CS453: Fundamentals of Testing

Shin Yoo

Why are we here again?

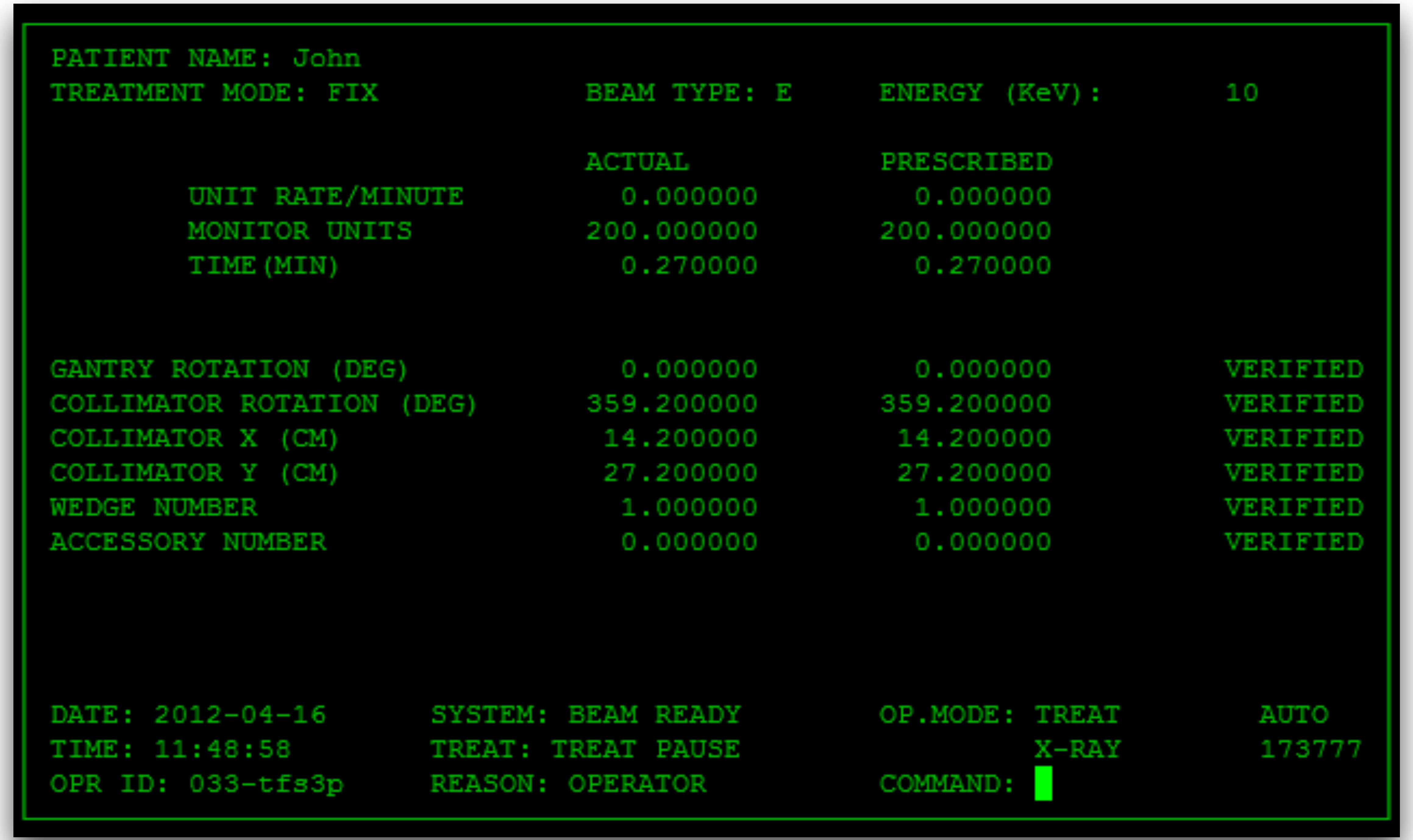
- To graduate, yes.
- To find bugs….?
- What happens if we do not find bugs…?
    - Here come the textbook examples…

Ariane 5

- Rocket developed by European Space Agency
- Exploded 40 seconds after launch, resulting in a loss of about 600M Euros
- Integer overflow!
- http://www.cas.mcmaster.ca/~baber/TechnicalReports/Ariane5/Ariane5.htm

THERAC-25

- Radiation therapy machine, developed by Atomic Energy of Canada, Limited
- Replaced hardware safety lock with a flawed software logic
- Exposed multiple patients to 100 times stronger X-ray, resulting in fatality and injuries



Swedish Stock Market

- A software bug resulted in incorrect buy 131 times of the country's entire GDP
- http://www.businessweek.com/articles/2012-11-29/software-bug-made-swedish-exchange-go-bork-bork-bork

Toyota Recall

- 625,000 Prius cars recalled in 2015 due to software glitches in the braking system
- https://www.bbc.com/news/business-33532673

Boeing 737 Max

- Software fault believed to be the root of the problem that grounded Boeing’s 737 Max in 2019
- Multiple glitches have been reported since
- https://www.theverge.com/2020/2/6/21126364/boeing-737-max-software-glitch-flaw-problem

…and every other software bug you experienced

- KLMS? :)
- Apps on your mobile phones?
- PintOS…?

Software testing: an investigation conducted to provide stakeholders with information about the **quality** of the product or service under test.

Quality?
Magic Moments?

Types of Quality: Dependability

- You should be able to depend on a piece of software. For this, the software has to be correct, reliable, safe, and robust.
- Correctness: with respect to a well formed formal specification, the software should be correct
    - This usually requires proofs, which are hard for any non-trivial systems

Types of Quality: Dependability

- Reliability: it is not sufficient to be correct every now and then - the software should have a high probability of being correct for period of time
  - We usually assume some usage profile (e.g. reliable when there are more than 100,000 users online)
  - Reliability is usually argued statistically, because it is not possible to anticipate all possible scenarios

Types of Quality: Dependability

- Safety: there should be no risk of any hazard (loss of life or property)
- Robustness: software should remain (reasonably) dependable even if the surrounding environment changes or degrades

Types of Quality: Performance

- Apart from functional correctness, software should also satisfy some performances related expectations
  - Execution time, network throughput, memory usage, number of concurrent users…
  - Hard to thoroughly test for, because performance is heavily affected by execution environment

Types of Quality: Usability

- Do users find the software easy enough to use?
  - This is hard to test in a lab setting. Usability testing usually involves focus groups, beta-testing, A/B testing, etc.

Types of Quality: Ethics?

- Fairness Testing: a growing subfield of software testing, the aim of which is to systematically evaluate how fair the System Under Test (SUT) is
  - Typically applied to AI/ML based systems
  - [1] S. Galhotra, Y. Brun, and A. Meliou. Fairness testing: testing software for discrimination. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, aug 2017.
  - [2] S. Udeshi, P. Arora, and S. Chattopadhyay. Automated directed fairness testing. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. ACM, sep 2018.
  - [3] P. Zhang, J. Wang, J. Sun, G. Dong, X. Wang, X. Wang, J. S. Dong, and T. Dai. White-box fairness testing through adversarial sampling. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. ACM, jun 2020.

Dimension for Automation

- Certain types of quality is easier to automatically test than others
  - Relatively easier and widely studied: dependability, reliability…
  - Relatively harder and more cutting edge: usability, non-functional performance, security…

Faults, Error, Failure

- The purpose of testing is to eradicate all of these.
- But how are they different from each other?

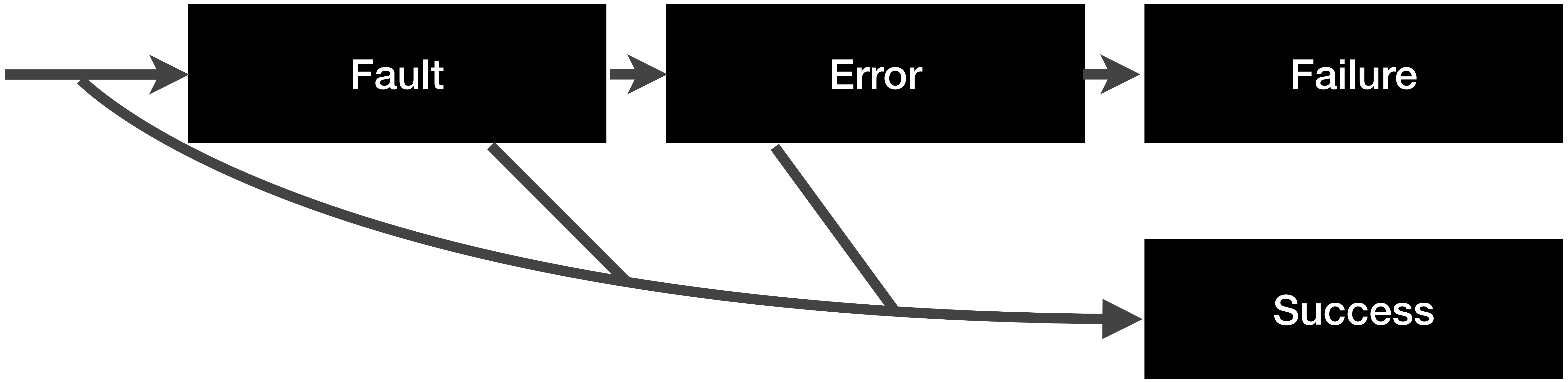
Terminology

- Fault : an anomaly in the source code of a program that may lead to an error
- Error: the runtime effect of executing a fault, which may result in a failure
- Failure: the manifestation of an error external to the program

Dynamic vs. Static

- Note that both error and failure are runtime events.
- Testing is a form of **dynamic analysis** - we *execute* the program to see if it behaves correctly
- To check the correctness without executing the program is **static analysis** - you will see this in the latter half of this course

Fault vs. Error vs. Failure



from IEEE Standard 729-1983, *IEEE Standard Glossary of Software Engineering Terminology*

https://ieeexplore.ieee.org/document/7435207

Fault vs. Error vs. Failure

- No error, no failure
  - The loop is never executed, the loop variable is never incremented

```
void rotateLeft (int* rgInt, int size)
{
  int i;
  for (i = 0; i < size; i++)
  {
    rgInt[i] = rgInt[i+1];
  }
}
```

C program taking an array of integers and 'rotating' the values one position to the left, with wraparound.

```
Test Input #1
input: rgInt [], size 0
output: rgInt []
```

Fault vs. Error vs. Failure

- Error, but no failure.
  - Error: The loop accesses memory outside the array
  - But the output array is coincidentally correct

```
void rotateLeft (int* rgInt, int size)
{
  int i;
  for (i = 0; i < size; i++)
  {
    rgInt[i] = rgInt[i+1];
  }
}
```

```
Test Input #2
input: rgInt [0, 1] 0, size 2
output: rgInt [1, 0] 0
```

Fault vs. Error vs. Failure

- Failure!

```
void rotateLeft (int* rgInt, int size)
{
  int i;
  for (i = 0; i < size; i++)
  {
    rgInt[i] = rgInt[i+1];
  }
}
```

```
Test case 3
input: rgInt [0,1] 66, size 2
output: rgInt [1,66] 66
```

Fault vs. Error vs. Failure

- But what exactly is *the fault*?
  - The loop indexes `rgInt` outside its bounds
  - The loop never moves `rgInt[0]` to another position
  - The loop never saves `rgInt[0]` for later wraparound
- There are also *multiple* possible fixes
  - The fix actually determines what *the fault* was!

```
void rotateLeft (int* rgInt, int size)
{
  int i;
  for (i = 0; i < size; i++)
  {
    rgInt[i] = rgInt[i+1];
  }
}
```

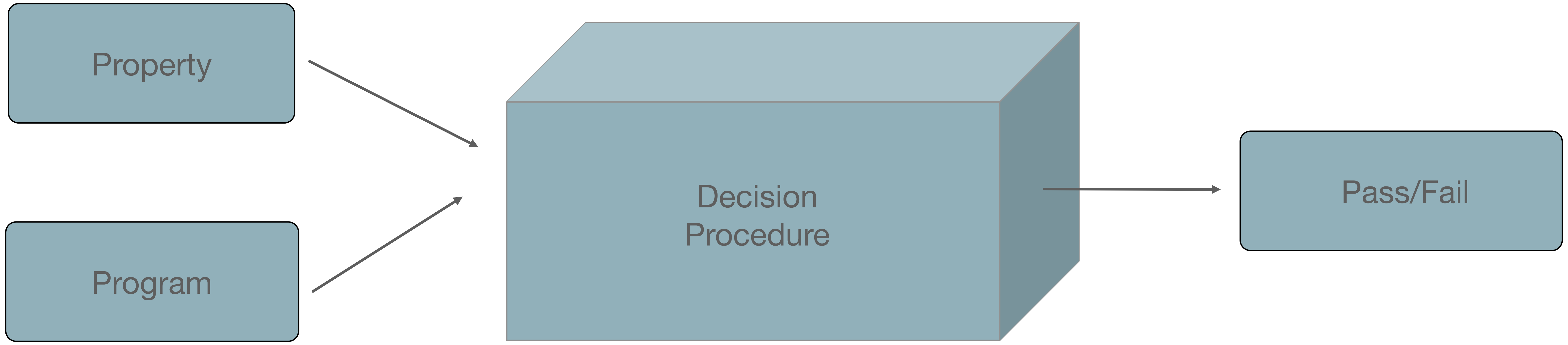
- You can pass all test cases and still be incorrect.
- You can execute the faulty statement and still pass (“coincidental correctness”).

More Terminology

- Test Input: a set of input values that are used to execute the given program
- Test Oracle: a mechanism for determining whether the actual behaviour of a test input execution matches the expected behaviour
  - In general, a very difficult and labour-intensive problem
- Test Case: Test Input + Test Oracle
- Test Suite: a collection of test cases
- Test Effectiveness: the extent to which testing reveals faults or achieves other objectives
- Testing vs. Debugging: testing reveals faults, while debugging is used to remove a fault

Why is testing hard?

You Can't ~~Always~~ Ever Get What You Want



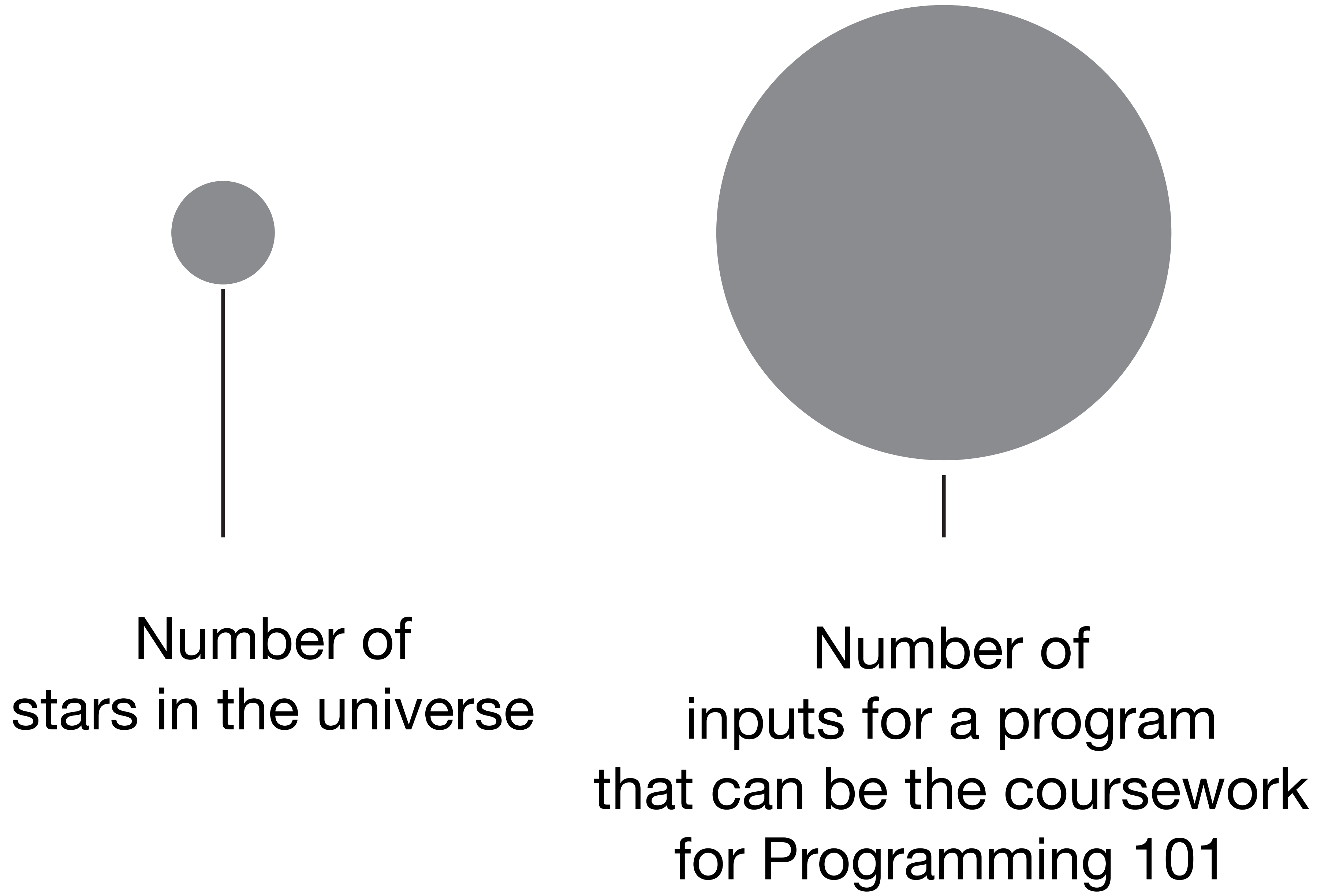
- Correctness properties are *undecidable:* Having one decision procedure is out of question.
- The Halting Problem can be embedded in almost *every* property of interest!

Exhaustive Testing

- Can we test each and every program with all possible inputs, and guarantee that it is correct every time? Surely then it IS correct.
- In theory, yes - this is the fool-proof, simplest method… or is it?
- Consider the triangle program
  - Takes three 32bit integers, tells you whether they can form three sides of a triangle, and which type if they do.
  - How many possible inputs are there?

Exhaustive Testing

- 32bit integers: between -2^{31} and $2^{31}-1$, there are 4,294,967,295 numbers
- The program takes three: all possible combination is close to 8^{28}
- Approximated number of stars in the known universe is 10^{24}
- Not. Enough. Time. In. The. Whole. World.



A Famous (or Infamous) Quote

- “Testing can only prove the presence of bugs, not their absence.” — Edsger W. Dijkstra
- Is it true?

Dijkstra vs. Testing

```
int testMe (int x, int y)
{
  return x / y;
}
```

What is the "bug"?

```
Test Input #1
(x, y) = (2, 1)
```

```
Test Input #2
(x, y) = (1, 2)
```

```
Test Input #3
(x, y) = (1, 0)
```

A Famous (or Infamous) Quote

- “Testing can only prove the presence of bugs, not their absence.” — Edsger W. Dijkstra
- An oft-repeated disparagement of testing that ignored the many problems of his favoured alternative (formal proofs of correctness)
- But the essence of the quote is true:
  - Testing allows only a sampling of an enormously large program input space
  - The difficulty lies in how to come up with effective sampling

We still keep on testing…

- Imagine you have two choices when boarding a flight
  - Flight control for airplane A has never been proven to work, but it has been tested with a *finite* number of *test flights*
  - Flight control for airplane B has never been executed in test flight, but it has been *statically verified to be correct*
- My personal belief is that testing (as in trial and error) is still fundamentally of the most basic human nature
- Certain things - for example, energy consumption - can only be tested and not verified

Test Oracle

- In the example, we immediately know something is wrong when we set y to 0: all computers will treat division by zero as an error
- What about those faults that forces the program to produce answers that are only slightly wrong?
- For every test input, we need to have an “oracle” - something that will tell us whether the corresponding output is correct or not
- Implicit oracles: system crash, unintended infinite loop, division by zero, etc - can only detect a small subset of faults!

Bug Free Software?

- However I'm constantly hearing business people spout off with "It's understood that software will be bug free, and if it's not all bugs should be fixed for free". I typically respond with "No, we'll fix any bugs found in the UAT period of (x) weeks" where x is defined by contract. This leads to a lot of arguments, and loss of work to people who are perfectly willing to promise the impossible.
    - http://stackoverflow.com/questions/2426623/exhaustive-testing-and-the-cost-of-bug-free

Automated Testing

- Sometimes requires purely analytic approaches, investigating the structure that is the source code.

```
[sensor.h]
...
int64_t read();
...

[controller.h]
...
void set_input1(int16_t arg);
...

[controller.c]
#include "sensor.h"
#include "controller.h"
...
value = read();
set_input(value);
...
```

Good Testing

- Sometimes requires purely analytic approaches, investigating the structure that is the source code.

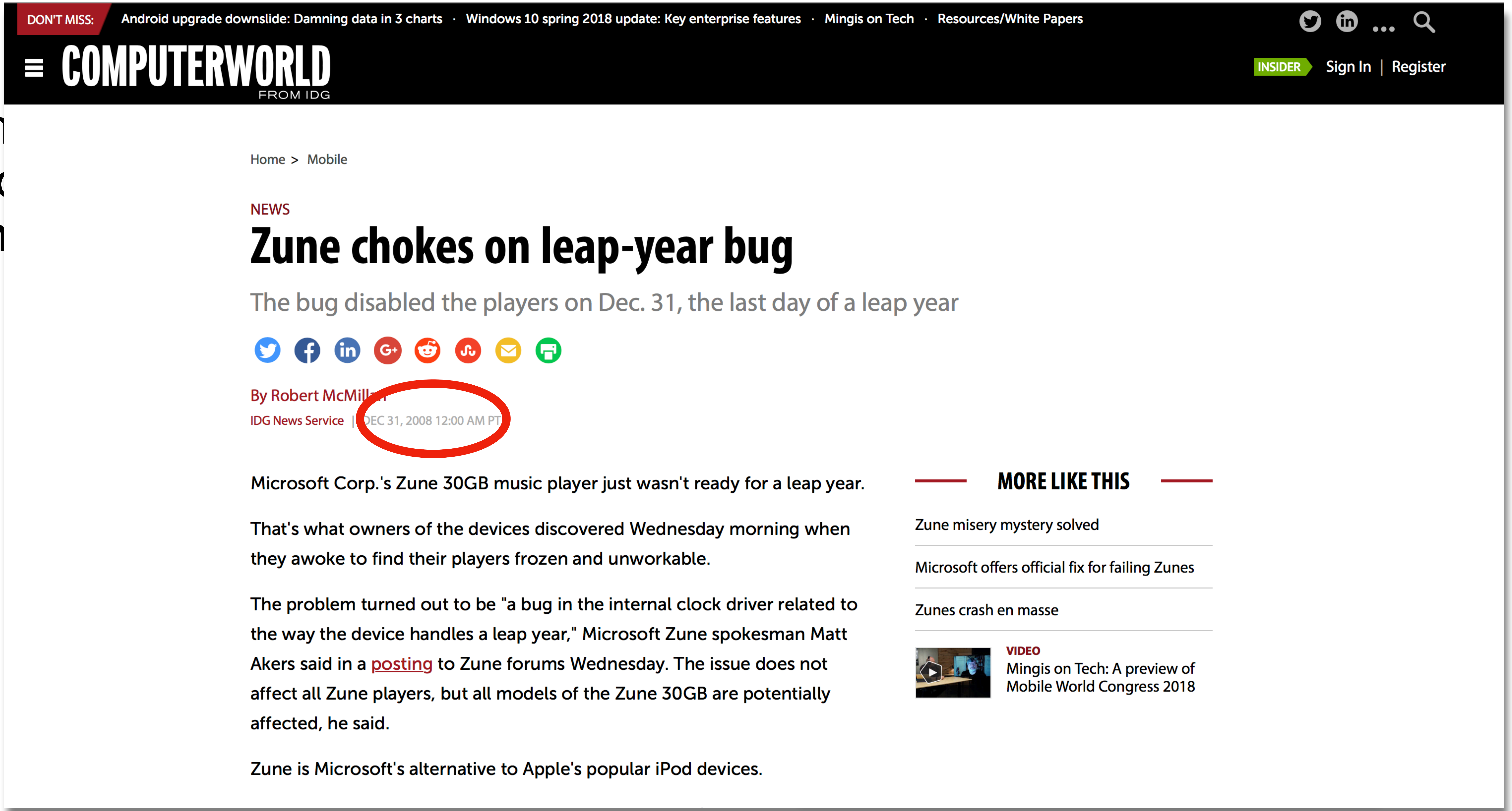
```
[sensor.h]
...
int64_t read();
...

[controller.h]
...
void set_input1(int16_t arg);
...

[controller.c]
#include "sensor.h"
#include "controller.h"
...
value = read();
set_input(value);
...
```

Good Testing

- Som thor dom inqu



DON'T MISS: Android upgrade downslide: Damning data in 3 charts · Windows 10 spring 2018 update: Key enterprise features · Mingis on Tech · Resources/White Papers

COMPUTERWORLD FROM IDG

INSIDER Sign In | Register

Home > Mobile

NEWS

Zune chokes on leap-year bug

The bug disabled the players on Dec. 31, the last day of a leap year

By Robert McMillan
IDG News Service | DEC 31, 2008 12:00 AM PT

Microsoft Corp.'s Zune 30GB music player just wasn't ready for a leap year.

That's what owners of the devices discovered Wednesday morning when they awoke to find their players frozen and unworkable.

The problem turned out to be "a bug in the internal clock driver related to the way the device handles a leap year," Microsoft Zune spokesman Matt Akers said in a posting to Zune forums Wednesday. The issue does not affect all Zune players, but all models of the Zune 30GB are potentially affected, he said.

Zune is Microsoft's alternative to Apple's popular iPod devices.

MORE LIKE THIS

Zune misery mystery solved

Microsoft offers official fix for failing Zunes

Zunes crash en masse

VIDEO
Mingis on Tech: A preview of Mobile World Congress 2018

Good Testing

- Sometimes requires either a thorough knowledge of the domain, or a very imaginative, inquisitive, and creative mind.

```
/* date correction for september 1752 */
if(special_days)
    result = "Day did not exist.";
else
    if (leapflag && is_september && day>13)
        result = dayName((addMonths(month,year)+
                    (--day)+firstJanuary(year)+10)%7);
    else ...
```

Why study/research testing?

- All the utilitarian reasons (correctness, safety, usability, fairness…)

- But it can also be very fun detective work, finding the balance between:
    - What the developer intended (no one knows…)
    - What the source code says
    - What the comment / documentation says
    - What other code typically does in a similar situation
    - What can be predicted statically
    - What can be observed dynamically

Testing Techniques

- There is no fixed recipe that works always.
- There is currently no technique that can understand the expected semantic of the system - we need both automation and human brain.
- You need to understand the pros and cons of each technique so that you can apply.
- There are two major classes of testing techniques:
  - Black-box: tester does not look at the code
  - White-box : tester does look at the code

Random Testing

- Can be both black-box or white box
- Test inputs are selected randomly
- Pros:
  - Very easy to implement, can find real faults
- Cons:
  - Can take very long to achieve anything, can be very dumb

Combinatorial Testing

- Black-box technique
- Tester only knows the input specification of the program.
- How do you approach testing systematically?
- The same principle applies to testing a single program in many different environments.

Structural Testing

- White-box technique.
- The adequacy of testing is measured in terms of structural units of the program source code (e.g. lines, branches, etc).
- Necessary but not sufficient (yet still not easy to achieve).

Mutation Testing

- White-box technique.
- A subclass of structural testing: we artificially inject faults and see if our testing can detect them.
- Huge potential but not without challenges.

Regression Testing

- Can be both black- and white-box.
- A type of testing that is performed to gain confidence that the recent modifications did not break the existing functionalities.
- Increasingly important as the development cycle gets shorter; organisations spend huge amount of resources.