

# Last Lecture

Testing, Quality Assurance, and Maintenance  
Winter 2019

Prof. Arie Gurfinkel



Syntax versus Semantics

# **TESTING AND VERIFICATION**

# Testing and Verification / Quality Assurance

**Testing:** Software validation the “old-fashioned” way

- create a test suite (a set of test cases)
- run and identify failures
- fix to address failures and repeat
- done when the test suite passes and achieves a desired criteria

**Verification:** formally prove that a computing system satisfies its specifications

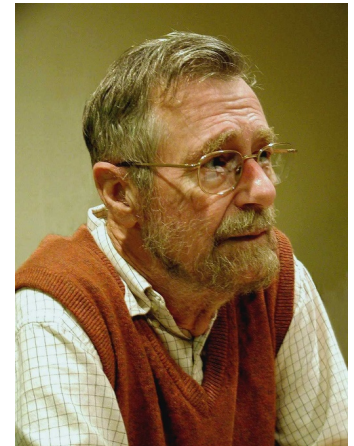
- Rigor: well established mathematical foundations
- Exhaustiveness: considers all possible behaviors of the system, i.e., finds all errors
- Automation: uses computers to build reliable computers

“Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.”

*Edsger W. Dijkstra*

Very hard to test the portion inside the “if” statement!

```
input x
if (hash(x) == 10) {
    ...
}
```



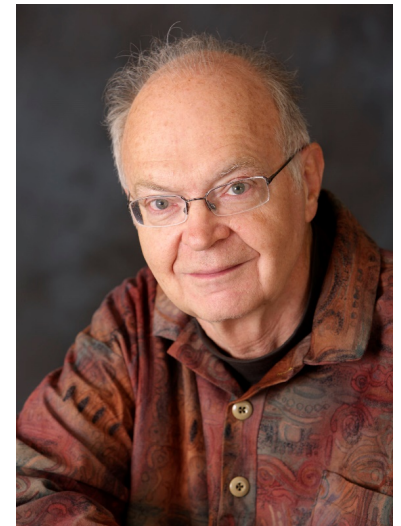
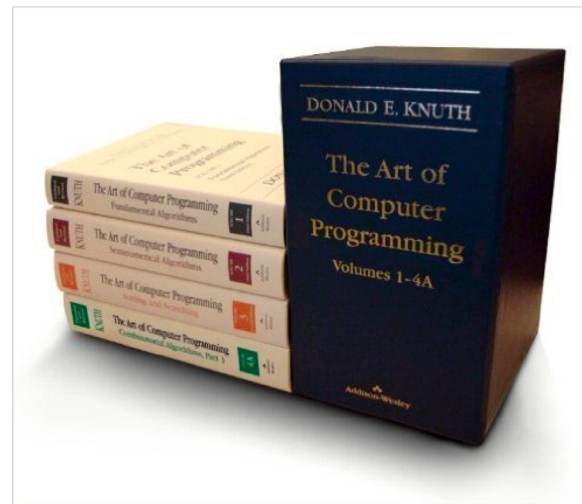


“Beware of bugs in the above code; I have only proved it correct, not tried it.”

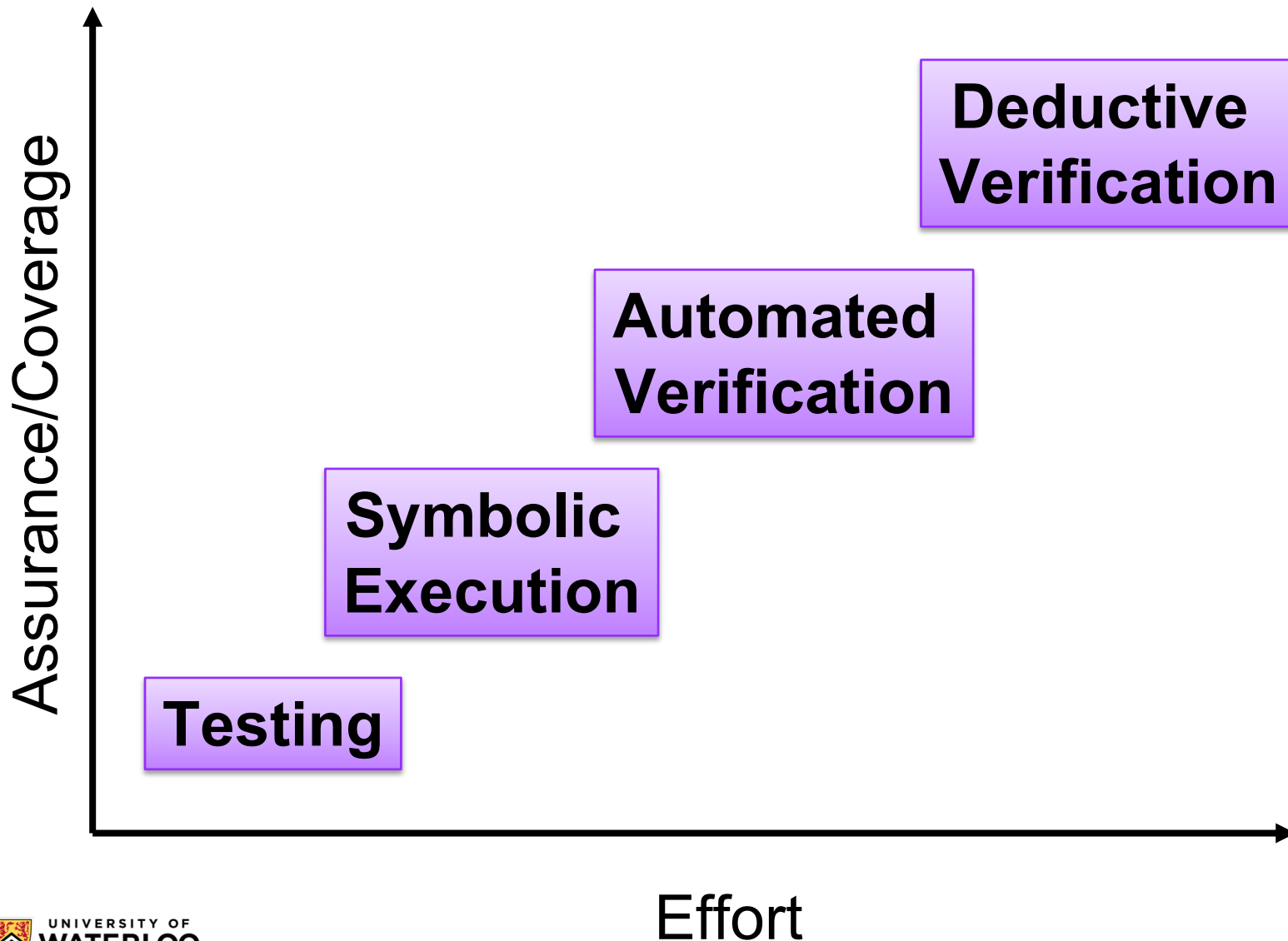
*Donald Knuth*

You can only verify what you have specified.

Testing is still important, but can we make it less impromptu?



# (User) Effort vs (Verification) Assurance



# Undecidability

A problem is undecidable if there does not exist a Turing machine that can solve it

- i.e., not solvable by a computer program

The halting problem

- does a program  $P$  terminate on input  $I$
- proved undecidable by Alan Turing in 1936
- [https://en.wikipedia.org/wiki/Halting\\_problem](https://en.wikipedia.org/wiki/Halting_problem)

Rice's Theorem

- for any non-trivial property of partial functions, no general and effective method can decide whether an algorithm computes a partial function with that property
- in practice, this means that there is no machine that can always decide whether the language of a given Turing machine has a particular nontrivial property
- [https://en.wikipedia.org/wiki/Rice%27s\\_theorem](https://en.wikipedia.org/wiki/Rice%27s_theorem)

# Topics Covered in the Course

## Foundations

- syntax, semantics, abstract syntax trees, visitors, control flow graphs

## Testing

- coverage: structural, dataflow, and logic

## Symbolic Execution

- using SMT solvers, constraints, path conditions, exploration strategies
- building a (toy) symbolic execution engine

## Deductive Verification

- Hoare Logic, weakest pre-condition calculus, verification condition generation
- verifying algorithm using Dafny, building a small verification engine

## Automated Verification

- ~~(basics of) software model checking~~

# Verification Tools in Practice

Testing, Quality Assurance, and Maintenance  
Winter 2019

Prof. Arie Gurfinkel

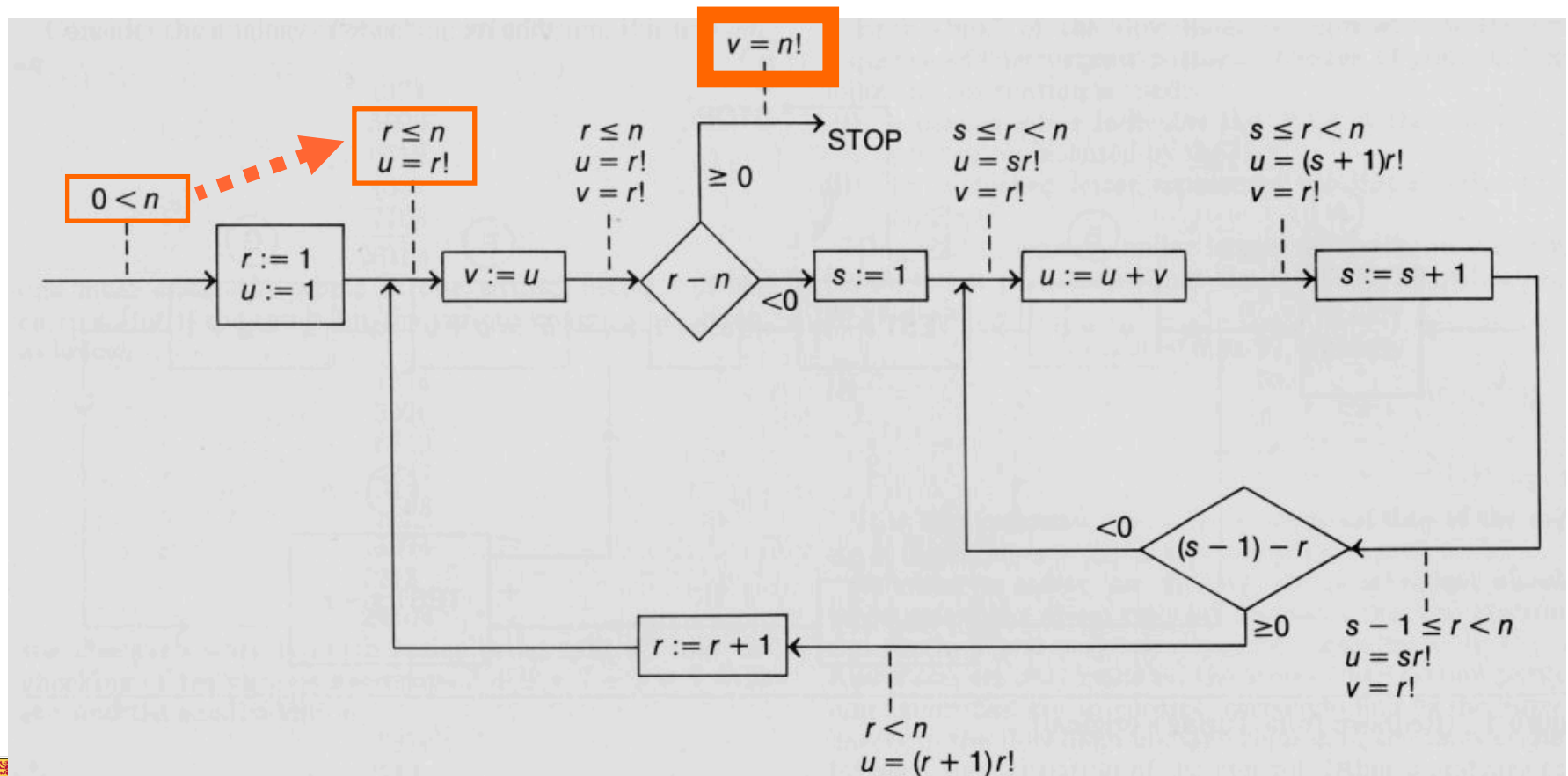


# Turing, 1949

Alan M. Turing. "Checking a large routine", 1949

How can one check a routine in the sense of making sure that it is right?

programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.



# Verification Competition

<http://www.pm.inf.ethz.ch/research/verifythis.html>

# Microsoft Visual Studio Products

## Code Contracts

- <https://marketplace.visualstudio.com/items?itemName=RiSEResearchinSoftwareEngineering.CodeContractsforNET>
- <https://github.com/Microsoft/CodeContracts>
- statically and dynamically checked method pre- and post-conditions

## IntelliTest

- <https://www.visualstudio.com/en-us/docs/test/developer-testing/intellitest-manual/introduction>
- automated test generation by dynamic symbolic execution



# WHY3

<http://why3.lri.fr/>

# VeriFast

<https://github.com/verifast/verifast>

# Viper

<http://www.pm.inf.ethz.ch/research/viper.html>

# Open JML

<http://www.openjml.org/>

# The KeY Project

<https://www.key-project.org/>

# Proving that Android's, Java's and Python's sorting algorithm is broken (and showing how to fix it)

🕒 February 24, 2015   📁 Envisage   ✍ Written by Stijn de Gouw. 🧑 \$s

Tim Peters developed the **Timsort hybrid sorting algorithm** in 2002. It is a clever combination of ideas from merge sort and insertion sort, and designed to perform well on real world data. TimSort was first developed for Python, but later ported to Java (where it appears as `java.util.Collections.sort` and `java.util.Arrays.sort`) by **Joshua Bloch** (the designer of Java Collections who also pointed out that **most binary search algorithms were broken**). TimSort is today used as the default sorting algorithm for Android SDK, Sun's JDK and OpenJDK. Given the popularity of these platforms this means that the number of computers, cloud services and mobile phones that use TimSort for sorting is well into the billions.

<http://envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/>

# Frama-C

<https://frama-c.com/>

# SPARKPro

<http://www.adacore.com/sparkpro/>




# Amazon S2N

The screenshot shows a web browser window displaying the AWS Security Blog. The address bar shows the URL <https://aws.amazon.com/blogs/security/automated-reasoning-and-amazon-s2n/>. The page header includes the AWS logo, navigation links (Menu, Products, Solutions, Pricing, Software, Support, More), and a 'Create an AWS Account' button. The main content area features the title 'Automated Reasoning and Amazon s2n' by Colm MacCarthaigh, dated 08 SEP 2016. The article text discusses the introduction of Amazon s2n in June 2015, its design goals, and its current status. A large 's2n' logo is prominently displayed. The right sidebar contains a search bar, 'Most recent posts' (including 'New AWS Big Data Blog Post: Analyze Security, Compliance, and Operational Activity Using AWS CloudTrail and Amazon Athena'), and 'Other AWS blogs' (including 'The Official AWS Blog', 'Amazon SES', and 'AWS Architecture').

Automated Reasoning and Amazon s2n

by Colm MacCarthaigh | on 08 SEP 2016 | in [Announcements](#) | [Permalink](#) | [Comments](#)

In June 2015, AWS Chief Information Security Officer Stephen Schmidt [introduced](#) AWS's new Open Source implementation of the SSL/TLS network encryption protocols, [Amazon s2n](#). s2n is a library that has been designed to be small and fast, with the goal of providing you with network encryption that is more easily understood and fully auditable.



In the 14 months since that announcement, development on s2n has continued, and we have merged more than 100 pull requests from 15 contributors on [GitHub](#). Those active contributors include members of the Amazon S3, Amazon CloudFront, Elastic Load Balancing, AWS Cryptography Engineering, Kernel and OS, and Automated Reasoning teams, as well as 8 external, non-Amazon Open Source contributors.

At the time of the initial s2n announcement, three external security evaluations and penetration tests on s2n had been completed. Those evaluations were code reviews and testing completed by security-focused experts, and came in addition to the code reviews and testing that are applied to every code change at Amazon as standard practice. We have continued to perform such evaluations, and we are pleased to have s2n be the focus of additional analysis from external academic and professional security researchers.

**Adding automated reasoning to s2n**

Because of s2n's role as security-critical software, one of our goals is to use s2n as a proving ground for new *automated reasoning* testing and assurance techniques that we can refine for broader adoption within Amazon and beyond. Increasingly, the availability of compute resources on demand such as [Amazon EC2](#) makes it possible to perform extensive security analysis, even on every code change.

Search the Security Blog

Search

Most recent posts

- New AWS Big Data Blog Post: Analyze Security, Compliance, and Operational Activity Using AWS CloudTrail and Amazon Athena
- Now Generally Available – AWS Organizations: Policy-Based Management for Multiple AWS Accounts
- s2n Is Now Handling 100 Percent of SSL Traffic for Amazon S3
- Easily Replace or Attach an IAM Role to an Existing EC2 Instance by Using the EC2 Console
- How to Audit Your AWS Resources for Security Compliance by Using Custom AWS Config Rules

Other AWS blogs

- The Official AWS Blog
- Amazon SES
- AWS Architecture

<https://aws.amazon.com/blogs/security/automated-reasoning-and-amazon-s2n/>

# IronClad and InronFleet

<https://github.com/Microsoft/Ironclad>

# Facebook Infer

Automatically prove correct memory handling (e.g., absence of null dereferencing)

<http://fbinfer.com/>

# KLEE



Symbolic execution for C/C++ based on  
LLVM

<https://klee.github.io/>

# Diffblue: AI for Code

[https://playground.diffblue.com/?utm\\_source=homepage](https://playground.diffblue.com/?utm_source=homepage)

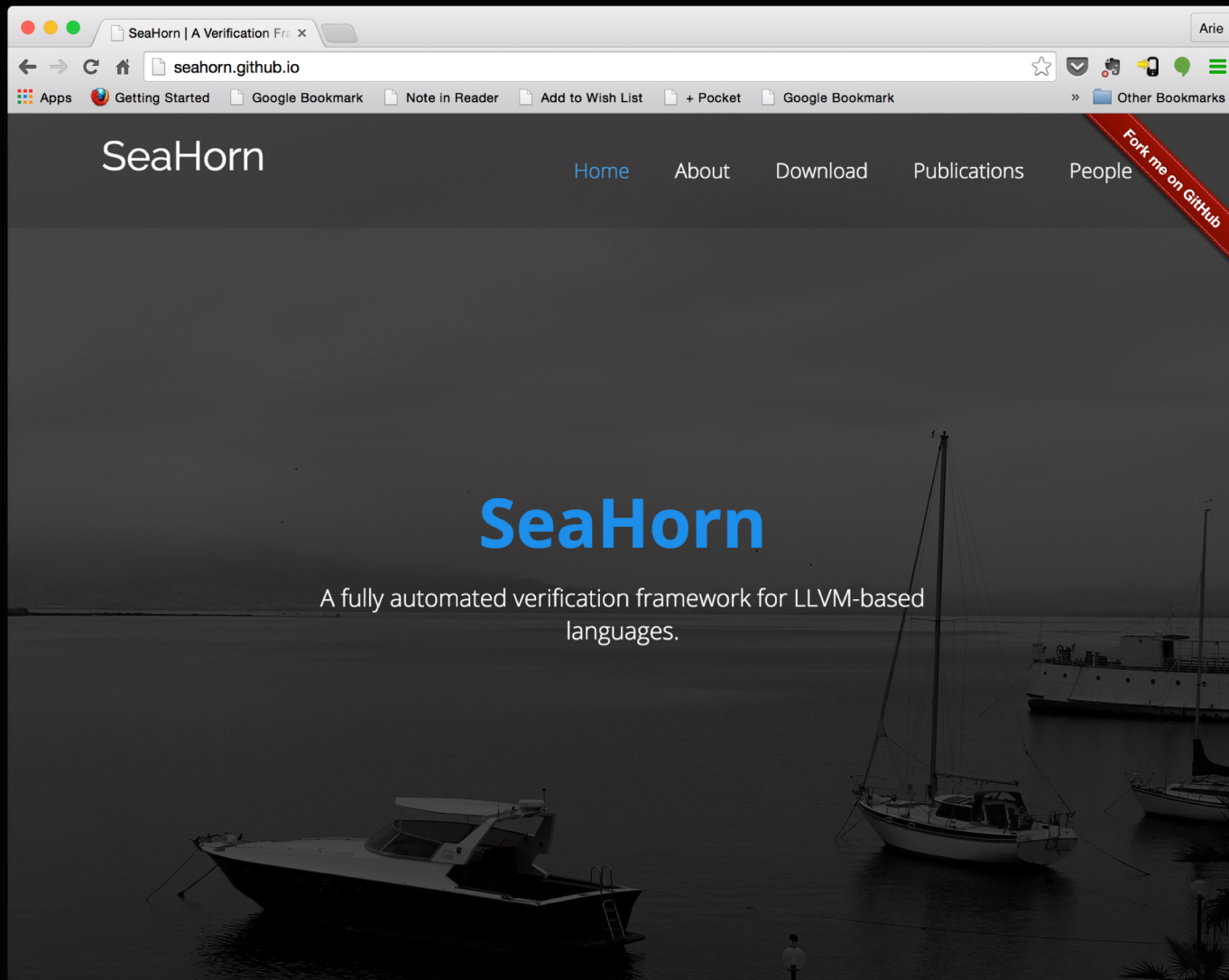
Automated test-case generation for Java

# Automated reasoning at AWS

<https://aws.amazon.com/blogs/security/tag/automated-reasoning/>

<https://www.youtube.com/watch?v=JfjLKBO27nw>

<https://blog.adacore.com/amazon-relies-on-formal-methods-for-the-security-of-aws>



<http://seahorn.github.io>



# Is Verification Enough

Can verified software fail?

Do we need both testing and verification?