# Randomizing Regression Tests Using Game Theory

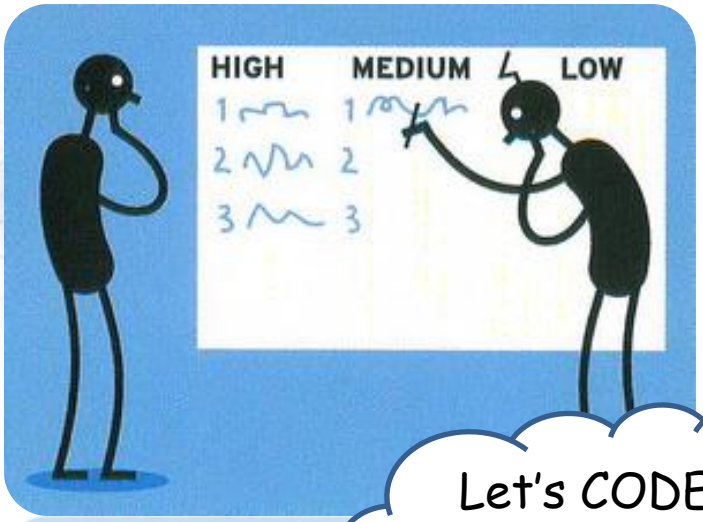
**Nupul Kukreja**, William G.J. Halfond, Milind Tambe

ASE 2013

# Outline

- Motivation
- Problem(s) with traditional test scheduling
- Game Theory and Randomization
- Modeling software testing as a 2-player game
- Evaluation
- Conclusion & Future Work

# Motivation


Dude! Suite XX is not gonna run!

Let's CODE NOW FIX LATER

DEVS

The deadline is **close** too!

# Motivating Problem(s)

- Existing test case scheduling activities are deterministic
- Developers know which test cases will be executed when
- Developers can check in insufficiently tested code closer to delivery deadline
- High-turn around time for fixing bugs in low priority features
- Random test-scheduling helpful but treats each test case as equally important

# Software Testing as a 2-Player Game

- This *tension* between software testers and developers can be modeled as a two-player game
- We *solve* the *game* to answer the following question:
  - Given an *adaptive adversary* (developers) and resource constraints (testers) what is the *optimum* test-scheduling *strategy* that *maximizes* the tester's *expected payoff*?

# Game Theory

- Study of strategic decision making among multiple players – corporations, software agents, testers and developers, regular humans etc.,

# Two-player "Security" Game

| | | Adversary: Terminal 1 | Adversary: Terminal 2 |
|---|---|---|---|
| Defender 60% | Terminal 1 | -3 / 5 | 1 / -1 |
| Defender 40% | Terminal 2 | 5 / -5 | -1 / 2 |

## Security game assumptions:

1. What is good for one player (+ve payoff) is bad for the other (-ve payoff)
2. Adversary can conduct perfect surveillance and act appropriately i.e., these are simultaneous move games or Stackelberg games

# Testing Game

| | | | Developer | |
|---|---|---|---|---|
| | | | Requirement 1 | |
| | | | Check in ITC* | Check in PC* |
| Tester | Requirement 1 | Test | -3<br>5 | 1<br>-1 |
| | | Don't Test | 5<br>-5 | -1<br>2 |

*ITC: Insufficiently tested code

*PC: Perfect code i.e., 100% tested

# Testing Game

| | Req 1 | | Req 2 | | Req 3 | | Req 4 | | Req 5 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ |
| **Tester** | 2 | -10 | 7 | -4 | 6 | -1 | 9 | -9 | 9 | -9 |
| **Developer** | -7 | 4 | -1 | 3 | -6 | 5 | -3 | 7 | -10 | 3 |

We solve the game 'efficiently' using the ERASER algorithm for solving security games

**Expected Payoff: Our Approach**

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| **Tester** | -8.32 | -2.52 | 0.61 | -0.83 | -8.26 |
| **Developer** | 2.46 | 2.46 | 2.46 | 2.46 | 2.46 |

**Expected Payoff: Uniform Random**

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| **Tester** | -7.6 | -.18 | 0.4 | -5.4 | -5.4 |
| **Developer** | 1.8 | 2.2 | 2.8 | 5 | 0.4 |

# Conclusion & Future Work

- Mapping of security games to testing games
- Can help increase software quality by deterring developers from checking in insufficiently tested code
- Future plan: To create a game theory based testing framework and evaluate its effectiveness using open source projects