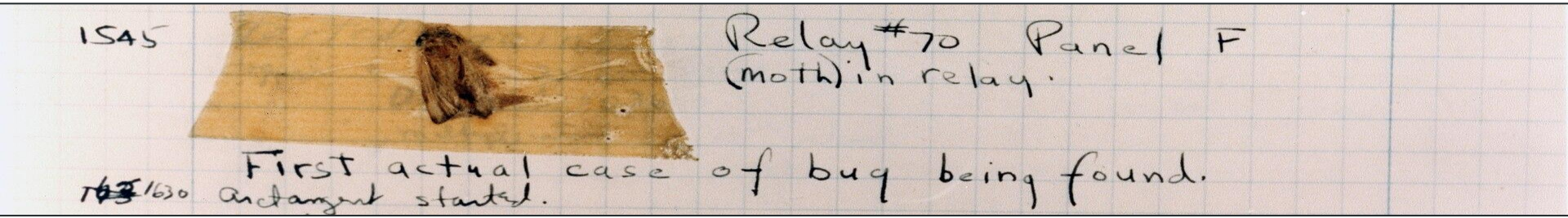
An Introduction to Software Testing

Fantastic Bugs and Where to Find Them



ENSEIRB-MATMECA · I3 GL
Test Logiciel
October 2023

Presentation

thomas.degueule@labri.fr
tdegueul.github.io

- CNRS Researcher at LaBRI
- Software *engineering* & *software languages*
- Reuse, modularity, evolution, testing
- Mutation testing

- Also a software developer & tester ;)

Course Objectives

- Master the general concept of software testing
- Grasp the alternatives to software testing (formal verification, code review, etc.)
- Know when/how to write *high-quality* tests, *systematically*

- How to *design tests* for object-oriented software
- How to *design object-oriented software* for tests
- The vast majority of the concepts we'll learn apply to any other language™

- Master popular testing tools & technologies
- **Effective software testing, *in practice***

Organization

- Nine two-hours sessions
- Most sessions will be a mix of lectures/discussions/lab work
- Project-based evaluation

- Interrupt me, ask questions and clarifications, discuss
- Reach out to me by email anytime (`thomas.degueule@labri.fr`)

- What's the best way to share material? (GitHub, Moodle, emails)

Quick Survey

Why do we test software?

0800 Arctan started

1000 " stopped - arctan ✓

{ 1.2700 · 9.037 847 025
9.037 846 995 correct

13" sec (032) MP - MC ~~1.9826647000~~ ~~2.130476415 (-3)~~ 4.615925059(-2)

(033) PRO 2 2.130476415

correct 2.130676415

Relays 6-2 in 033 failed special speed test

in Relay " 10.000 test.

Relays changed

Relay 2145

Relay 3370

1700 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545

Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

~~1615~~ 1630 Arctangent started.

1700 closed down.

Ariane 5: 1996

- Flight V88
- Self-destroyed after 36 seconds
- The inertial navigation system reused from Ariane 4 triggered a faulty conversion from 64-bit float to 16-bit integer
- The overflow caused an hardware exception
- Cost ~= $370M

Therac-25: 1985–1987

- Magnetic Resonance Imaging device implemented with PDP-11
- Race conditions triggered when the operator quickly switched radiation mode
- 100x the intended dose
- Several deaths, many injured

```
PATIENT NAME: John
TREATMENT MODE: FIX            BEAM TYPE: E     ENERGY (KeV):       10

                               ACTUAL           PRESCRIBED
        UNIT RATE/MINUTE         0.000000         0.000000
        MONITOR UNITS          200.000000       200.000000
        TIME(MIN)                0.270000         0.270000


GANTRY ROTATION (DEG)            0.000000         0.000000          VERIFIED
COLLIMATOR ROTATION (DEG)      359.200000       359.200000          VERIFIED
COLLIMATOR X (CM)               14.200000        14.200000          VERIFIED
COLLIMATOR Y (CM)               27.200000        27.200000          VERIFIED
WEDGE NUMBER                     1.000000         1.000000          VERIFIED
ACCESSORY NUMBER                 0.000000         0.000000          VERIFIED



DATE: 2012-04-16       SYSTEM: BEAM READY        OP.MODE: TREAT        AUTO
TIME: 11:48:58         TREAT: TREAT PAUSE                 X-RAY        173777
OPR ID: 033-tfs3p      REASON: OPERATOR          COMMAND:
```

And the list goes on and on…

- In September 1983, the Russian system Oko wrongly reported a nuclear attack by the USA. Operator Stanislav Petrov intervened to prevent a likely nuclear war
- The German social services and unemployment system A2LL wrongly filled up bank account numbers with zeros on the end instead of the beginning, making payment impossible
- …

- It is estimated that for any given software project, 40–50% is allocated to testing

Software Testing in Practice

"50% of the people at Microsoft are testers, and the programmers spend 50% of their time testing, thus ***Microsoft is more of a testing than a development organization****. When we do a new release of Windows, which is, say, a billion-dollar effort,* ***over half that is going into the quality****."*

Q&A: Bill Gates On Trustworthy Computing. John Foley and Chris Murphy. *Information Week*, 2002

"In an average day, Google runs ***150 Million tests****."*

Taming Google-Scale Continuous Testing. Memon et al. *International Conference on Software Engineering, 2017*

"We have created Chaos Monkey, a program that ***randomly chooses a server and disables it during its usual hours of activity****. Knowing that this would happen frequently has created a strong alignment among engineers to build redundancy and process automation to survive such incidents."*

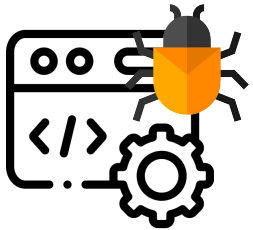
Netflix TechBlog, 2016

Terminology

*"A programmer makes an **error** (mistake), which results in a **fault** (defect, **bug**) in the software source code. If this fault is executed, in certain situations the system will produce wrong results, causing a **failure**."*

- Not all faults yields failures!
  - Dead code, exotic environments, etc.
- Software testing is all about finding *failures*, debugging is all about findings *faults*

Error/Mistake



Fault/Bug

Failure

Verification and Validation (V&V)

Verification

*"Are we building **the product right**?"*

- Assuming that the requirements are correct
- Ensures that the software meets its requirements
- Typically done during development, at different stages
- Done by *developers/testers*, not *users/stakeholders*

Validation

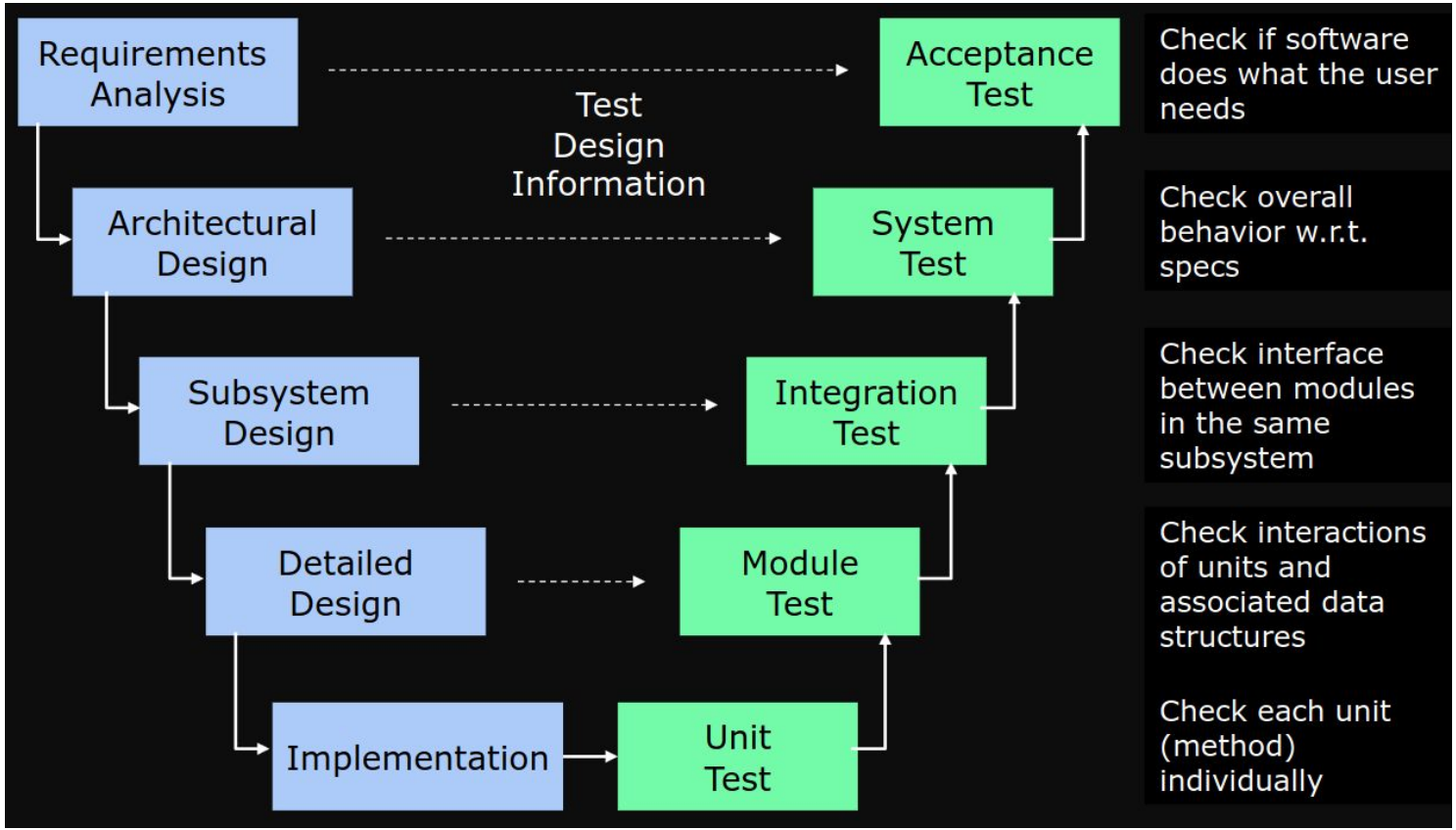
*"Are we building **the right product**?"*

- Question the software requirements
- Ensures that the software meets the intended usage by actual users
- Typically done at the end of development
- Done by *users/stakeholders*, not *developers/testers*

A Taxonomy of Tests

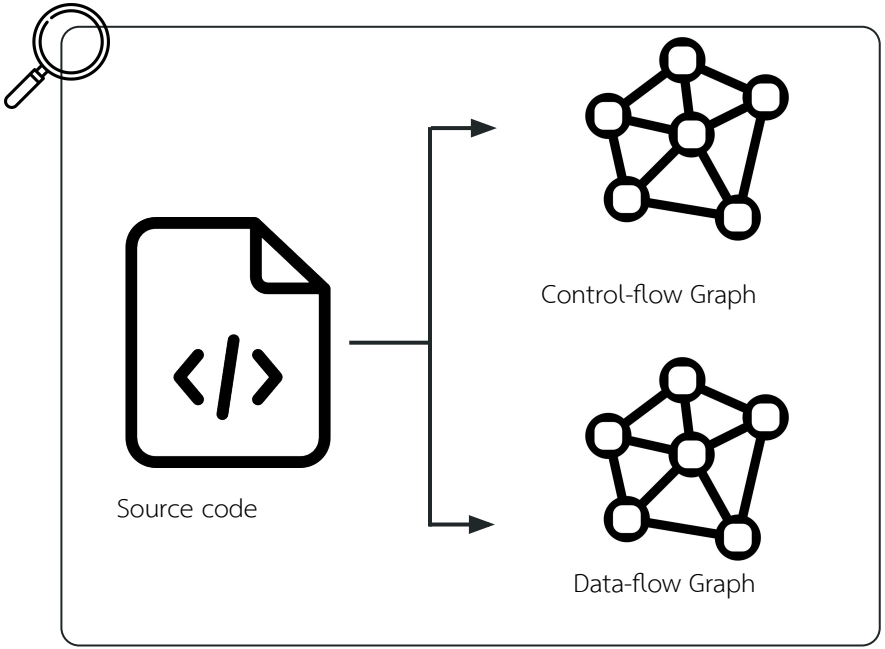
- Unit vs Integration vs System vs Acceptance
- Static vs Dynamic
- White-box vs Black-box
- Functional vs Extra-functional
- Domain-specific testing

Testing Levels and Granularity



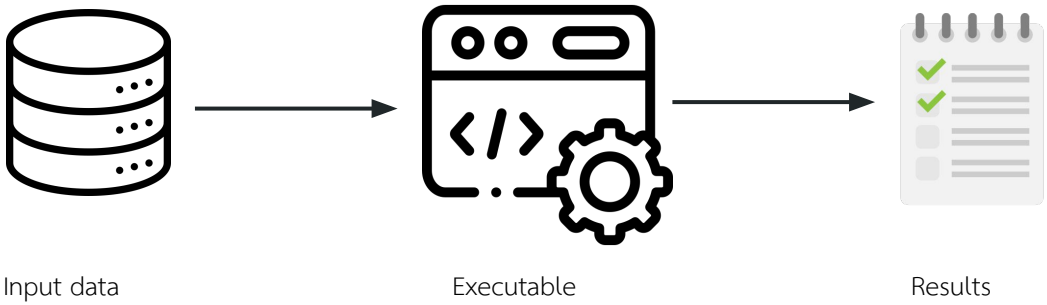
Static vs Dynamic Testing

- **Static testing** involves examining the System Under Test (SUT) *without executing it*
- By inspecting the source code directly or an abstraction/model
- Examples: code review, code inspection, model checking, etc.



Static vs Dynamic Testing

- **Dynamic testing** involves *running* the System Under Test (SUT) with carefully crafted inputs to inspect its outputs and compare them against an *oracle* to find failures
- Examples: acceptance testing, release candidates, fuzzing, exploratory testing, etc.
- This is our main focus for this course



White-box vs Black-box Testing

- **White-box** generally done by *developers* who know about the implementation
- Typically (but not necessarily!) done at the unit level
- Examples: function/statement/branch coverage, mutation testing, code review, etc

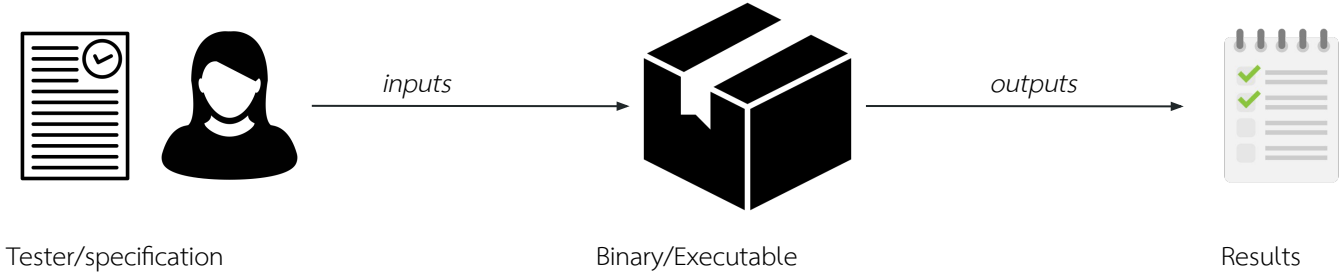
```
7  public class Library {
8      
9      public static void main(String[] args) {
10         Library library = new Library();
11         library.someLibraryMethod();
12     }
13     
14     public boolean someLibraryMethod() {
15         return true;
16     }
17     
18     public boolean notTestedLibraryMethod() {
19         return true;
20     }
21 }
```

JaCoCo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.jacoco.examples		58%		64%	24	53	97	193	19	38	6	12
org.jacoco.core		97%		93%	107	1,388	115	3,347	21	720	2	139
org.jacoco.agent.rt		77%		84%	31	121	62	310	21	74	7	20
jacoco-maven-plugin		90%		81%	35	183	44	407	8	110	0	19
org.jacoco.cli		97%		100%	4	109	10	275	4	74	0	20
org.jacoco.report		99%		99%	4	572	2	1,345	1	371	0	64
org.jacoco.ant		98%		99%	4	163	8	429	3	111	0	19
org.jacoco.agent		86%		75%	2	10	3	27	0	6	0	1
Total	1,355 of 27,352	95%	143 of 2,125	93%	211	2,599	341	6,333	77	1,504	15	294

White-box vs Black-box Testing

- **Black-box** generally done by *testers* who ignore the implementation
- Requires some kind of specification of the system, component, or unit under test
- Appropriate for unit testing, prevalent for higher-level testing
- Examples: partitioning, boundary values, pairwise, exploratory, etc



Extra-functional Testing

- Performance, security, safety, accessibility, usability, privacy, scalability, etc.
- aka. *non-functional* testing, but non-functional properties are often sort of functional
  - A brake that takes 2 seconds to trigger
  - A social network that leaks private data

Domain-specific Testing

```java
public void testFind_firstElement() {
  Iterable<String> list = Lists.newArrayList("cool", "pants");
  Iterator<String> iterator = list.iterator();
  assertEquals("cool", Iterators.find(iterator, Predicates.equalTo("cool")));
  assertEquals("pants", iterator.next());
}

public void testFind_lastElement() {
  Iterable<String> list = Lists.newArrayList("cool", "pants");
  Iterator<String> iterator = list.iterator();
  assertEquals("pants", Iterators.find(iterator, Predicates.equalTo("pants")));
  assertFalse(iterator.hasNext());
}
```

JUnit, TestNG, etc.

```javascript
describe('My First Test', () => {
  it('Gets, types and asserts', () => {
    cy.visit('https://example.cypress.io')

    cy.contains('type').click()

    // Should be on a new URL which
    // includes '/commands/actions'
    cy.url().should('include', '/commands/actions')

    // Get an input, type into it
    cy.get('.action-email').type('fake@email.com')

    //  Verify that the value has been updated
    cy.get('.action-email').should('have.value', 'fake@email.com')
  })
})
```

Cypress, Selenium, Puppeteer, Playwright, etc.

Static V&V Techniques

Manual Code Review & Pair Programming

- A survey of Microsoft developers shows that the n°1 benefit of pair programming is the introduction of *fewer bugs*

Pair Programming: What's in it for me?
Andrew Begel, Nachiappan Nagappan
In *Empirical Software Engineering and Measurement*, 2008.

Automated Code Review: SAT and Linters

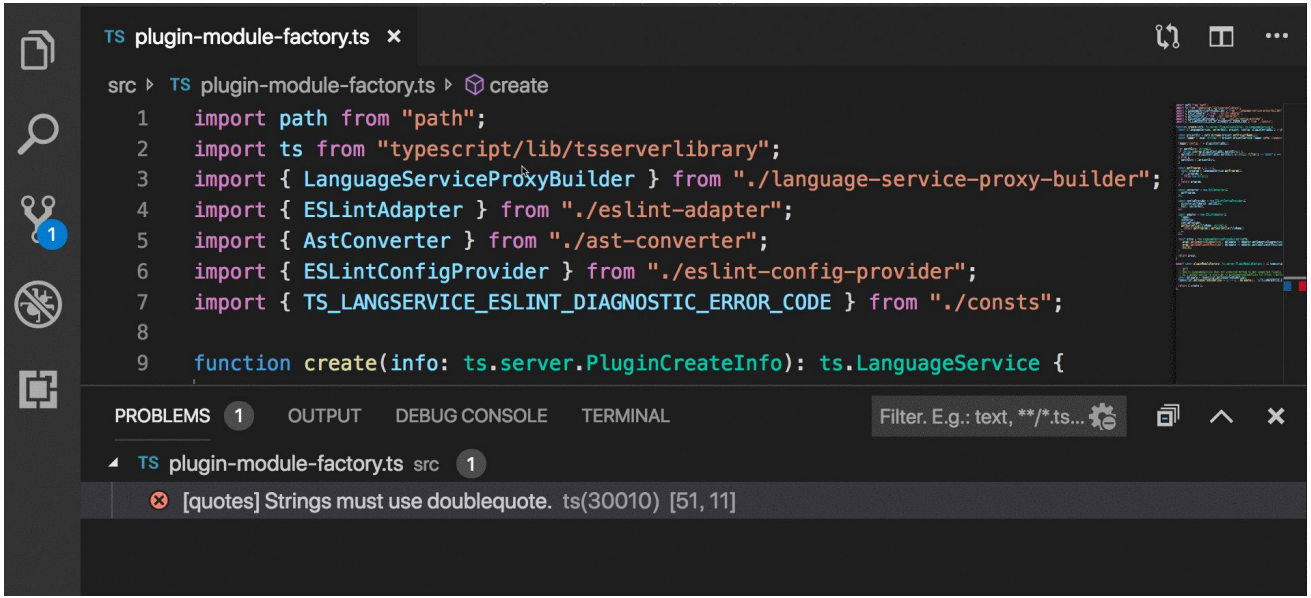
- Typically look for *code smells* and *best practices* rather than faults
  - SAT and linters know nothing about your specifications!

sonarlint

FindBugs™

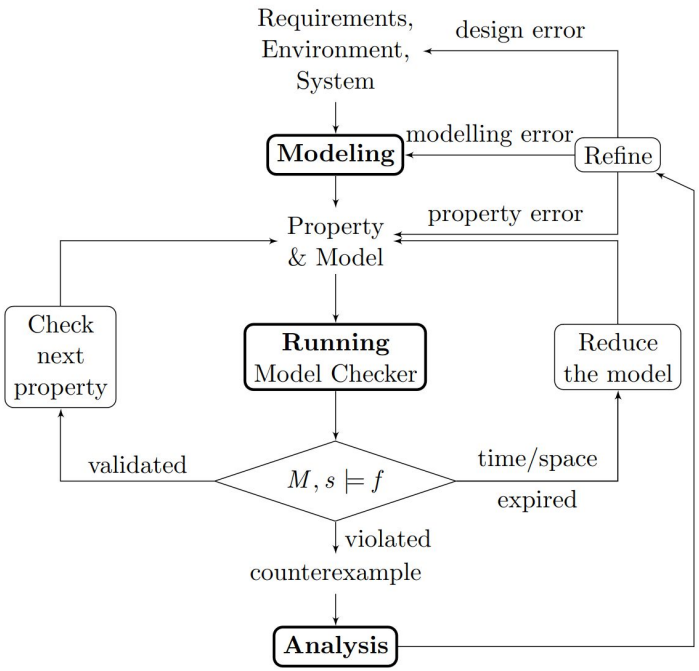
Semgrep

ESLint



Formal Software Verification: Model Checking

- Build a finite-state model of the system and check desirable properties against it
- Typically, properties expressed in temporal logic: liveness, correct states, termination, etc.

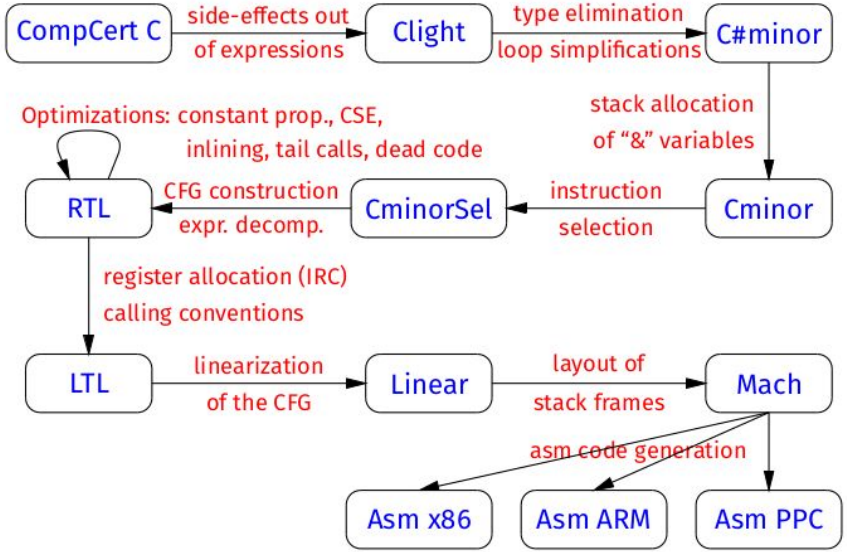


$\Box(requested \Rightarrow \Diamond received)$

Model Checking
E. M. Clarke
In *Foundations of Software Technology and Theoretical Computer Science. 1997.*

Formal Software Verification: Program Proofs

- CompCert: a C compiler with a mathematical, machine-checked proof that the generated executable code behaves exactly as prescribed by the semantics of the source program




```
Lemma expect_incr: forall te e t1 t2 e',
  expect e t1 t2 = OK e' -> S.satisf te e' -> S.satisf t
Proof.
  unfold expect; intros. destruct (typ_eq t1 t2); inv H;
Qed.
Global Hint Resolve expect_incr: ty.

Lemma expect_sound: forall e t1 t2 e',
  expect e t1 t2 = OK e' -> t1 = t2.
Proof.
  unfold expect; intros. destruct (typ_eq t1 t2); inv H;
Qed.

Lemma type_expr_incr: forall te a t e e',
  type_expr e a t = OK e' -> S.satisf te e' -> S.satisf
Proof.
  induction a; simpl; intros until e'; intros T SAT; try
- destruct (type_unop u) as [targ1 tres]; monadInv T; ea
- destruct (type_binop b) as [[targ1 targ2] tres]; monad
Qed.
Global Hint Resolve type_expr_incr: ty.
```

CompCert: a Formally Verified Optimizing Compiler
X. Leroy, S. Blazy, D. Kästner, B. Schommer, M. Pister, C. Ferdinand
In *Embedded Real Time Software and Systems*, 2016.

Dynamic V&V Techniques

Acceptance Testing

- Determine whether a system satisfies the *acceptance criteria* and enable the user to determine whether to accept the system.
- A concrete example: *user stories* in *Behavior-driven Development* (BDD)

Title: Returns and exchanges go to inventory.

As a store owner,
I want to add items back to inventory when they are returned or exchanged,
so that I can sell them again.

Scenario 1: Items returned for refund should be added to inventory.
Given that a customer previously bought a black sweater from me **and** I have three black sweaters in inventory,
when they return the black sweater for a refund,
then I should have four black sweaters in inventory.

Scenario 2: Exchanged items should be returned to inventory.
Given that a customer previously bought a blue garment from me **and** I have two blue **and** three garments in inventory
when they exchange the blue garment for a black garment,
then I should have three blue garments in inventory **and** two black garments in inventory.

Alpha, Beta, and Release Candidates

Exploit the expertise of (expert) users/clients

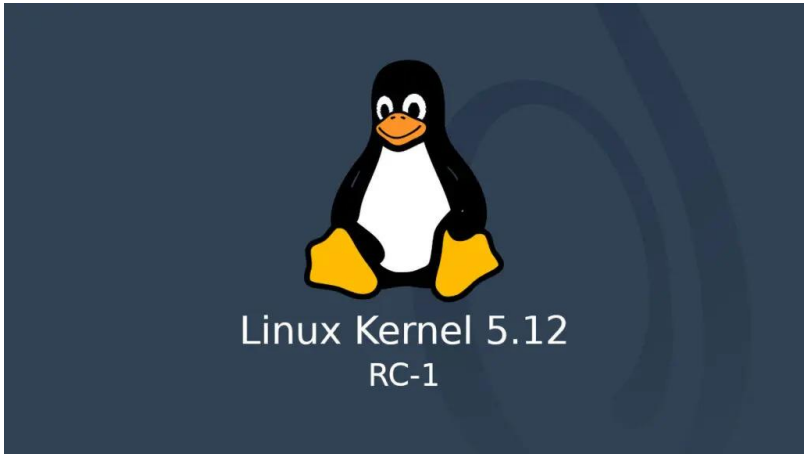
From: Linus Torvalds
To: Linux Kernel Mailing List
Subject: Linux 5.19-rc8
Date: Sun, 24 Jul 2022

As already mentioned last week, this release is one of those "extra week of rc" ones, and here we are, with release candidate #8.

[...]

We'll let this simmer for another week, and **please do give it another round of testing to make this last week count**, ok?

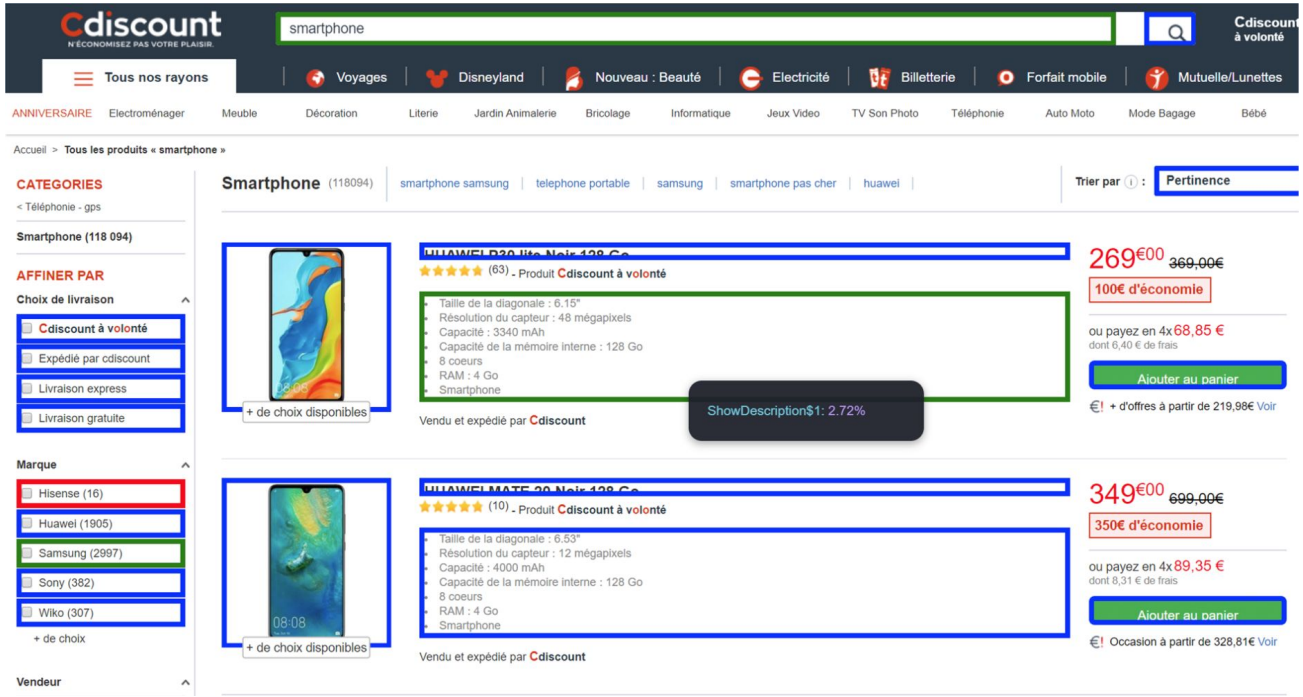
Linus



This merge window, we had a very innocuous code cleanup and simplification that raised no red flags at all, but had a subtle and **very nasty bug** in it: **swap files stopped working right**. And they stopped working in a particularly bad way: the offset of the start of the swap file was lost.

Exploratory Testing

- Send testers on the live system!



Fostering the Diversity of Exploratory Testing in Web Applications
Julien Leveau, Xavier Blanc, Laurent Réveillère, Jean-Rémy Falleri, Romain Rouvoy
In *Software Testing, Verification and Reliability*, 2022.

The Exhaustivity Problem

The Exhaustivity Problem

```
double divide(int numerator, int denominator)
```

The Exhaustivity Problem

2^{32} 2^{32}

```
double divide(int numerator, int denominator)
```

The Exhaustivity Problem

2^{32} 2^{32}

```
double divide(int numerator, int denominator)
```

2^{64} microseconds per test = 548k years!

The Exhaustivity Problem

2^{32} 2^{32}

```
double divide(int numerator, int denominator)
```

2^{64} microseconds per test = 548k years!

- Are some input values more interesting than others?
  - If we have a test for `divide(10, 2)`, do we need a test for `divide(10, 3)`?
  - What about `divide(0, 5)`? `divide(5, 0)`?
  - What about `divide(-1, 1)`? `divide(5, INT_MAX_VALUE)`?
- Gut feeling vs experience vs <u>systematic</u>
  - Equivalence partitioning, boundary value analysis, pairwise testing, etc.
- Accept that we may not be able to find every fault ⟶ adequacy/coverage criteria
- Faults happen more in some places than others

"Program testing can be used to show the *presence* of bugs, but never to show their *absence*!"

Edsger W. Dijkstra, Notes on Structured Programming (EWD249), 1970

EOF