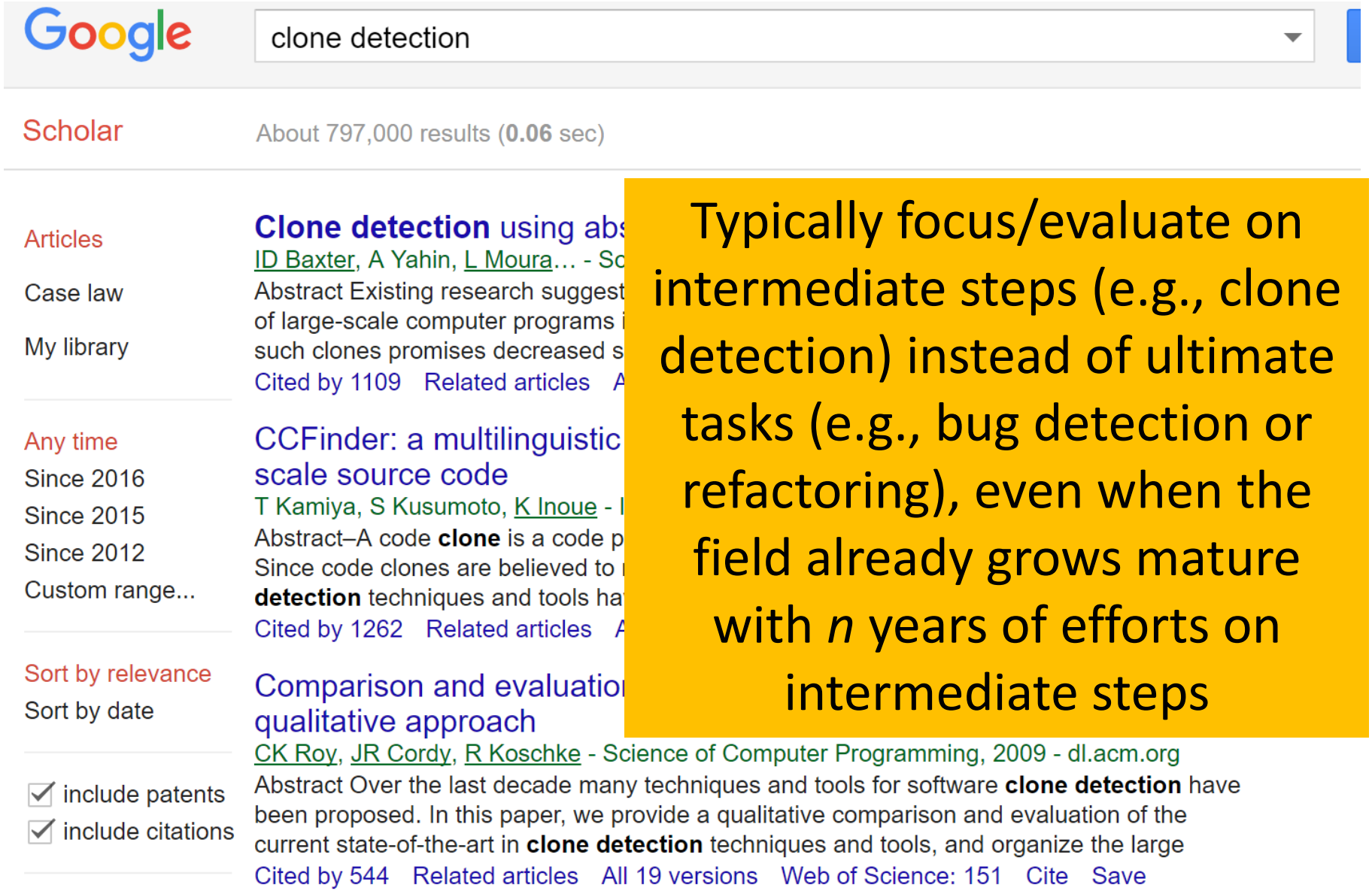
Section 4.3 Designing an Analysis for a Client’s Needs

“Barbara Ryder expands on this topic: “… We can all write an unbounded number of papers that compare different pointer analysis approximations in the abstract. However, this does not accomplish the key goal, which is to design and engineer pointer analyses that are **useful for solving real software problems for realistic programs**.”

Michael Hind. **Pointer analysis: haven't we solved this problem yet?.** In *Proc. ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering* (PASTE 2001)

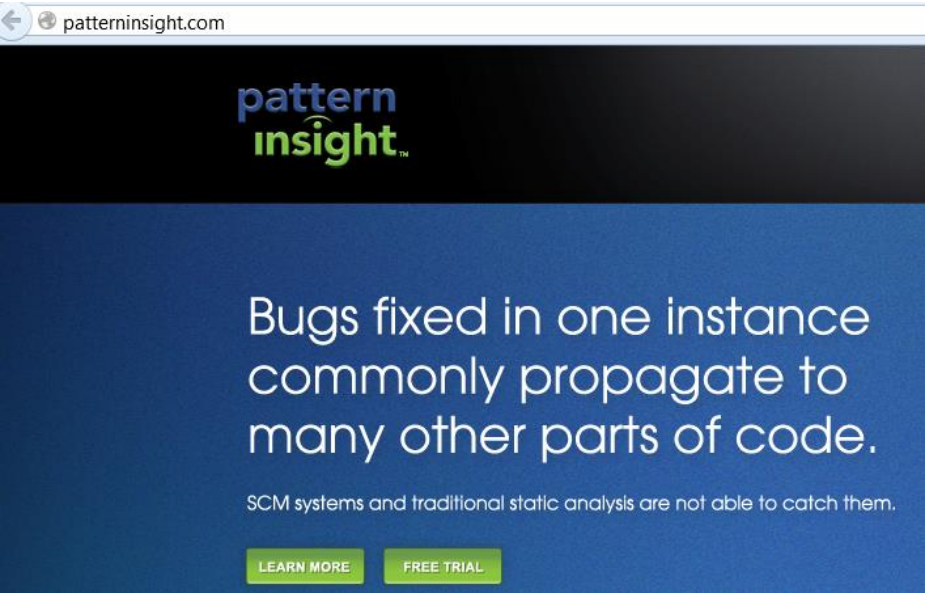
Source©M. Hind&B. Ryder

Google Scholar: “Clone Detection”



Typically focus/evaluate on intermediate steps (e.g., clone detection) instead of ultimate tasks (e.g., bug detection or refactoring), even when the field already grows mature with *n* years of efforts on intermediate steps

Some Success Stories of Applying Clone Detection [**Focus** on **Ultimate** Tasks]



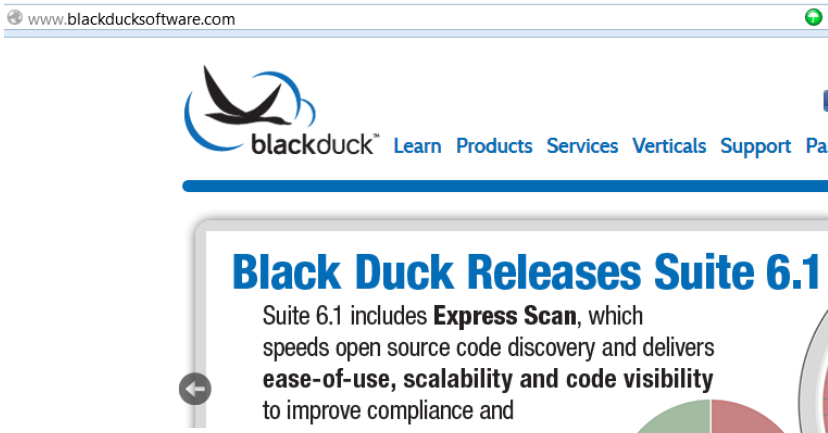
Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. **CP-Miner: a tool for finding copy-paste and related bugs in operating system code**. In *Proc. OSDI 2004.*

http://patterninsight.com/

MSRA XIAO

Yingnong Dang, Dongmei Zhang, Song Ge, Chengyun Chu, Yingjun Qiu, and Tao Xie. **XIAO: Tuning Code Clones at Hands of Engineers in Practice**. In *Proc. ACSAC 2012*,

http://research.microsoft.com/en-us/groups/sa/



http://www.blackducksoftware.com/

XIAO Usage: MS security bulletin MS12-034

Combined security update for Microsoft Office, Windows, .NET Framework, and Silverlight, published: Tuesday, May 08, 2012

3 publicly disclosed vulnerabilities and seven privately reported involved. Specifically, one is exploited by the Duqu malware to execute arbitrary code when a user opened a malicious Office document.

Insufficient bounds check within the font parsing subsystem of win32k.sys

Cloned copy in gdiplus.dll, ogl.dll (office), Silverlight, and Windows Journal viewer

Microsoft Security Research & Defense Blog about this bulletin

"However, we wanted to be sure to address the vulnerable code wherever it appeared across the Microsoft code base. To that end, we have been working with Microsoft Research to develop a "Cloned Code Detection" system that we can run for every MSRC case to find any instance of the vulnerable code in any shipping product. This system is the one that found several of the copies of CVE-2011-3402 that we are now addressing with MS12-034."

http://blogs.technet.com/b/srd/archive/2012/05/08/ms12-034-duqu-ten-cve-s-and-removing-keyboard-layout-file-attack-surface.aspx

Trying Real Tool – Parasoft Jtest 5.0

Back in Year 2004

```
Test 1 (T1):
 BST t1 =
    new BST();
 t1.insert(2);
 t1.insert(1);
 t1.remove(1);
 t1.insert(3);
 t1.size();
```

```
Test 2 (T2):
 BST t2 =
    new BST ();
 t2.insert(2);
 t2.insert(3);
```

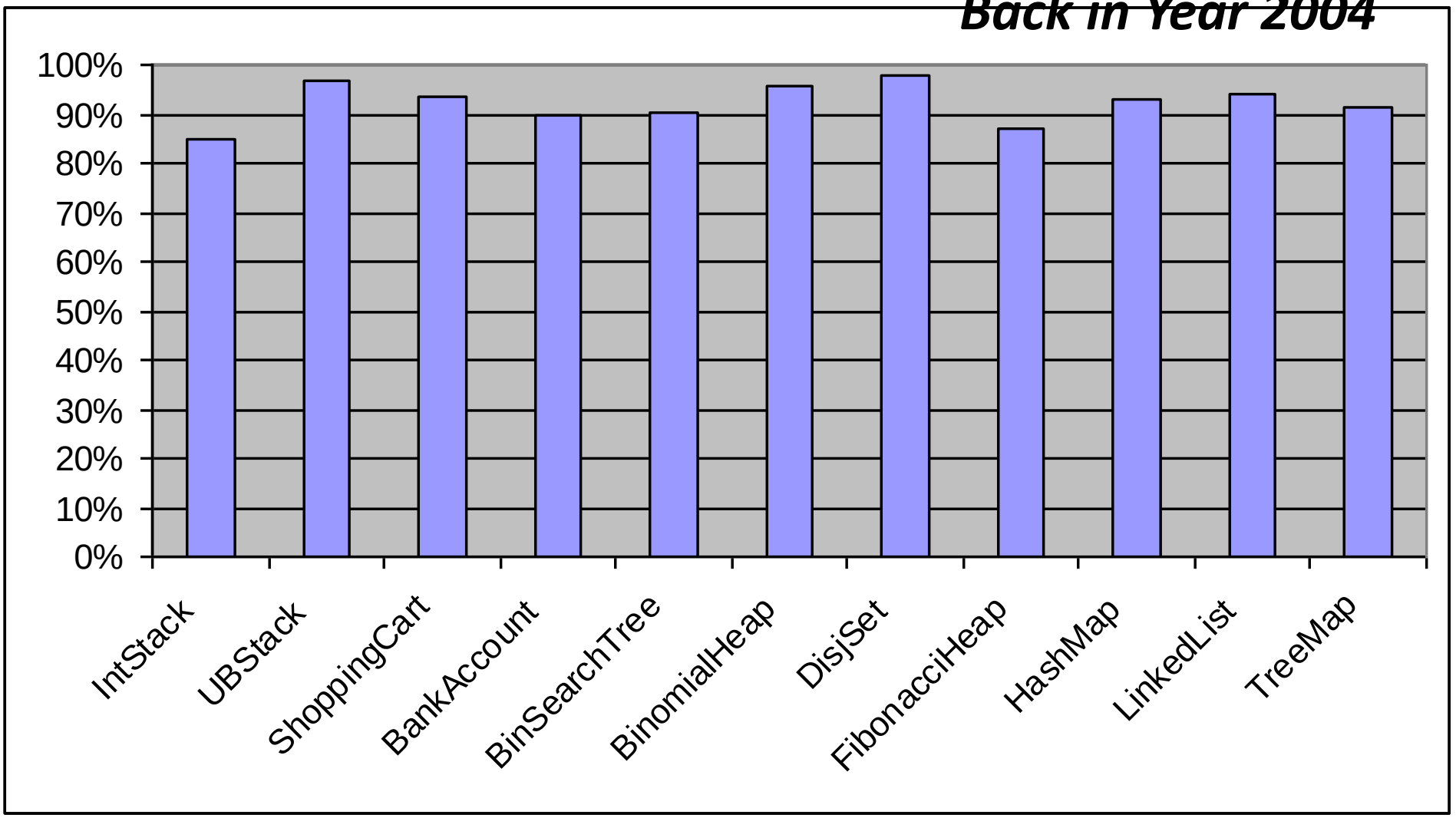
```
Test 3 (T3):
 BST t3 =
    new BST ();
 t3.insert(2);
 t3.insert(1);
 t3.size();
```

Approach using method sequence removes **no tests**

Rostra removes **T2 and T3** because T2 and T3 are redundant w.r.t. T1

Redundancy among Jtest-Generated Tests

Back in Year 2004



- 90% of generated tests are redundant!
- Minimized tests preserve the same code (branch) coverage and seeded-bug coverage

Industry Impact — Parasoft Jtest

Back in Year 2004

- People do use Jtest
  - Recognized with numerous awards, including Jolt Product Excellence Award and JDJ Editor's Choice Award in 2004; adopted by thousands of development teams worldwide.

    — businesswire.com

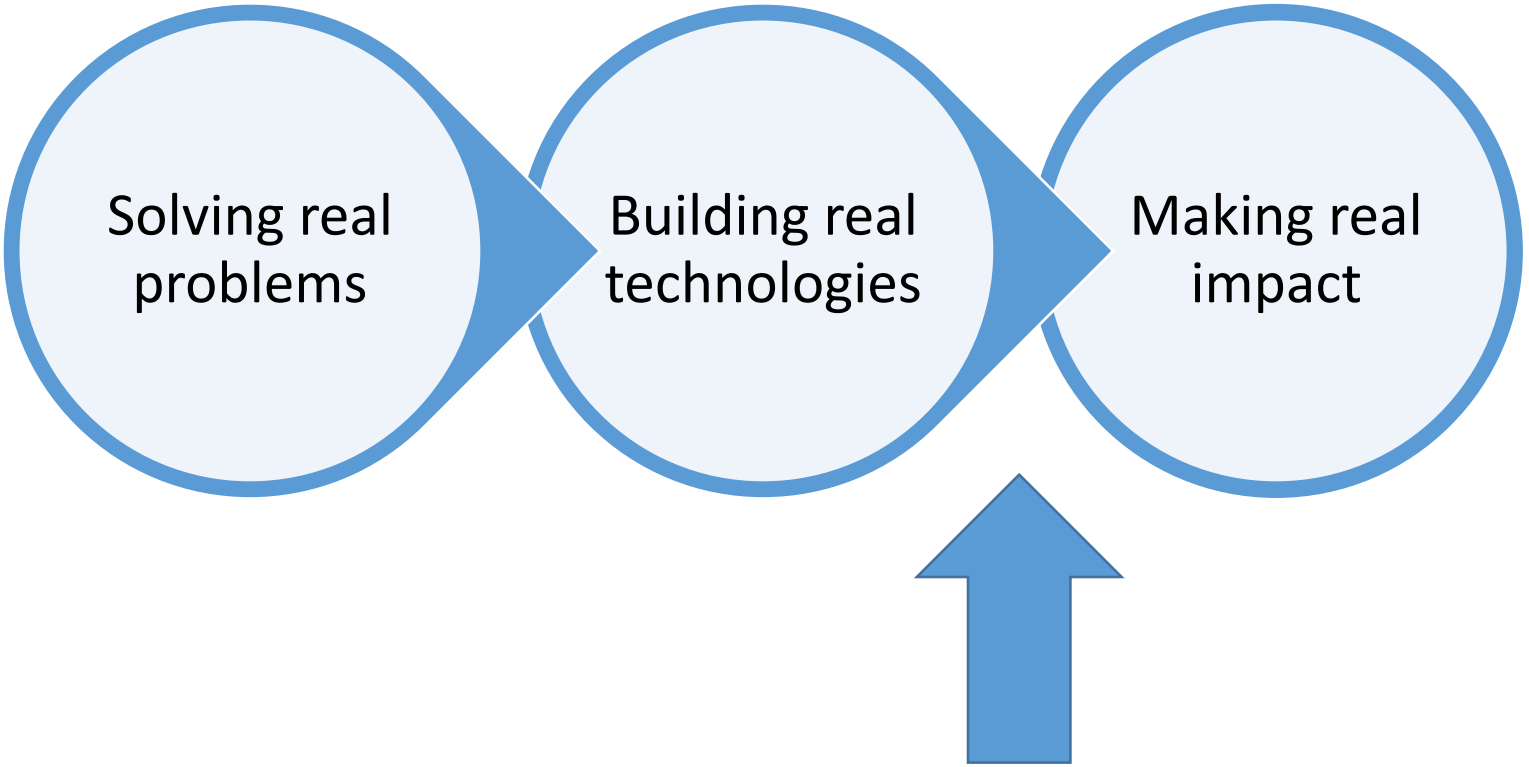
- But didn't seem to love its test generation
  - "I can't think of anyone telling me that they love Jtest's test-generation feature."

    —Jo[illegible]er, JUnit book author, 02/05@junit user mailing list

Parasoft VP contacted me informing me that they incorporated our new techniques into Jtest's later versions!

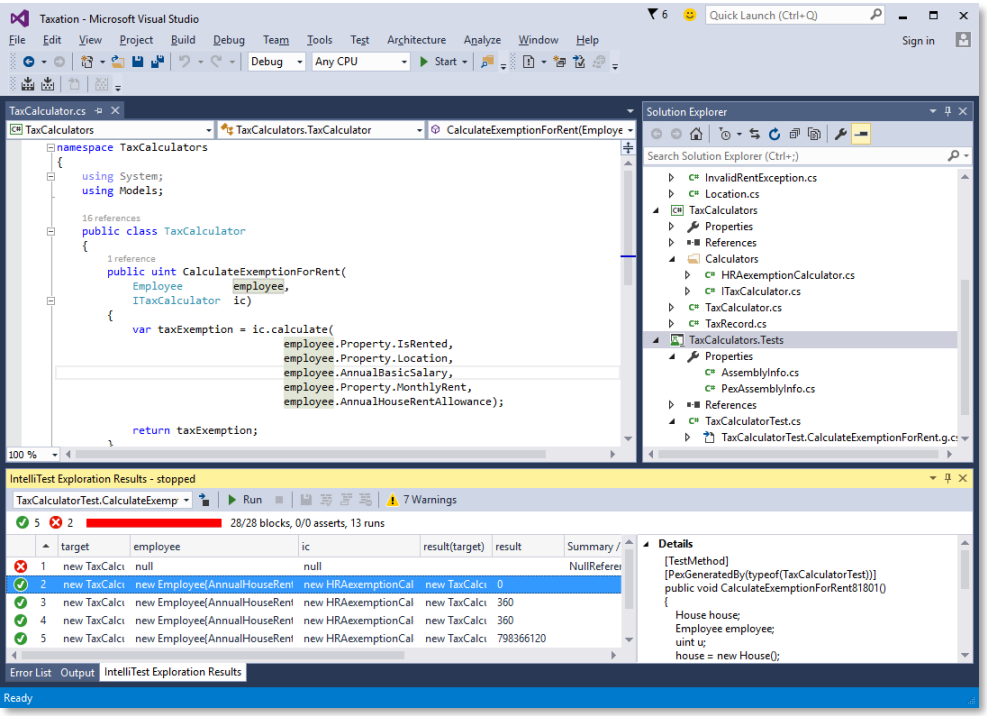
Getting Real to Produce Practice Impact

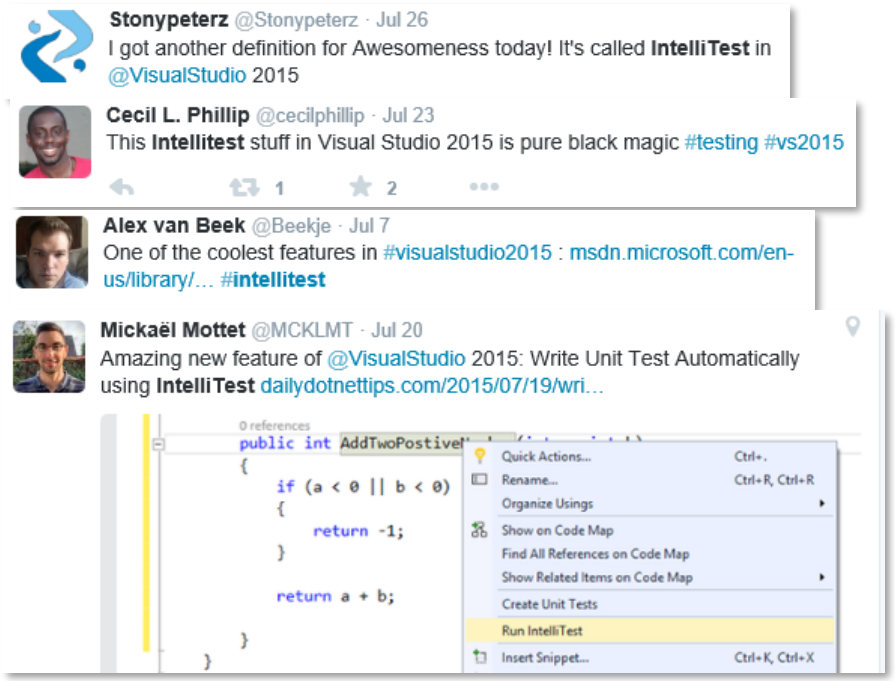
Software testing/analysis tools are naturally tied with software development practice



Pex to IntelliTest

Shipped in Visual Studio 2015 Enterprise Edition!





[NUnit Extension](#) – 13024 downloads

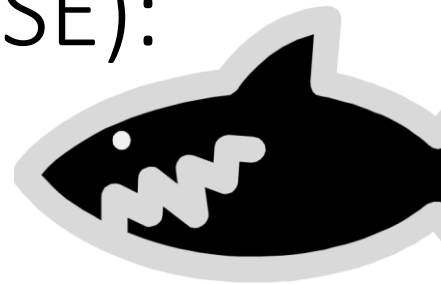
[xUnit.net Extension](#) - 8589 downloads

From http://bbcode.codeplex.com/

Why is it as stable as we claim?

We have used Visual Studio IntelliTest (Announcement for Visual Studio 2015: https://www.visualstudio.com/en-us/products/vs-2015-product-editions.aspx) to extensively test some important properties of this BBCode-Parser. We used IntelliTest to ensure that the parser never crashes and that it never emits any dangerous tag such as

(Dynamic) Symbolic Execution (DSE): Explosion of Search Space

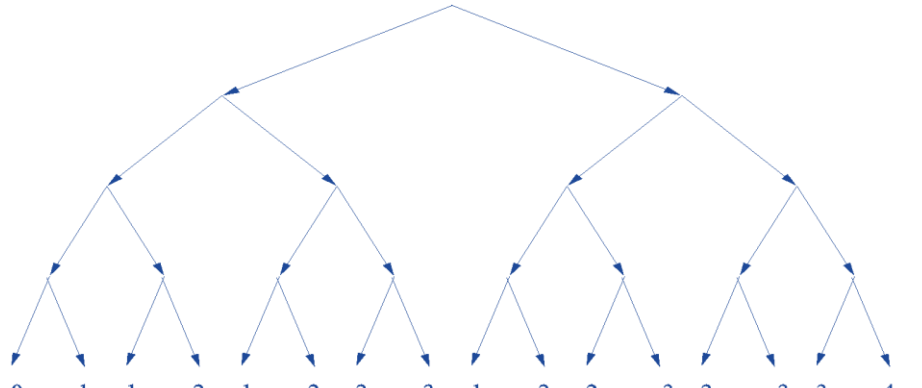


There are decision procedures for individual path conditions, but…

- Number of potential paths grows exponentially with number of branches
- Reachable code not known initially
- Without guidance, same loop might be unfolded forever

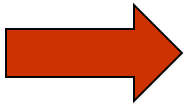
Fitnex search strategy

[Xie et al. DSN 09]
http://taoxie.cs.illinois.edu/publications/dsn09-fitnex.pdf



DSE Example

```
public bool TestLoop(int x, int[] y) {
  if (x == 90) {
      for (int i = 0; i < y.Length; i++)
            if (y[i] == 15)
                  x++;
      if (x == 110)
            return true;
    }
   return false;
}
```

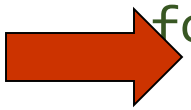


TestLoop(0, {0})

Path condition:
!(x == 90)
↓
New path condition:
(x == 90)
↓
New test input:
TestLoop(90, {0})

DSE Example

```
public bool TestLoop(int x, int[] y) {
  if (x == 90) {
      for (int i = 0; i < y.Length; i++)
          if (y[i] == 15)
              x++;
      if (x == 110)
          return true;
    }
   return false;
}
```



TestLoop(90, {0})

Path condition:
(x == 90) && !(y[0] ==15)
↓
New path condition:
(x == 90) && (y[0] ==15)
↓
New test input:
TestLoop(90, {15})

Challenge in DSE

```
public bool TestLoop(int x, int[] y) {
  if (x == 90) {
        for (int i = 0; i < y.Length; i++)
              if (y[i] == 15)
                    x++;
        if (x == 110)
              return true;
    }
   return false;
}
```

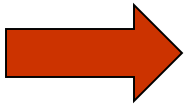
TestLoop(90, {15})

Path condition:
(x == 90) && (y[0] ==15)
&& !(x+1 == 110)
↓
New path condition:
(x == 90) && (y[0] ==15)
&& (x+1 == 110)
↓
New test input:
No solution!?

A Closer Look

```
public bool TestLoop(int x, int[] y) {
  if (x == 90) {
        for (int i = 0; i < y.Length; i++)
              if (y[i] == 15)
                    x++;
        if (x == 110)
              return true;
    }
   return false;
}
```

TestLoop(90, {15})

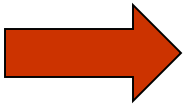


Path condition:
(x == 90) && (y[0] ==15)
&& (0 < y.Length)
&& !(1 < y.Length)
&& !(x+1 == 110)
↓
New path condition:
(x == 90) && (y[0] ==15)
&& (0 < y.Length)
&& (1 < y.Length)
➔ **Expand array size**

A Closer Look

```
public bool TestLoop(int x, int[] y) {
  if (x == 90) {
        for (int i = 0; i < y.Length; i++)
              if (y[i] == 15)
                    x++;
        if (x == 110)
              return true;
    }
    return false;
}
```

TestLoop(90, {15})



We can have infinite paths!

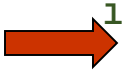
Manual analysis → need at least 20 loop iterations to cover the target branch

Exploring all paths up to 20 loop iterations is infeasible:

2^{20} paths

Fitnex: Fitness-Guided Exploration

```
public bool TestLoop(int x, int[] y) {
 if (x == 90) {
    for (int i = 0; i < y.Length; i++)
      if (y[i] == 15)
         x++;
    if (x == 110)
      return true;
 }
 return false;
}
```



```
TestLoop(90, {15, 0})
TestLoop(90, {15, 15})
```

Key observations: with respect to the coverag target

- not all paths are equally promising for branch-node flipping
- not all branch nodes are equally promising to flip

- Our solution:
  - Prefer to flip branch nodes on the most ***promising*** paths
  - Prefer to flip the most ***promising*** branch nodes on paths
  - Fitness function to measure “promising” extents

Fitness Function

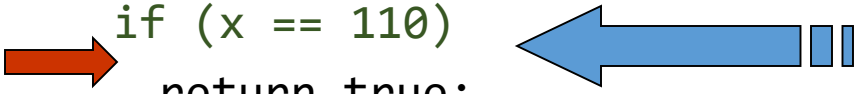
- FF computes fitness value (distance between the current state and the goal state)
- Search tries to minimize fitness value

Predicate	Fitness function	
	True	False
$F(a == b)$	0	$\|a - b\|$
$F(a > b)$	0	$(b - a) + K$
$F(a >= b)$	0	$(b - a)$
$F(a < b)$	0	$(a - b) + K$
$F(a <= b)$	0	$(a - b)$
$F(P_1 \;\&\&\; P_2)$	0	$\mathrm{F}(P_1) + \mathrm{F}(P_2)$
$F(P_1 \;\|\|\; P_2)$	0	$(\mathrm{F}(P_1) * \mathrm{F}(P_2))/(\mathrm{F}(P_1) + \mathrm{F}(P_2))$

[Tracey et al. 98, Liu at al. 05, ...]

Fitness Function for (x == 110)

```
public bool TestLoop(int x, int[] y) {
 if (x == 90) {
    for (int i = 0; i < y.Length; i++)
      if (y[i] == 15)
         x++;
    if (x == 110)
      return true;
 }
 return false;
}
```



Fitness function: |110 – x |

Compute Fitness Values for Paths

```
public bool TestLoop(int x, int[] y) {
 if (x == 90) {
    for (int i = 0; i < y.Length; i++)
      if (y[i] == 15)
         x++;
    if (x == 110)
      return true;
 }
 return false;
}
```

Fitness function: |110 – x |

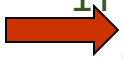
(x, y)	Fitness Value
(90, {0})	20
(90, {15})	19
(90, {15, 0})	19
(90, {15, 15})	18
(90, {15, 15, 0})	18
(90, {15, 15, 15})	17
(90, {15, 15, 15, 0})	17
(90, {15, 15, 15, 15})	16
(90, {15, 15, 15, 15, 0})	16
(90, {15, 15, 15, 15, 15})	15
…	

Give preference to flip paths with better fitness values

We still need to address which branch node to flip on paths …

Compute Fitness Gains for Branches

```
public bool TestLoop(int x, int[] y) {
 if (x == 90) {
    for (int i = 0; i < y.Length; i++)
      if (y[i] == 15)
         x++;
    if (x == 110)
      return true;
 }
 return false;
}
```



Fitness function: |110 – x |

Branch b1: i < y.Length
Branch b2: i >= y.Length
Branch b3: y[i] == 15
Branch b4: y[i] != 15

(x, y)	**Fitness Value**
(90, {0})	20
(90, {15}) ← flip b4	19
(90, {15, 0}) ← flip b2	19
(90, {15, 15}) ← flip b4	18
(90, {15, 15, 0}) ← flip b2	18
(90, {15, 15, 15}) ← flip b4	17
(90, {15, 15, 15, 0}) ← flip b2	17
(90, {15, 15, 15, 15}) ← flip b4	16
(90, {15, 15, 15, 15, 0}) ← flip b2	16
(90, {15, 15, 15, 15, 15}) ← flip b4	15
…	

- Flipping Branch b4 (b3) gives us average 1 (-1) fitness gain (loss)
- Flipping branch b2 (b1) gives us average 0 fitness gain (loss)

Compute Fitness Gain for Branches cont.

- For a flipped node leading to *Fnew*, find out the old fitness value *Fold* before flipping
  - Assign Fitness Gain (*Fold* – *Fnew*) for the branch of the flipped node
  - Assign Fitness Gain (*Fnew* – *Fold*) for the other branch of the branch of the flipped node
- Compute the average fitness gain for each branch over time

Search Frontier

- Each branch node candidate for being flipped is prioritized based on its composite fitness value:
  - (Fitness value of node – Fitness gain of its branch)
- Select first the one with the best composite fitness value

To avoid local optimal or biases, the fitness-guided strategy is **integrated** with Pex's fairness search strategies

http://taoxie.cs.illinois.edu/publications/dsn09-fitnex.pdf

Automated Test Input Generation for Android: Are We Really There Yet in an Industrial Case?

Xia Zeng[1] **Dengfeng Li**[2] Wujie Zheng[1] Fan Xia[1] **Yuetang Deng**[1] Wing Lam[2] Wei Yang[2] Tao Xie[2]
[1]Tencent, Inc., China [2]University of Illinois at Urbana-Champaign, USA

FSE Industry 2016

http://taoxie.cs.illinois.edu/publications/fse16industry-wechat.pdf

Motivation

- Choudhary et al. [*ASE'15*]: Do we have good-enough tools to test Android apps?
  - Evaluated six research tools and Monkey
  - Monkey tool outperformed all six research tools

- Their study can be further extended
  - No **industrial-strength** Android app was studied
  - No demonstration on whether and how techniques can further improve Monkey under industrial settings

Challenges on Testing Industrial Mobile Apps

- Challenges for code coverage measurement:
  - Requiring app's source code
  - Industrial-strength Android app can cause 64K reference limit exception during instrumentation
- Challenges for applicability:
  - Scalability on testing apps with large codebases
  - OS compatibility of testing tool

WeChat Overview

- WeChat = WhatsApp+Facebook+Instagram+PayPal+Uber…
- 846 million monthly active users
- Daily number: dozens of billion messages sent, hundreds of million photos uploaded, hundreds of million payment transactions executed
- WeChat backend: 2K+ microservices running on 40k+ servers
  - 10M queries per second during Chinese New Year Eve
- Large codebase on WeChat Android

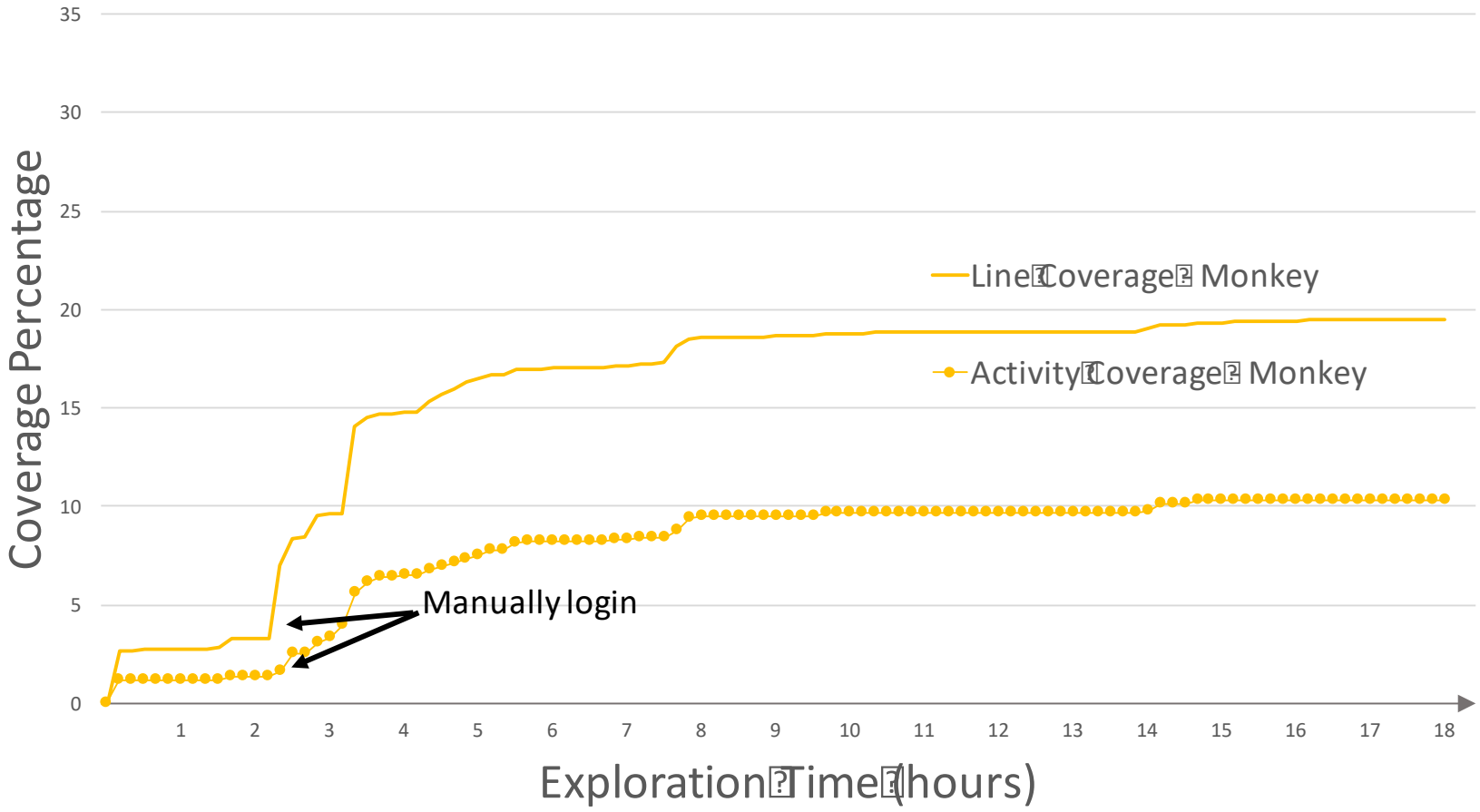
# of executable Java code lines:	610,629
# of Java classes:	8,425
# of Android activities:	607
# of C or C++ code lines:	~40,000

Monkey: Experimental setup

- Experiment Setup
  - Set Monkey to fire random events every 500 milliseconds
  - Run Monkey on WeChat 5 times independently
  - Run Monkey for 18 hours each time (2 hours without login)

- Evaluation Metric
  - Line coverage
  - Activity coverage

Monkey: Coverage Result Findings

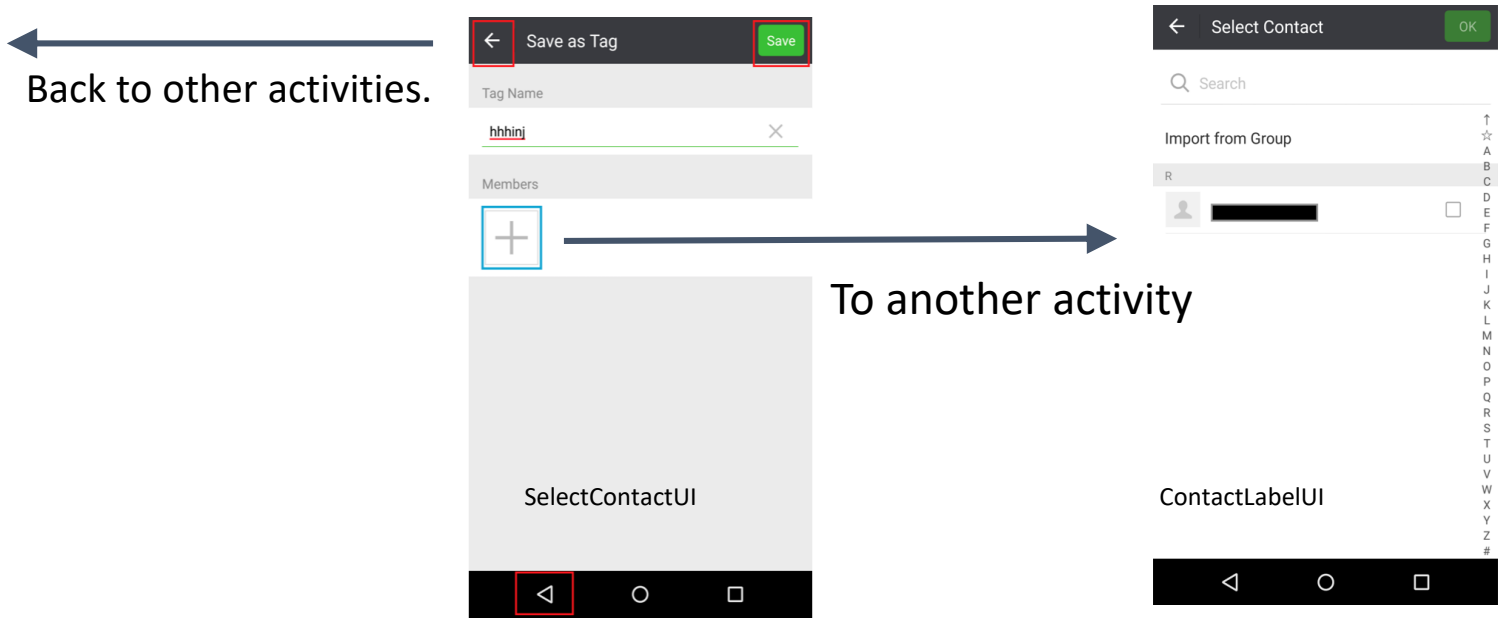


Finding 1: Monkey has low line coverage (**19.5%**) and low activity coverage (**10.3%**). s

Finding 2: Monkey allocates a lopsided distribution of exploration time on each activity.

Monkey: Exploration time challenges

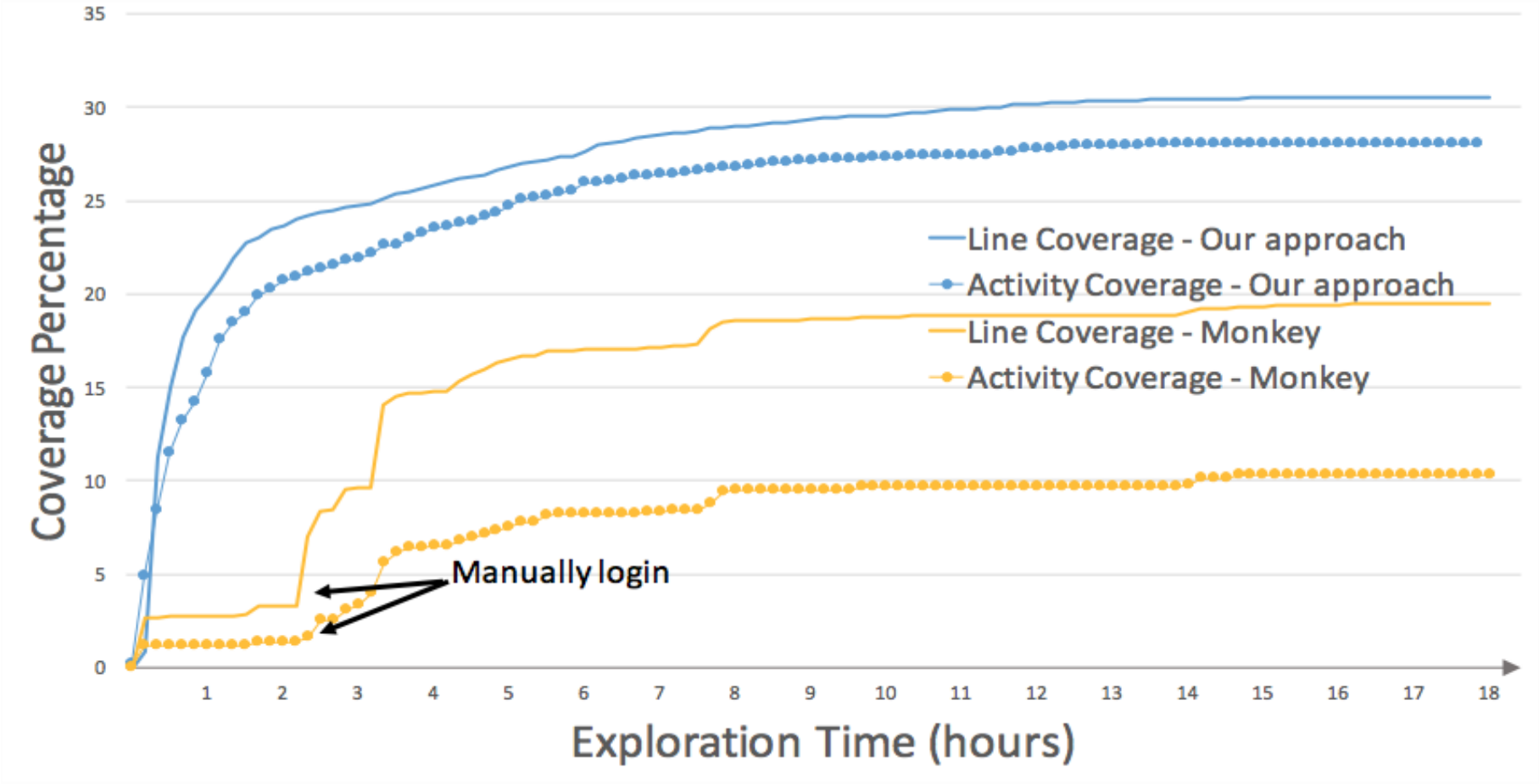
- **Widget obliviousness:** It is difficult to generate events at the small-sized GUI element
- **State obliviousness:** Monkey explores the same two activities repeatedly without contributing to new code coverage



New Approach

- Design goals
  - Have direct access to UI elements on activity under test
  - Allocate more exploration time towards new GUI states
- Techniques
  - Use UIAutomator framework to gain UI layout tree
  - Abstract GUI states to guide firing of state-changing events

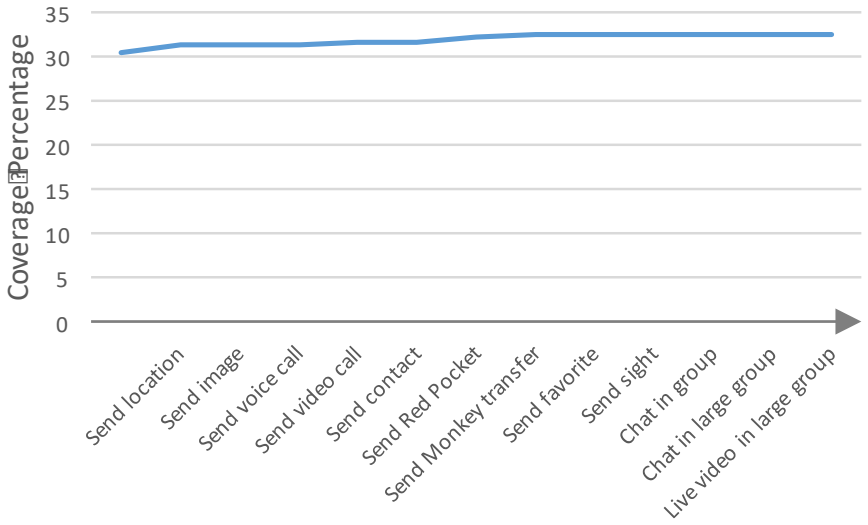
New Approach: Coverage result



New approach covers an additional **11.1% p.p.** more lines and **18.4% p.p.** more activities than Monkey does!

Human Intervention

Events	Condition
Scan valid QR code	Require a valid QR picture to be captured by the phone's camera.
Order taxi	Require valid address information.
Shake	Find friends through shaking the phone, which a tool cannot generate such shaking event.
Sign up	Require valid account information (*e.g.*, phone number or email address).
Inter-app communication	Require invoking other app to send an intent to WeChat.



- Provide client-to-client interaction
  - Line coverage increases 1.9% p.p.
  - Activity coverage increases 0.6% p.p..
- Provide additional app events
  - Line coverage increases 0.8% p.p.
  - Activity coverage increases 0.8% p.p.
- Human intervention doesn't help much

Ongoing/Future Work

- Incorporating fine-grained code analysis
  - Construct activity transition graph to guide the tools to explore not-covered activities
- Reusing exploration paths
  - Extracting relevant paths from explored paths and reusing them to boost tools' effectiveness
- Conducting residual coverage analysis
- Getting out of stuck states with manually written rules

Summary

- First industrial case study of applying Monkey on WeChat
- Empirical findings on Monkey's limitations in an industrial setting
- A new approach that addresses the major limitations of Monkey and accomplished substantial code-coverage improvements
- Empirical insights for future enhancements for both Monkey and our approach

Learning for Test Prioritization

An Industrial Case Study

Benjamin Busjaeger
Salesforce

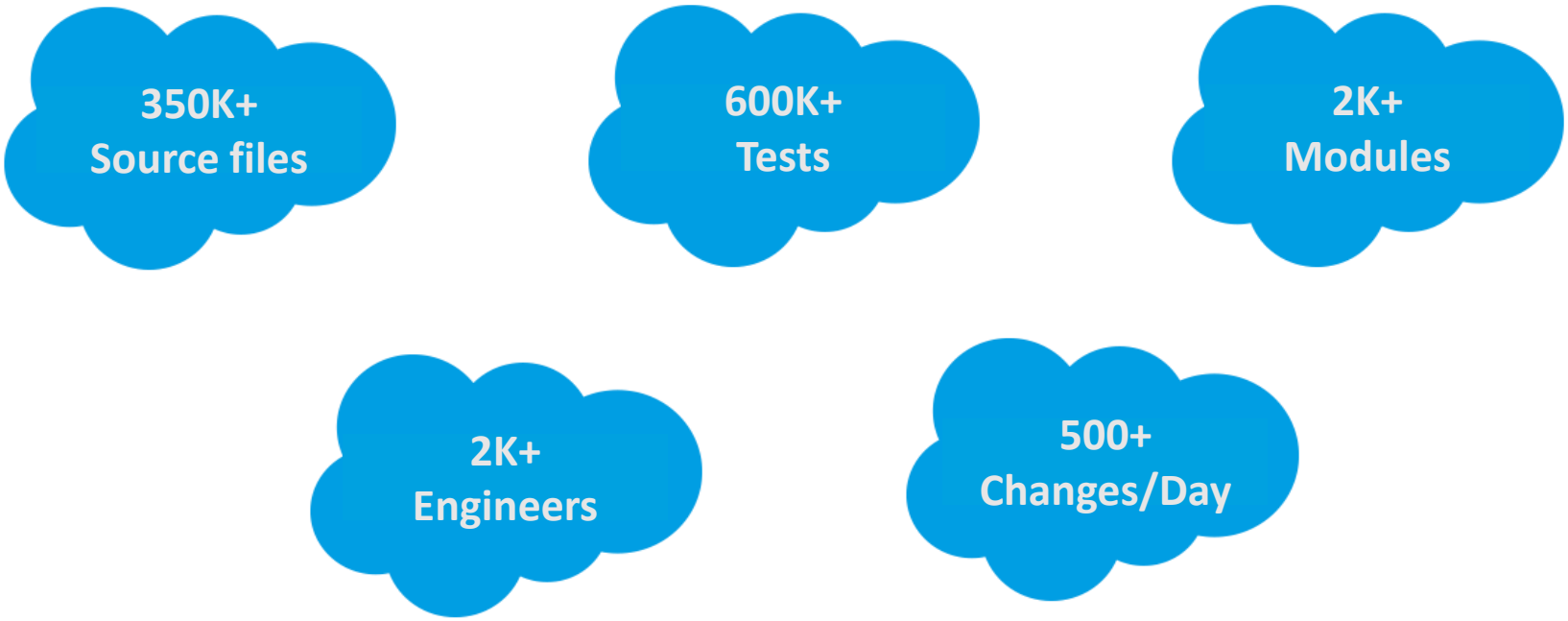
Tao Xie
University of Illinois,
Urbana-Champaign

FSE 2016 Industry Track

http://taoxie.cs.illinois.edu/publications/fse16industry-learning.pdf

Large Scale Continuous Integration

Main repository ...



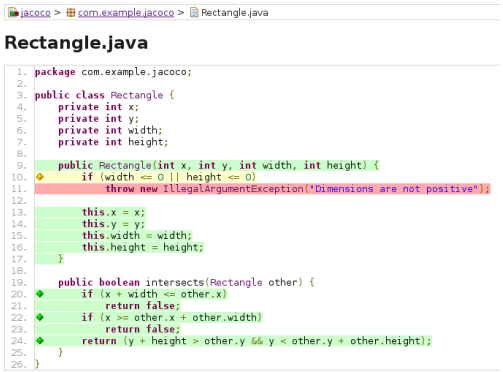
Motivation

	Before Commit	**After Commit**
Objective	Reject faulty changes	Detect test failures as quickly as possible
Current	Run hand-picked tests	Parallelize (8000 concurrent VMs) & batch (1-1000 changes per run)
Problem	Too complex for human	Feedback time constrained by capacity & batching complicates fault assignment
Desired	***Select*** top-k tests likely to fail	***Prioritize*** all tests by likelihood of failure

Insight: Need Multiple Existing Techniques

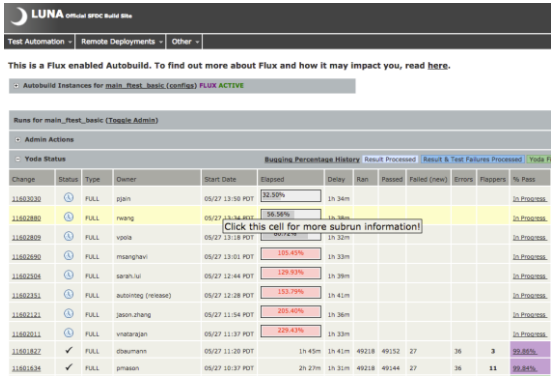
- Heterogeneous languages: Java, PL/SQL, JavaScript, Apex, etc.
- Non-code artifacts: metadata and configuration
- New/recently-failed tests more likely to fail

Test code coverage of change



Textual similarity between test and change

Test age and recent failures

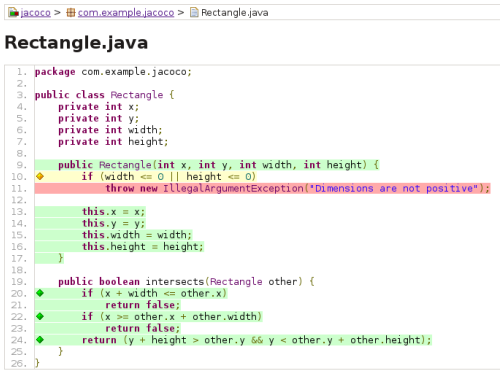


Insight: Need Scalable Techniques

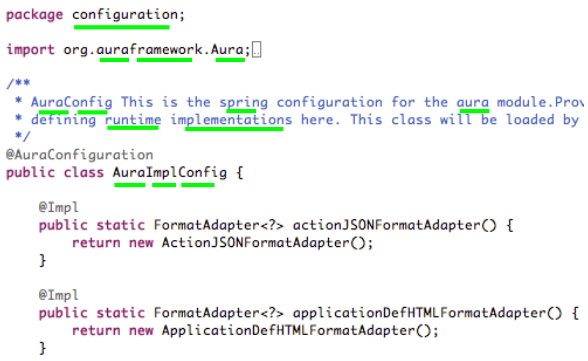
- Complex integration tests: concurrent execution
- Large data volume: 500+ changes, 50M+ test runs, 20TB+ results per day

→ Our approach: **Periodically** collect **coarse-grained** measurements

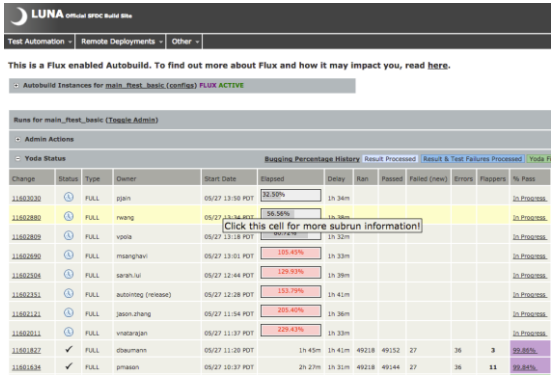
Test code coverage of change



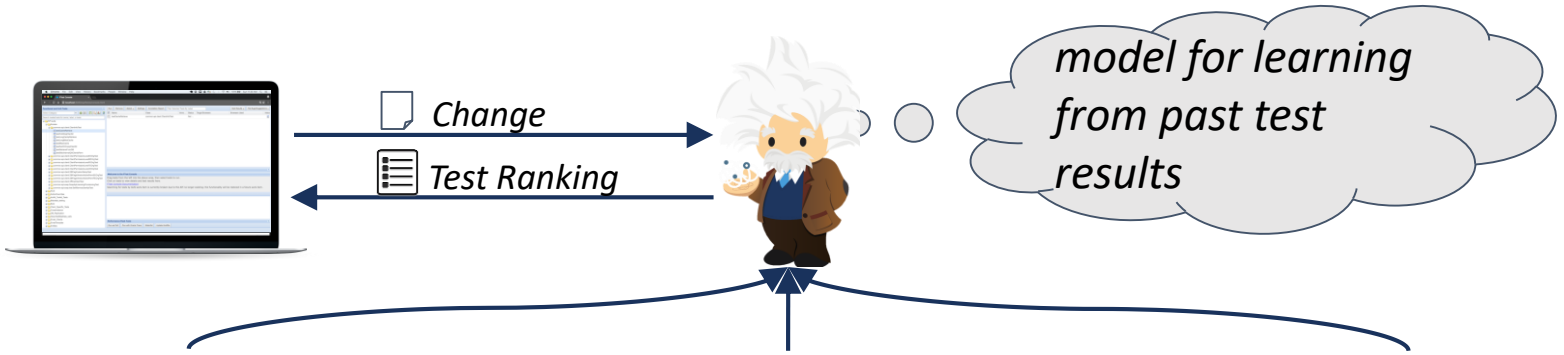
Textual similarity between test and change



Test age and recent failures



New Approach:
Learning for Test Prioritization



Test code coverage of change

jacoco > com.example.jacoco > Rectangle.java

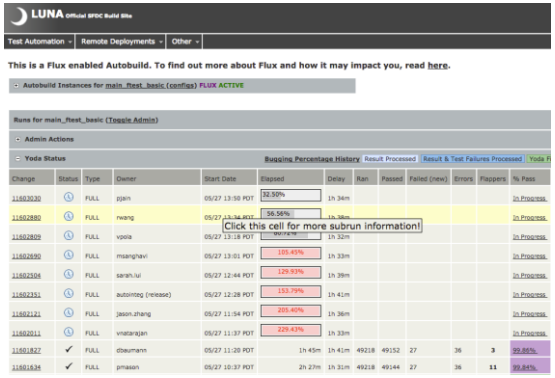
Rectangle.java

```
1.  package com.example.jacoco;
2.
3.  public class Rectangle {
4.      private int x;
5.      private int y;
6.      private int width;
7.      private int height;
8.
9.      public Rectangle(int x, int y, int width, int height) {
10.         if (width <= 0 || height <= 0)
11.             throw new IllegalArgumentException("Dimensions are not positive");
12.
13.         this.x = x;
14.         this.y = y;
15.         this.width = width;
16.         this.height = height;
17.     }
18.
19.     public boolean intersects(Rectangle other) {
20.         if (x + width <= other.x)
21.             return false;
22.         if (x >= other.x + other.width)
23.             return false;
24.         return (y + height > other.y && y < other.y + other.height);
25.     }
26. }
```

Textual similarity between test and change

```
package configuration;

import org.auraframework.Aura;

/**
 * AuraConfig This is the spring configuration for the aura module.Prov
 * defining runtime implementations here. This class will be loaded by
 */
@AuraConfiguration
public class AuraImplConfig {

    @Impl
    public static FormatAdapter<?> actionJSONFormatAdapter() {
        return new ActionJSONFormatAdapter();
    }

    @Impl
    public static FormatAdapter<?> applicationDefHTMLFormatAdapter() {
        return new ApplicationDefHTMLFormatAdapter();
    }
```

Test age and recent failures



→ **Implementation** currently in **pilot** use at Salesforce

salesforce

Empirical Study: Setup

- Test results of 45K tests for ~3 month period
- In this period, 711 changes with ≥1 failure
  - 440 for training
  - 271 for testing

- Collected **once** for each test:
  - Test code coverage
  - Test text content
- Collected **continuously**:
  - Test age and recent failures

Results: Test selection (before commit)

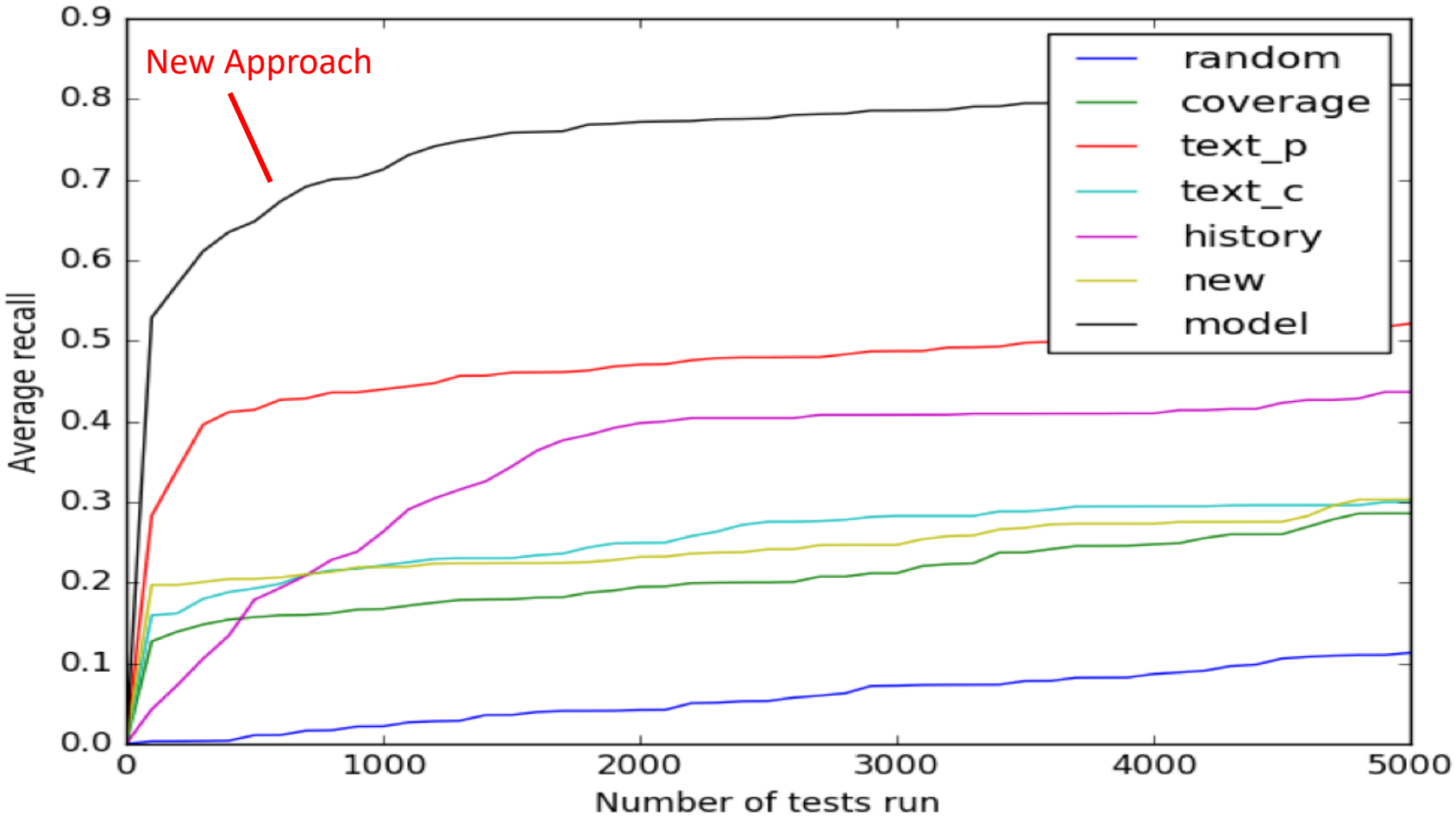


Figure 1: Average Recall

- New approach achieves highest average recall at all cutoff points
  - 50% failures detected with top 0.2%
  - 75% failures detected with top 3%

Results: Test prioritization (after commit)

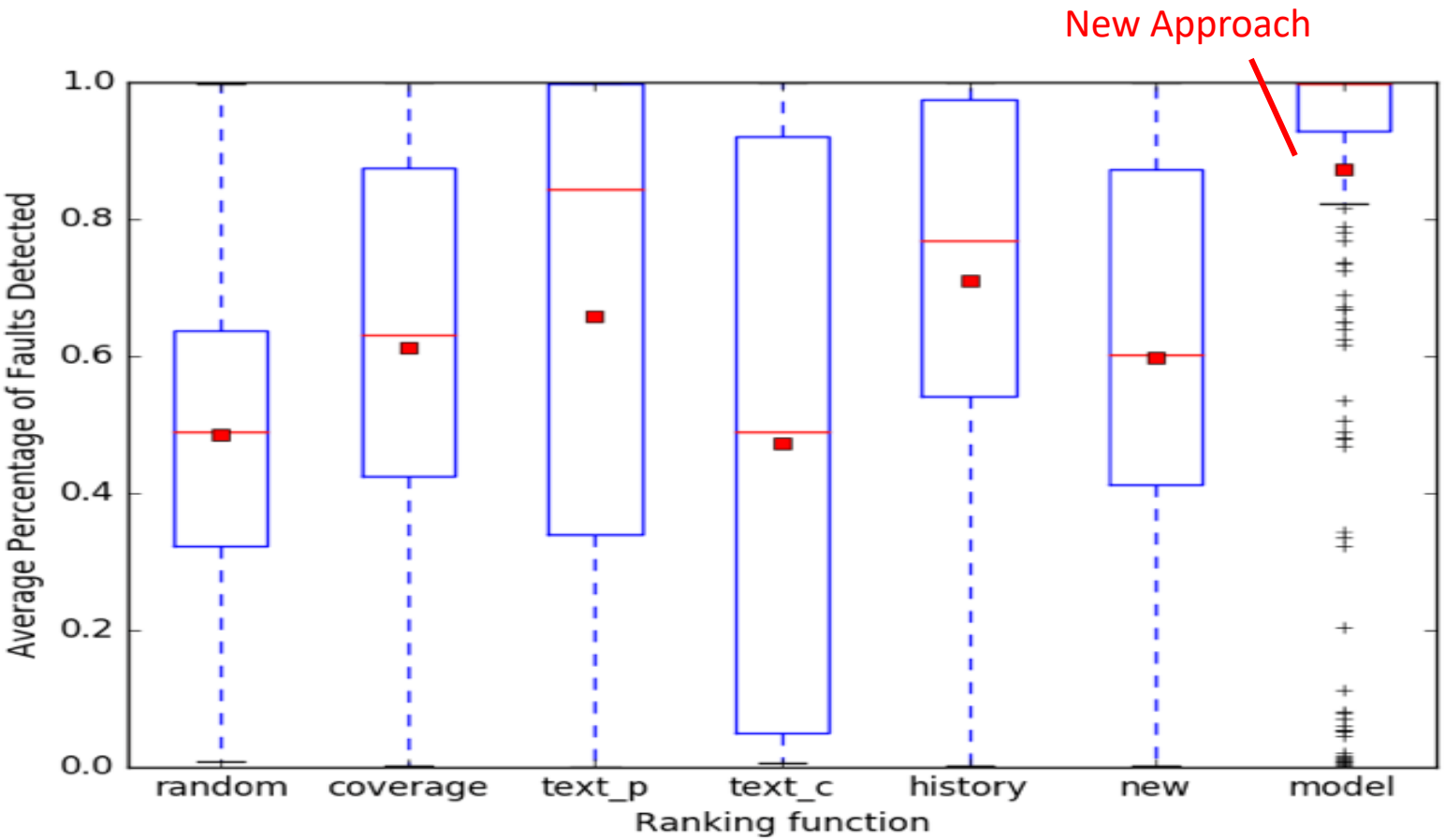


Figure 2: Average Percentage of Faults Detected

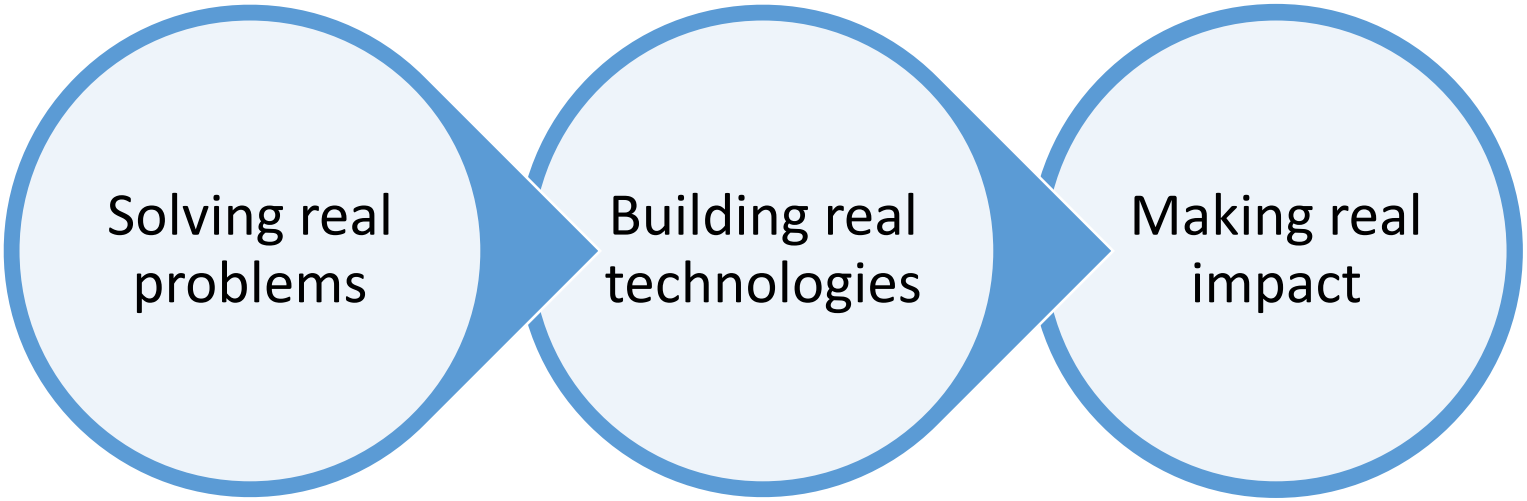
- New approach achieves highest APFD with least variance
  - Median: 85%
  - Average: 99%

Summary

- Main insights gained in conducting test prioritization in industry
- Novel learning-based approach to test prioritization
- Implementation currently in pilot use at Salesforce
- Empirical evaluation using a large Salesforce dataset

Getting Real to Produce Practice Impact

Software testing/analysis tools are naturally tied with software development practice



PARASOFT. Jtest®

MSRA XIAO

Pex

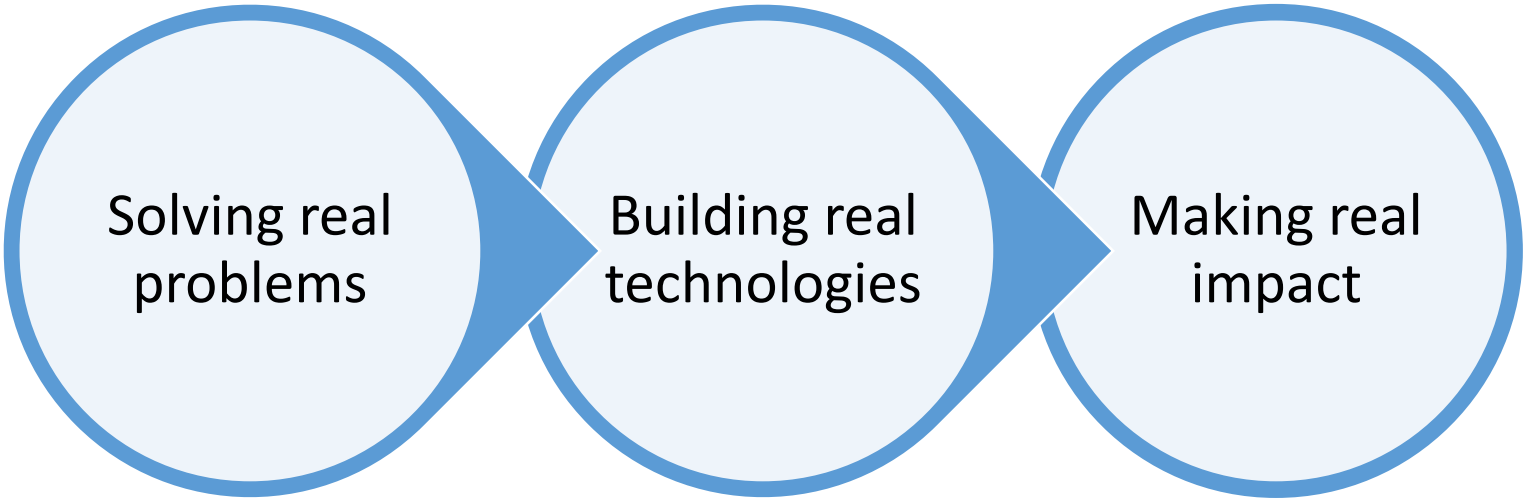
Visual Studio Enterprise 2015

salesforce

Thank you! Questions?

Getting Real to Produce Practice Impact

Software testing/analysis tools are naturally tied with software development practice



PARASOFT. Jtest®

MSRA XIAO

Pex

Visual Studio Enterprise 2015

salesforce