SE 4367.001

Software Testing Verification Validation and Quality Assurance

Wei Yang

Department of Computer Science
University of Texas at Dallas
Outline

• Overview
• Course Content
Outline

• Overview
• Course Content
Course Info

- Instructor: Wei Yang
- Office: ECSS 4.225
- Email: wei.yang@utdallas.edu
- Homepage: http://youngwei.com/
- TA: TBD
- Course web page
Research Interests

- **MobileSecurity** ([WHYPER](#), [Pluto](#), [AppContext[1][2][3]](#), [Telemade](#), [MRV](#), [CLAP[1]](#), [EnMobile](#), [MalScan](#))

- **Automated Testing** ([ORBIT](#), [WCTester[1][2]](#), [NMTtest[1]](#), [REINAM](#))

- **SE/Security for Machine Learning** ([PerInv](#), [MRV](#), [Telemade](#), [NMTtest](#))

- **ML/NLP for SE/Security** ([WHYPER](#), [Pluto](#), [CLAP](#), [SemRegex[1]](#), [REINAM](#))

- **IoT Security** ([iRuler](#))
## CSRankings: Computer Science Rankings

CSRankings is a metrics-based ranking of top computer science institutions around the world. Click on a triangle (▶️) to expand areas or institutions. Click on a name to go to a faculty member's home page. Click on a chart icon (📊) after a name or institution to see the distribution of their publication areas as a bar chart. Click on a Google Scholar icon (🔍) to see publications, and click on the DBLP logo (📚) to go to a DBLP entry.

Applying to grad school? Read this first.

Rank institutions in USA by publications from 2011 to 2021.

### All Areas [off | on]

- [AI [off | on]
  - Artificial intelligence
  - Computer vision
  - Machine learning & data mining
  - Natural language processing
  - The Web & information retrieval

- [Systems [off | on]
  - Computer architecture
  - Computer networks
  - Computer security
  - Databases
  - Design automation
  - Embedded & real-time systems
  - High-performance computing
  - Mobile computing
  - Measurement & perf. analysis
  - Operating systems
  - Programming languages
  - Software engineering

<table>
<thead>
<tr>
<th>#</th>
<th>Institution</th>
<th>Count</th>
<th>Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Carnegie Mellon University</td>
<td>25.3</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>North Carolina State University</td>
<td>20.7</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Univ. of California - Irvine</td>
<td>17.3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Univ. of Illinois at Urbana-Champaign</td>
<td>16.8</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>Purdue University</td>
<td>13.9</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>Univ. of California - Davis</td>
<td>13.2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>University of Southern California</td>
<td>11.9</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>University of Texas at Dallas</td>
<td>11.6</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>Univ. of California - Los Angeles</td>
<td>10.6</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>University of Washington</td>
<td>10.3</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>George Mason University</td>
<td>9.9</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>Iowa State University</td>
<td>9.9</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>Massachusetts Institute of Technology</td>
<td>9.0</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>University of Virginia</td>
<td>8.6</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>Univ. of California - Berkeley</td>
<td>8.1</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>College of William and Mary</td>
<td>7.2</td>
<td>8</td>
</tr>
</tbody>
</table>
Electronic communication

- [https://elearning.utdallas.edu](https://elearning.utdallas.edu): announcements

- MS Teams: questions, answers
• This course will mainly be a walk through for https://www.fuzzingbook.org/.

• Some (text)books recommended

• Reading linked from Schedule (Provided later)

• Course Website: http://youngwei.com/page/CS4367-001-22S/index.html
Grading

- Mid-term Exam (30%)
- Assignments (20%)
- Online Discussion & Class Participation (10%)
- Final Exam (40%)

For fairness, we REPORT all cheating

- Please avoid copy-pasting as much as possible. For any material (especially graphics and anything included by copy-pasting) not created by you but included in your deliverable, you must acknowledge the source on the same page.
Outline

• Overview
• Course Content
• Sample presentation by the instructor
The facts

• Only 32% of software projects are considered successful
  • (full featured, on time, on budget)

• Software failures cost the US economy $59.5 billion dollars every year [NIST 2002 Report]

• On average, 1-5 bugs per KLOC (thousand lines of code) In mature software (more than 10 bugs in prototypes)

- 35MLOC
- 63K known bugs at the time of release
- 2 bugs per KLOC
Testing

- Caused due to numeric overflow error
  - Attempt to fit 64-bit format data in 16-bit space

- Cost
  - $100M’s for loss of mission
  - Multi-year setback to the Ariane program

- Read more at [http://www.around.com/ariane.html](http://www.around.com/ariane.html)
Security Vulnerabilities

• Exploits of errors in programs

• Widespread problem
  • Moonlight Maze (1998)
  • Code Red (2001)
  • Titan Rain (2003)
  • Stuxnet Worm

• Getting worse ...

2011 Mobile Threat Report (Lookout™ Mobile Security)

• 0.5-1 million Android users affected by malware in first half of 2011
• 3 out of 10 Android owners likely to face web-based threat each year
• Attackers using increasingly sophisticated ways to steal data and money
A few more examples

Pac-Man (1980)
- Should always have no ending
- Has “Split Screen” at level 256
- Cause: Integer overflow
- 8 bits: maximum representable value

\[
\begin{array}{c}
11111111 \\
255 \\
\end{array} + \begin{array}{c}
00000000 \\
1 \\
\end{array} = \begin{array}{c}
1000000000 \\
0 \end{array}
\]
A few more examples

• Mars Climate Orbiter (1998)
  • Sent to Mars to relay signal from Mars

• Lander
  • Smashed to the planet

• Cause: Failing to convert between different metric standards
  • Software that calculated the total impulse presented results in pound-seconds
  • The system using these results expected its inputs to be in newton-seconds
A few more examples

• USS Yorktown (1997)
  • Left dead in the water for 3 hours

• Cause: Divide by zero error

\[
\frac{\text{Number}}{0} = \text{Explosion}
\]
Fuzzing

/dev/random → Input → Execute → Program

w00t[a%#]

Bugs = Crashes (segfaults, aborts, etc.)
Pulling JPEGs out of thin air

This is an interesting demonstration of the capabilities of afl; I was actually pretty surprised that it worked!

$ mkdir in_dir
$ echo 'hello' >in_dir/hello
$ ./afl-fuzz -i in_dir -o out_dir ./jpeg-9a/djpeg
• AFLFast [CCS 2016]
• Driller [NDSS 2016]
• AFLGo [CCS 2017]
• Vuzzer [NDSS 2017]
• Steelix [FSE 2017]
• SlowFuzz [CCS 2017]
• PerfFuzz [ISSTA 2018]
• FairFuzz [ASE 2018]
• Angora [IEEE S&P 2018]
• T-Fuzz [IEEE S&P 2018]
• NEUZZ [IEEE S&P 2019]
• Nautilus [NDSS 2019]
• Redqueen [NDSS 2019]
• Superion [ICSE 2019]
• MOPT [Usenix Sec 2019]
• GRIMOIRE [Usenix Sec 2019]
• MemFuzz [ICST 2019]
• Zest [ISSTA 2019]
• DifFuzz [ICSE 2019]
• AFLSmart [IEEE TSE 2019]
• FuzzChick [OOPSLA 2019]
• ...
Fuzzing for Security

Releasing jsfunfuzz and DOMFuzz
Tuesday, July 28th, 2015

Today I’m releasing two fuzzers: jsfunfuzz, which tests JavaScript engines, and DOMFuzz, which tests layout and DOM APIs.

Over the last 11 years, these fuzzers have found 6450 Firefox bugs, including 790 bugs that were rated as security-critical.

What is Microsoft Security Risk Detection?

Security Risk Detection is Microsoft’s unique fuzz testing service for finding security-critical bugs in software. Security Risk Detection helps customers quickly adopt practices and technology battle-tested over the last 15 years at Microsoft.

Google Testing Blog
Announcing OSS-Fuzz: Continuous Fuzzing for Open Source Software
Thursday, December 01, 2016

Linux 4.14-rc5
From: Linus Torvalds
Date: Sun Oct 15 2017 - 21:48:49 EST

The other thing perhaps worth mentioning is how much we’ve been doing fuzzing in the kernel, and it’s helping with that. We’ve always done fuzzing (who remembers the old “crashme” program that just generated random code and jumped to it?) and now we’re doing some nice targeted fuzzing of driver subsystems etc., and the results have been various (not just this last week either) coming out of those efforts. Very nice to see.

CVE-2014-6277: “ShellShock” bug in Bash
CVE-2014-0160: “Heartbleed” bug in OpenSSL
Responses from Computing Researchers to HUD’s Implementation of the Fair Housing Act’s Disparate Impact Standard

January 8th, 2020 / in Announcements, CCC, policy, research horizons, Research News / by Helen Wright

The following blog post is from Computing Community Consortium (CCC) Vice Chair Elizabeth Bradley (University of Colorado Boulder) and CCC Executive Council member Suresh Venkatasubramanian (University of Utah).

Algorithmic bias can be insidious, making it all but impossible to pinpoint factors that contribute to discrimination. This is particularly concerning in the context of high-stakes decisions. The new Department of Housing and Urban Development (HUD) guidelines around the use of algorithms to aid in housing decisions are an example of this. This HUD proposal acknowledges the existence of algorithmic bias but would shift much of the burden of proof to demonstrate discriminatory behavior back onto the plaintiffs, using standards for algorithmic transparency and explainability that seem unmoored from extant science about what we can hope to extract from algorithmic decision pipelines. Among other things, this would allow landlords and lenders to deflect lawsuits with an overly naive statistical approach, looking at individual factors rather than taking them in combination and thereby ignoring the potential collective effect of many lenders using the same third-party algorithm. Writing in Forbes, Elizabeth Fernandez suggests that this could undermine the Fair Housing Act.

Computing researchers who study these issues have submitted formal responses to the public call for comments regarding these new guidelines. These included a coordinated response by members of the GRAIL network, a new initiative led by the Center for Democracy and Technology (CDT) and the R Street Initiative. GRAIL’s goal is to connect technical and policy experts to inform discussions around technology policy in Washington and provide deep, rapid responses to questions of tech policy. Their response, which was led by Natasha Duarte at CDT and involved CCC Council member Suresh Venkatasubramanian, details how the different components of the
This Proposed Rule document was issued by the Department of Housing and Urban Development (HUD). For related information, Open Docket Folder.

**Action**

Proposed rule.

**Summary**

Title VIII of the Civil Rights Act of 1968, as amended (Fair Housing Act or Act), prohibits discrimination in the sale, rental, or financing of dwellings and in other housing-related activities on the basis of race, color, religion, sex, disability, familial status, or national origin. HUD has long interpreted the Act to create liability for practices with an unjustified discriminatory effect, even if those practices were not motivated by discriminatory intent. This rule proposes to amend HUD's interpretation of the Fair Housing Act's disparate impact standard to better reflect the Supreme Court's 2015 ruling in Texas Department of Housing and Community Affairs v. Inclusive Communities Project, Inc., and to provide clarification regarding the application of the standard to State laws governing the business of insurance. This rule follows a June 20, 2018, advance notice of proposed rulemaking, in which HUD solicited comments on the disparate impact standard set forth in HUD's 2013 final rule, including the disparate impact rule's burden-shifting approach, definitions, and causation standard, and whether it required amendment to align with the decision of the Supreme Court in Inclusive Communities Project, Inc.
UI testing agent with reinforcement learning

If $(x==8)$

$$x+=1$$

$$x+=2$$

Traditional program (control flow graph)

Neural network

Input

Program Logic

Covered

Output

Not Covered

An Empirical Study of Android Test Generation Tools in Industrial Cases

Wang et al. ASE 2018
Yin-Yang view of data-driven app testing

- **Security**: Behavioral data: API invocations, network incoming and outgoing traffic, keyboard logs, app execution trace, bug/crash reports, static analysis.

- **Behavioral data**: Behavioral data: API invocations, network incoming and outgoing traffic, keyboard logs, app execution trace, bug/crash reports, static analysis.

- **User Awareness**: Check Consistency

- **User data**: Contextual data: user usage data, user reviews, UI screen (labels, hints, screen, buttons, sequence of screen), app descriptions, privacy policy, pictures/videos, tags (app category).

- **Main Challenge**: Lack of labeled data

- **Short Term**: Multimodal representation learning

- **Long Term**: Transfer learning
Testing criterion for machine learning

Traditional program (control flow graph)

- Variable x
- If (x == 8)
  - x += 1
  - x += 2

Neural network

Input

Program Logic

Covered

Not Covered

Output
Testing criterion for NLP

Automatic Detection of Under- and Over-Translation in Neural Machine Translation

Peng et al.
Under submission
Code Obfuscation/De-obfuscation/Transformation

Traditional program (control flow graph) vs. Neural network

Obfuscated Code

De-obfuscated Code

Open-source, unobfuscated applications

Probabilistic model $P(x|y)$

Semantic representation
Learning-based testing

- Security of AI software
- Integrity (Causing ML to produce unintended results)
- Privacy (Learning sensitive information about training data)
Physical testing to smart cities systems

- New application: smartphone -> smart infrastructure
  - Transportation, smart home (intelligent dwelling), smart city (urban planning)
Reference

- https://securify.chainsecurity.com/
- https://www.probfuzz.com/