With Unit Tests Like These, Who Needs Static Typing?

Julian Orbach

Contents

- Claim:
  Static typing bad; unit-testing good.

- Experiment
  - Method, Results, Conclusions
  - Look at the Feedback

But first... some definitions.

Type System Definitions

Dynamic type system
A type system that annotates each **value** with a type at **run-time** (RT), to detect any type errors.

Examples: Python, PHP, Perl

Static type system
A type system that annotates each **variable** with a type at **compile-time**, to detect any type errors.

Examples: C, Java, Haskell

Type inference
A static type system that infers the type of an value by its usage.

Examples: Haskell, Miranda

Unit Test Definitions

Unit Test (UT);
code to test individual parts of a piece of software.

Statement coverage;
UT goal to ensure that every single line of the code under test is executed at least once.

Evan Farrer's Master Thesis

1) Static typing rejects some valid programs.
2) Static typing is insufficient for detecting bugs; unit testing is required.
3) Once you have UT, static type checking is redundant

Conclusion: Static typing is “harmful”, insufficient and redundant. UTs are necessary and find all the static typing bugs & more.

???????

Step 1: Choose Source Language

- The language should be dynamically typed
- The language should have support for and a culture of unit testing
- The language should have a large corpus of open source software for studying
- The language should be well known and considered a good language among dynamic typing proponents

Python (Yay!)

Step 2: Choose Target Language

- The language should be statically typed
- The language should execute on the same platform as Python
- The language should be strongly typed
- The language should be considered a good language among static typing proponents

Haskell (Oooh!?)

Step 3: Choose random projects

- **The Python NMEA Toolkit**
  - API/Drivers for some sort of GPS.
- **MIDIUtil**
  - OO Abstraction of MIDI (music) files.
- **PyFontInfo**
  - Library to extract header info from font files.
- **GrapeFruit**
  - Library to convert colour spaces (sRGB, CMYK, etc)

Step 4: Translate

- Painstakingly translated each library from Python to Haskell.
  - Not redesign.
  - Line-by-line port, to the degree possible.
- Noted where not possible.
  - struct.pack and struct.unpack
    - use format strings to determine type-conversions used.

Step 5: Record Problems

- Type Errors
  - found by static type checking, not found by UT suite.
  - Subdivision: What if stmt coverage had been used? Would have been found/mightn't have been found.
- Redundant Effort
  - UTs that only tested types.
  - Run-time (RT) type tests

Results

Project	Type Errors		Redundant Check		Examples
	Would	Might not	UT	RT	
NMEA Toolkit	1	8	2	0	3x malformed input, 6x incorrect usage of API
MIDIUtil	1	1	0	1	__eq__() raises attribute error Notes of -ve duration.
PyFontInfo	0?	6	1	2	Ex: Private methods are incorrectly exposed to the public; might not be properly initialised.
GrapeFruit	0	0	1	1	

Limited Conclusions

Before (H_0):

- Static typing could reject valid code, insufficient and redundant.
- UTs are necessary and find all the static typing bugs & more.

After:

- Static typing could reject valid code (but didn't), insufficient, but not completely redundant.
- UTs are necessary and find more than static typing, but not all the typing bugs.
- You also have to add more UTs if you don't have static typing.
- True for typical and for 100% statement coverage.

Some Feedback

- UTs are very useful; static typing doesn't replace UTs.
  - Yes
- NMEA Toolkit's UTs were simplistic!
  - Okay, but.
- Static typing still rejects valid programs.
  - Yes, but...
- The small no. of bugs found means static typing isn't worth it.
  - Maybe, but...
- Dynamically typed code is smaller, thus easier to maintain.
  - Not necessarily. Separate experiment required.

Conclusion

- Limited value
  - win arguments about language design
  - not helpful for day-to-day code.
- Knew:
  - Static typing doesn't obviate unit testing.
- Learnt:
  - Unit testing doesn't obviate static typing, in practice, even with ambitious targets.
- "I argue for unit tests because they lead to a better design and a better API. I would still promote unit testing even if they never caught any bugs." - Evan Farrer

Refs/Q&A

Evan Farrer's MSc Thesis:

- “A QUANTITATIVE ANALYSIS OF WHETHER UNIT TESTING OBVIATES STATIC TYPE CHECKING FOR ERROR DETECTION”, 2011, California State University.

Evan Farrer's blog article:

- http://evanfarrer.blogspot.com.au/2012/06/unit-testing-isnt-enough-you-need.html

These slides (incl. notes)

- http://somethinkodd.com/sypy/farrer.pdf

Backup Slides

Type Inference Example

```
def addition(left, right):
     return left + right
x = "Foo"
y = 2
print addition(x, y)
```

Condition/Decision Coverage

```
def foo(x):
    if (x > 10 or
            (x > 0 and
                1/"a")):
        print True
    else:
        print False
```

```
>>> foo(2000)
True

>>> foo(-2000)
False
```

100% Statement coverage achieved! You can stop writing unit-tests!

```
>>> foo(5)
Traceback (most recent call last):
      …
TypeError: unsupported operand
type(s) for /: 'int' and 'str'
```

Point-Counter Point

- Types and tests? One and the same!
  - A static type system provides a universal qualification that a property is satisfied across a program.
  - A test provides existential qualification that a certain property holds in a certain situation.
  - Thus: Tests are weaker, but test a much broader class of problems.
- Response: Qualitatively different!
  - Static type system can identify the absence of certain unwanted behaviors *regardless of whether the developer is thinking about them.*
  - Tests make assertions about things a developer remembers.

With Unit Tests Like These, Who Needs Static Typing?

Julian Orbach

Contents

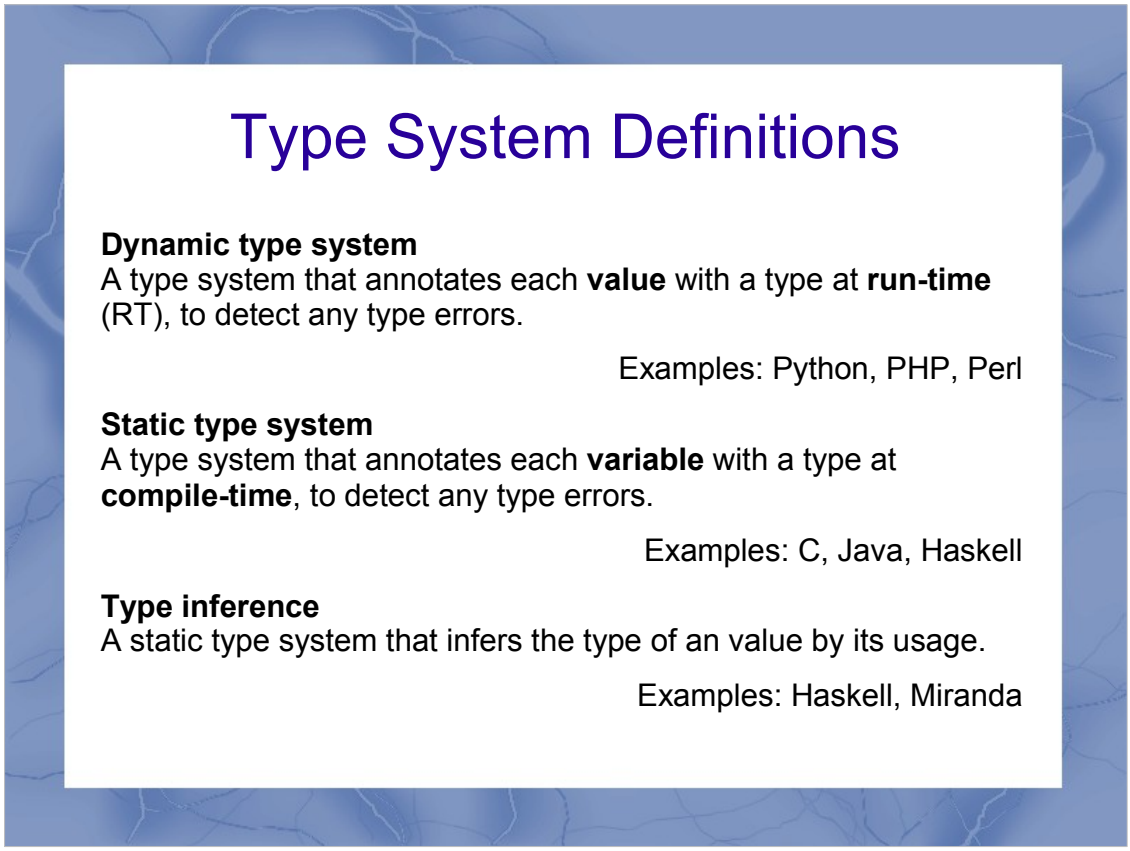
- Claim:
  Static typing bad; unit-testing good.
- Experiment
  - Method, Results, Conclusions
  - Look at the Feedback

But first... some definitions.

I want to talk about an interesting paper I read, which looks at Python, static typing and unit-testing and whether static typing is a bad idea.

This isn't a very practical talk; you won't be rushing off to change your implementation - but it introduces some evidence to some common discussions around programming languages.

I'm going to be using some words a lot that people tend to use loosely, so first I need to eat my vegetables by defining some terms:



Just because it is statically typed, doesn't mean it is a good type system! Simple example: C considers a character to be an integer. Characters are NOT integers. They never were integers. They never will be integers.

Check if people understand type-inference, and jump to backup slide if necessary.

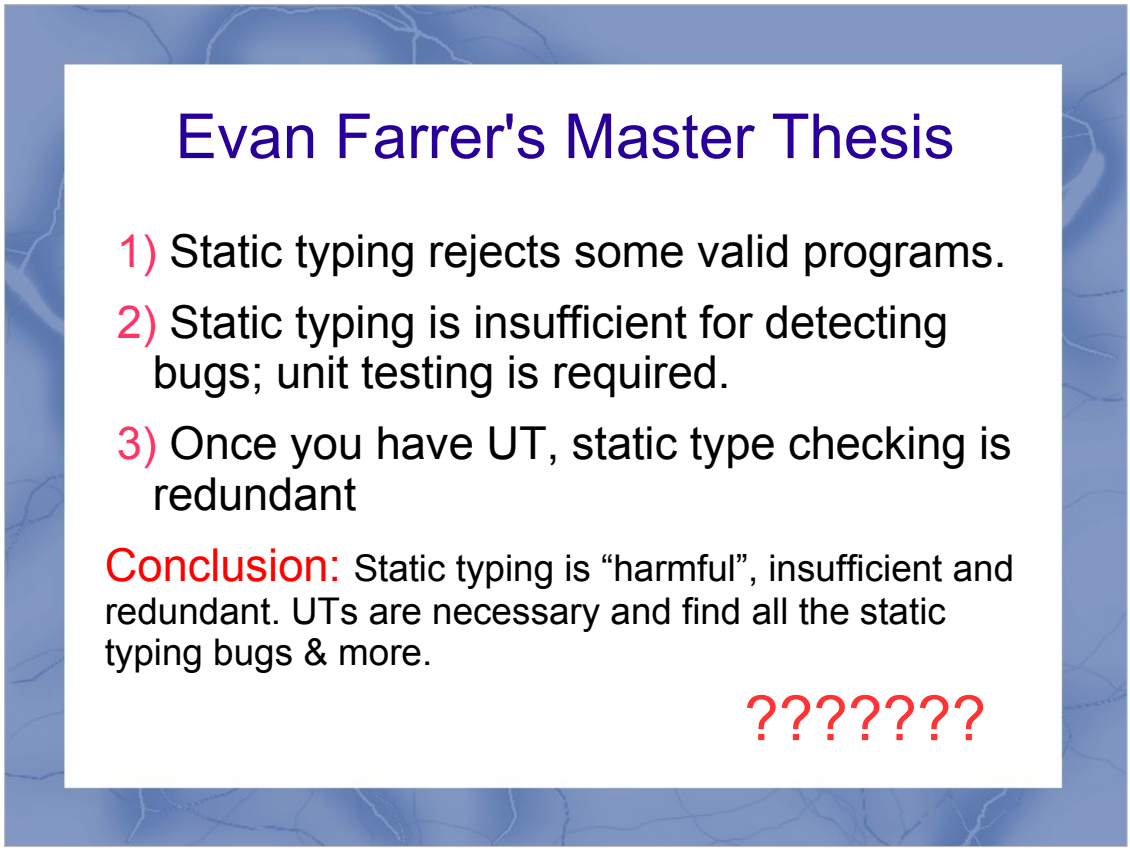
Unit Test Definitions

Unit Test (UT);
code to test individual parts of a piece of software.

Statement coverage;
UT goal to ensure that every single line of the code under test is executed at least once.

Re: 100% stmt coverage: We saw last time, that is very high standard, in practice. However, there are higher standards:

Backup slide available for statement coverage if required.



Evan Farrer is a bloke who wrote an interesting Master's Thesis in 2011, based on an experiment he performed.

Farrer says this is a frequent argument:

1. I can confirm! Church Numerals in Miranda.
2. I can confirm! Ignoring formal methods
3. Conclusion: Yay, Python

It would be nice if some people in the room agree with this argument, for two reasons: It will make this talk more relevant, and it will demonstrate that Farrer isn't arguing against a strawman.

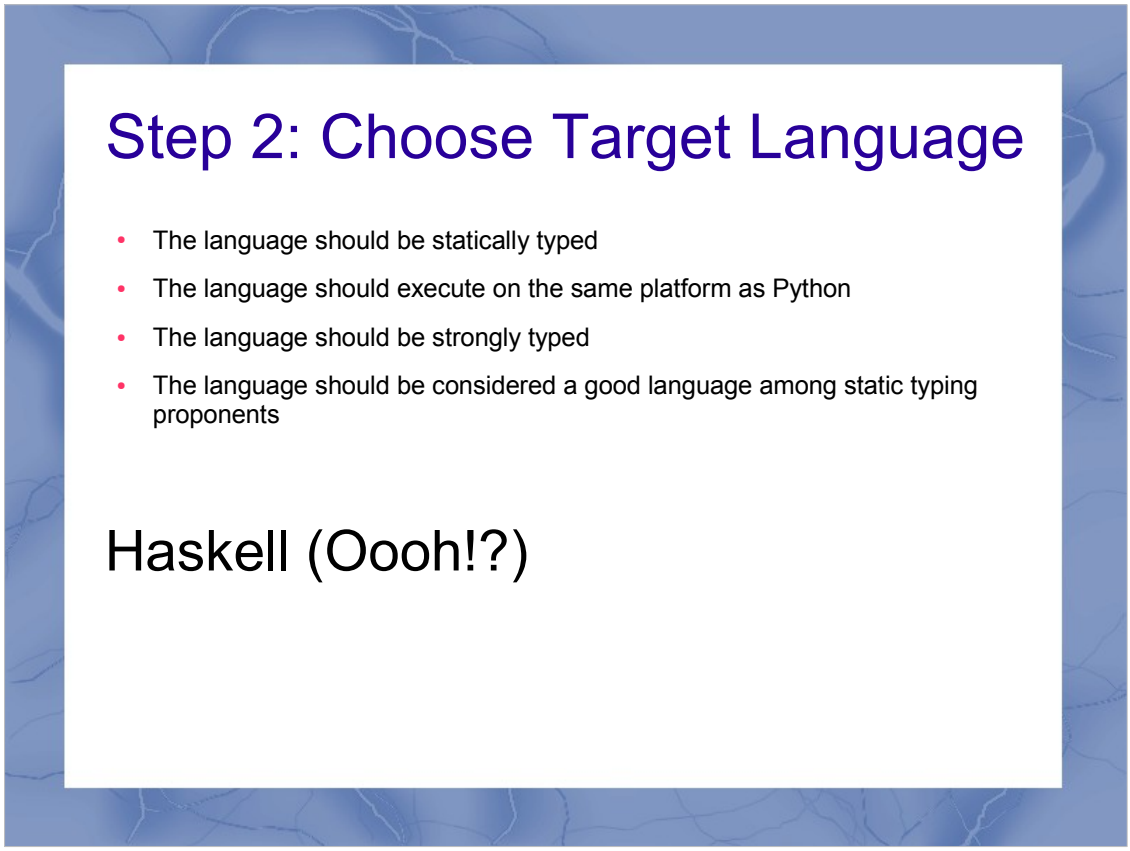
Click to add question marks; (make it clear this is NOT what Farrer believed, but what he wanted to test – in fact, this is the Null Hypothesis – what his experiment set out to disprove.)
Farrer complained that these claims weren't based on good evidence, so he set out to conduct an experiment.

What happens if you add static typing to an existing, dynamically typed program?

Step 1: Choose Source Language

- The language should be dynamically typed
- The language should have support for and a culture of unit testing
- The language should have a large corpus of open source software for studying
- The language should be well known and considered a good language among dynamic typing proponents

Python (Yay!)



He DIDN'T choose C. He DIDN'T choose Java. He chose a type-inferencing language.

Step 3: Choose random projects

- **The Python NMEA Toolkit**
  - API/Drivers for some sort of GPS.
- **MIDIUtil**
  - OO Abstraction of MIDI (music) files.
- **PyFontInfo**
  - Library to extract header info from font files.
- **GrapeFruit**
  - Library to convert colour spaces (sRGB, CMYK, etc)

Not terribly familiar with any of them, but they have unit-tests, and they seem to be in domains where unit-testing would fit well. Not large, but non-trivial. Seems a fair set for this experiment.

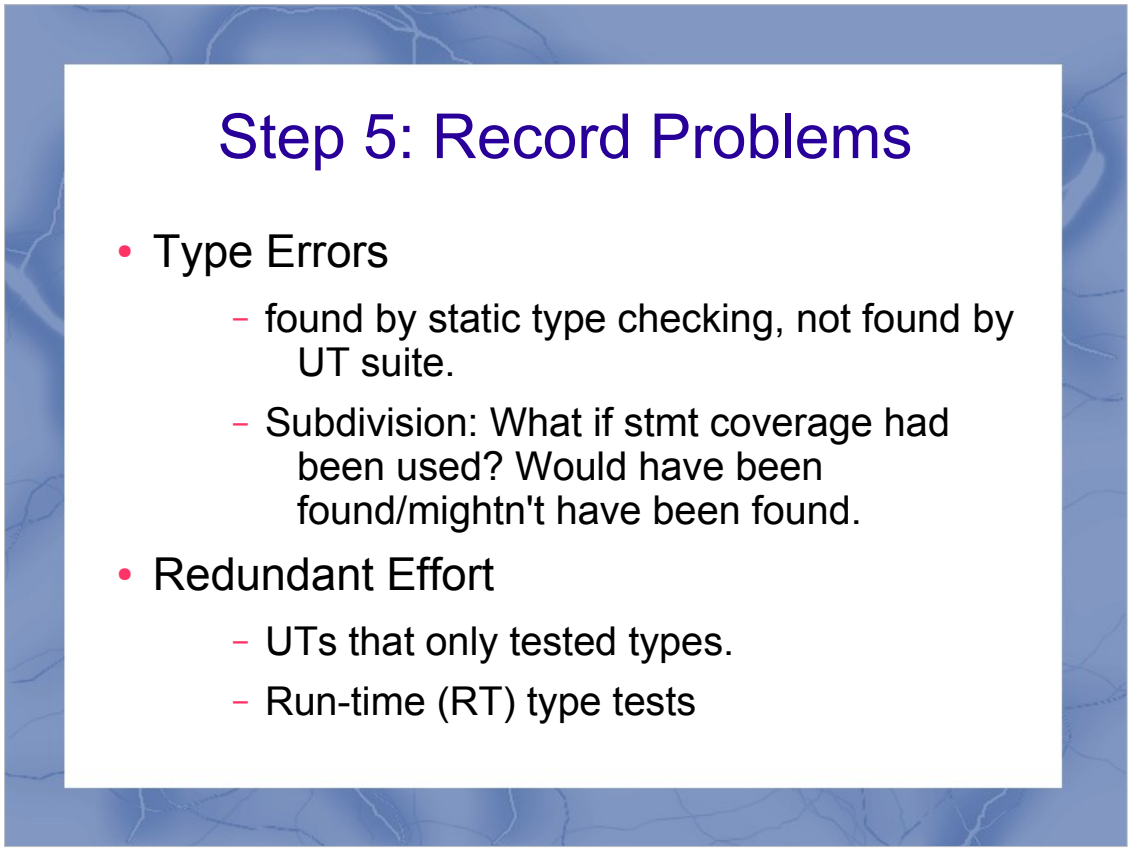
Use this slide to support that Farrer chose them randomly, if challenged.

Step 4: Translate

- Painstakingly translated each library from Python to Haskell.
  - Not redesign.
  - Line-by-line port, to the degree possible.
- Noted where not possible.
  - struct.pack and struct.unpack
    - use format strings to determine type-conversions used.

The Hard Work!

Struct: That would be implemented a different way in Haskell – not harder, not easier, just different.



He looked for the existence of:

(1) Bugs found by static type-checking that were not found by the existing unit tests. If he found any, he categorised it:
(1a) If the unit-tests had been 100% statement coverage, they definitely would have found this.
Examples of these would show that, in practice, static-typing is more rigorous than unit-testing for those bugs.
(1b) If the unit-tests had been 100% statement coverage, they still might not have found this.

Examples of these would show that, even in theory, static-typing is not redundant with even very strong unit-testing. (Not infinitely strong!)

(2) Unit-tests that *only* tested typing
i.e. that need not have been written in the first place, if a static-typing language had been used.

Examples of these would demonstrate that any extra development costs of static typing need to be offset against the costs of the extra unit-tests required for dynamic typing.

(3) Explicit run-time errors that check for type-safety - errors that could never occur in a statically typed language

Results

Project	Type Errors		Redundant Check		Examples
	Would	Might not	UT	RT	
NMEA Toolkit	1	8	2	0	3x malformed input, 6x incorrect usage of API
MIDIUtil	1	1	0	1	__eq__() raises attribute error Notes of -ve duration.
PyFontInfo	0?	6	1	2	Ex: Private methods are incorrectly exposed to the public; might not be properly initialised.
GrapeFruit	0	0	1	1	

Sure enough, he found examples of all of these. Perhaps not a *lot*, but some.

PyFontInfo: The thesis claimed several of these 6 wouldn't have been found, but it wasn't clear whether ALL of the 6 wouldn't've been found.

While we can argue over the severity (ability to making a library crash with bad input is a potential security bug!)

Some of these bugs found would have a severity of "Trivial" if they had been reported, IMHO (NOT the view of Farrer who declined to rate them.) But some are serious.

Limited Conclusions

Before (H_0):

- Static typing could reject valid code, insufficient and redundant.
- UTs are necessary and find all the static typing bugs & more.

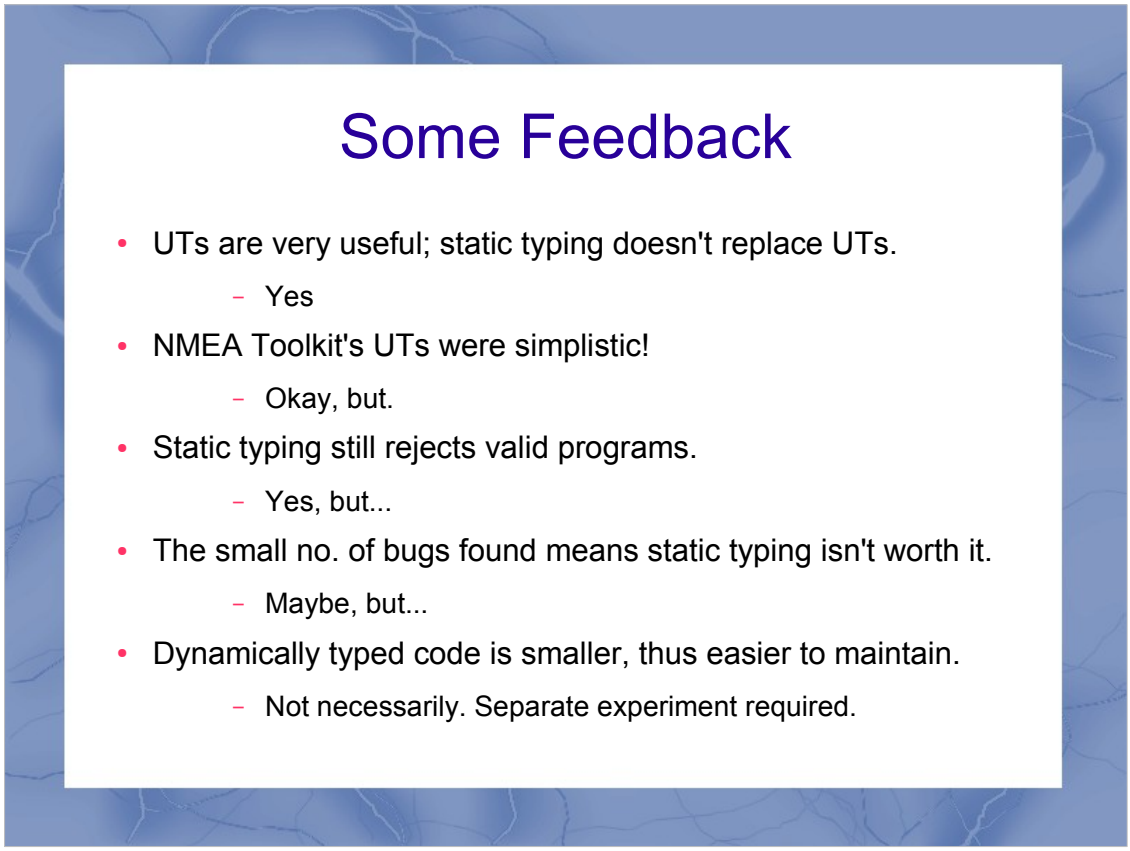
After:

- Static typing could reject valid code (but didn't), insufficient, but not completely redundant.
- UTs are necessary and find more than static typing, but not all the typing bugs.
- You also have to add more UTs if you don't have static typing.
- True for typical and for 100% statement coverage.

Conclusions are fairly limited; this is NOT a powerful result, but kind of interesting.

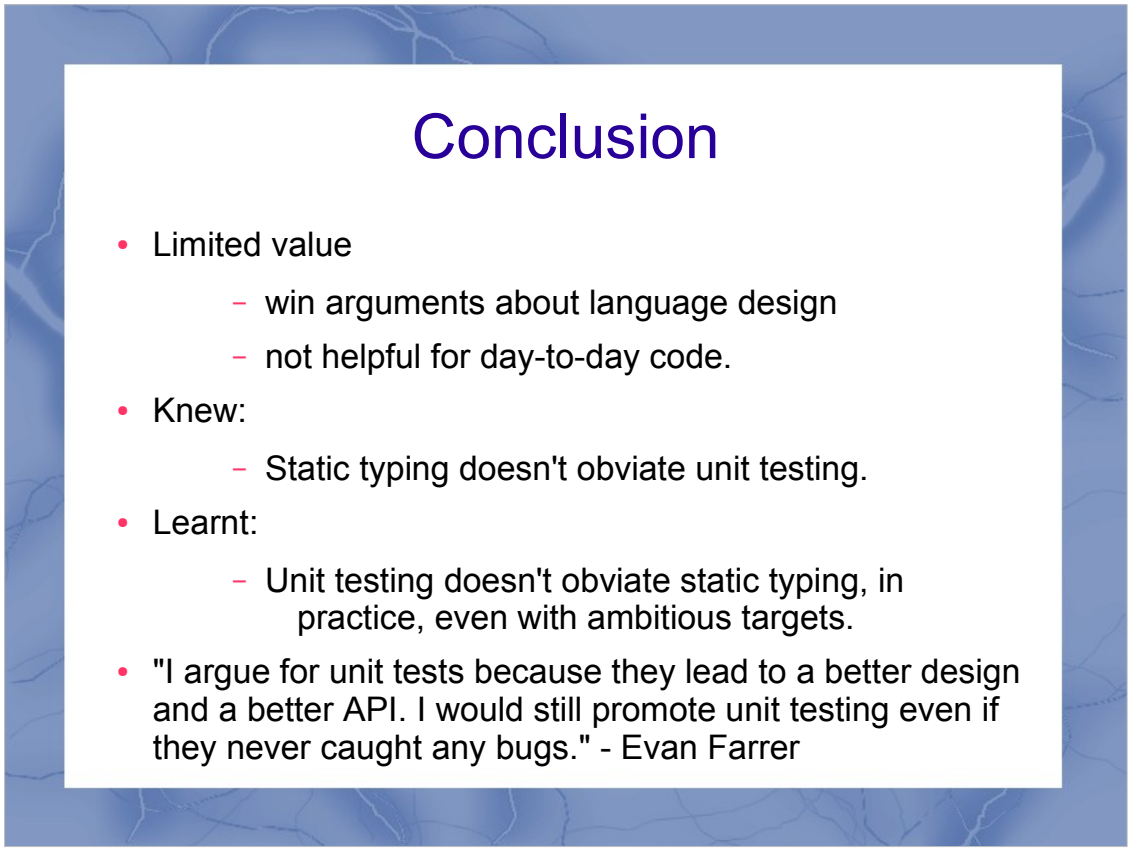
Reiterate “harmful” only means that some valid programs are rejected. And it didn't occur in practice for these 4 projects.

Branch coverage? Farrer thought even that wouldn't help – private email



The comments on the blog article systematically attacked it for saying things it didn't say! [Probably what made the article more interesting to me was how the Python fan-bois poorly fought against the conclusions.

1) Sure, the unit-tests probably found lots of bugs... but no-one said otherwise.
2) Yes, but not any of the examples here, (although sample space was small). So it is a real problem, but maybe not a common one.
3) If you are thinking of Java and all its elaborate type-declarations, sure! But there are better systems (*cough* Haskell *cough*), that don't add much. Would need a separate experiment.
4) Okay, but it is a library used in practice. And 100% statement coverage still wouldn't have worked.
5) Was trying to disprove an absolute.
   - “This doesn't mean that no one should ever use a dynamically typed programming language, it just means that there is a tradeoff. If you are writing software for the mars rovers perhaps finding one or two more bugs out of 1000 is worth it, for your family blog perhaps not.” - Evan Farrer



There's not a strong message here; there's nothing here to improve your code. It is just some evidence against some people who take an overly strong position against static-typing.

I want to give the last word to Evan Farrer, who is strongly pro unit-tests, even if they don't catch all the bugs...

Refs/Q&A

Evan Farrer's MSc Thesis:

- “A QUANTITATIVE ANALYSIS OF WHETHER UNIT TESTING OBVIATES STATIC TYPE CHECKING FOR ERROR DETECTION”, 2011, California State University.

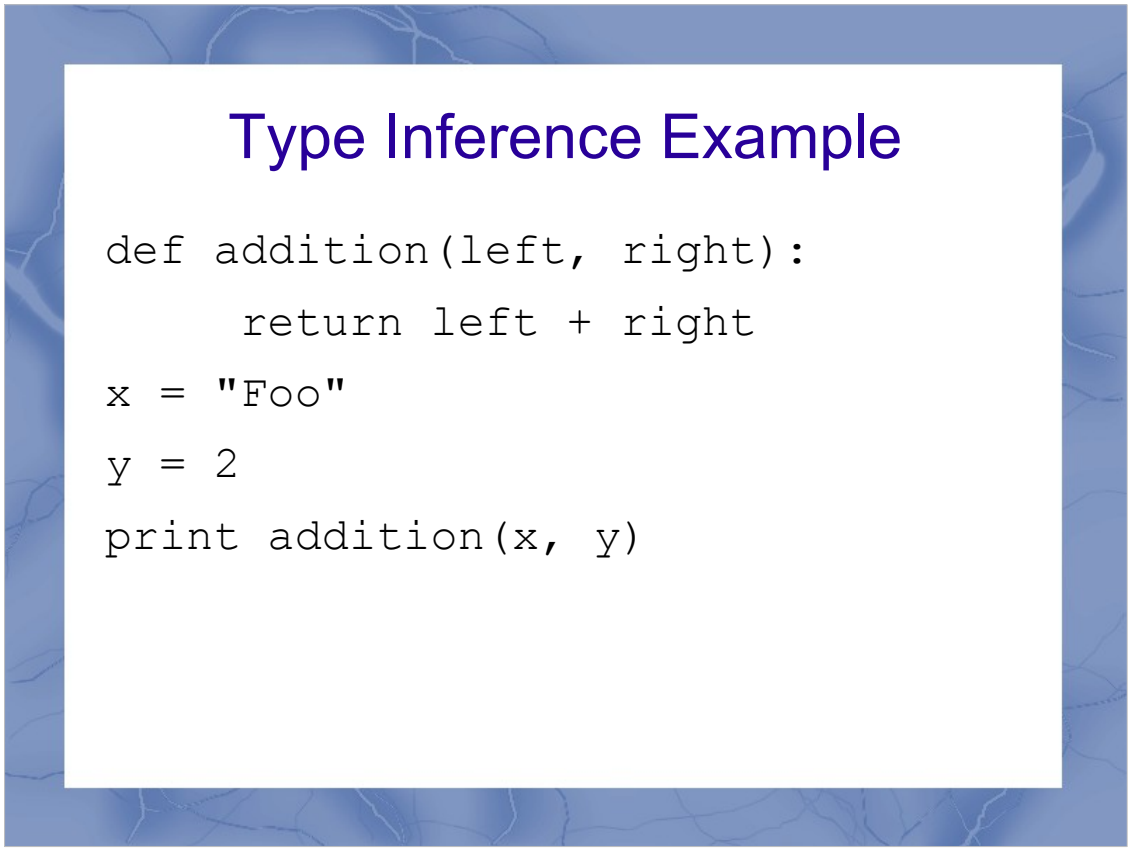
Evan Farrer's blog article:

- http://evanfarrer.blogspot.com.au/2012/06/unit-testing-isnt-enough-you-need.html

These slides (incl. notes)

- http://somethinkodd.com/sypy/farrer.pdf

Backup Slides

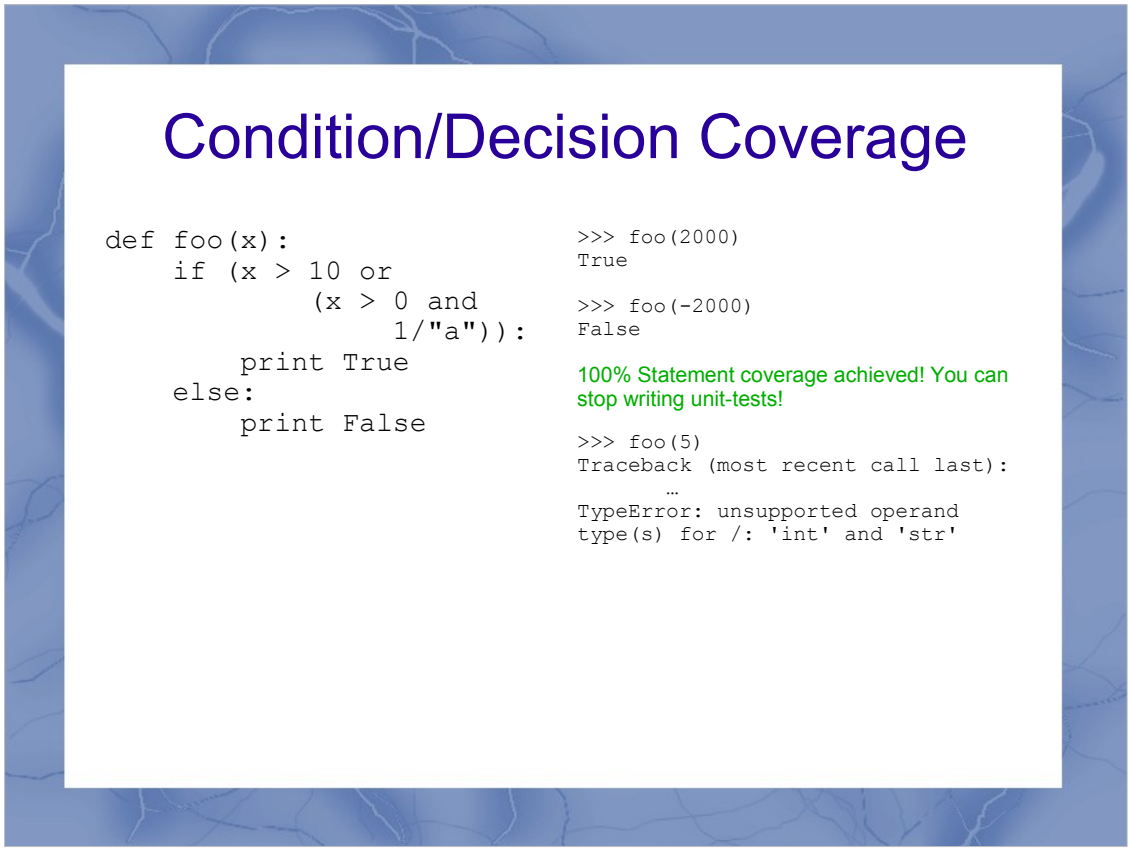


This Python code is going to fail. You know that. I know that. And we know that even though the type of x, y, left and right have never been specified. We inferred the value of x is a string. We inferred the value of y is an integer. We inferred left and right will be string and integer respectively. We know that you can't add strings and integers.

A type-inference typing system will report this as a type-error, even without it being annotated.

My experience, in practice, with Miranda, which :
The type error may flow through many function calls, and the type error becomes this global property. For maintainability, it makes sense to include optional annotations about what is expected as a parameter. It means when there is a type error, the compiler can point more closely to where it all went wrong.

I haven't programmed in Miranda for 17 years. I miss Miranda's type-inference almost every day that I use Python.



Wikipedia:

There are a number of coverage criteria, the main ones being:[3]

Function coverage - Has each function (or subroutine) in the program been called?

Statement coverage - Has each node in the program been executed?

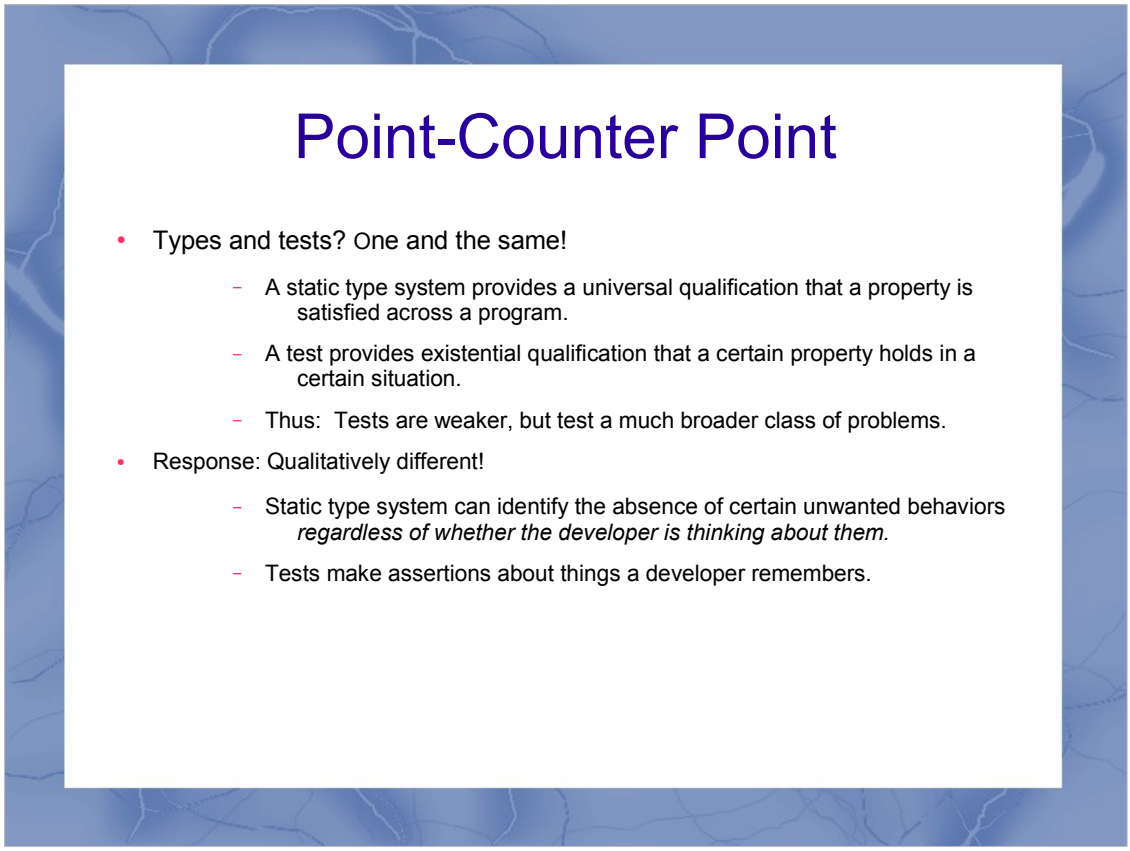
Decision coverage (not the same as branch coverage.[4]) - Has every edge in the program been executed? For instance, have the requirements of each branch of each control structure (such as in IF and CASE statements) been met as well as not met?

Condition coverage (or predicate coverage) - Has each boolean sub-expression evaluated both to true and false? This does not necessarily imply decision coverage.

Condition/decision coverage - Both decision and condition coverage should be satisfied.

Parameter Value Coverage - In a method taking parameters, has all the common values for such parameter been considered?

Input coverage – generally intractable.



Some feedback on the blog.

2) Lightly paraphrased.
Response from random, NOT Farrer. Accepted the same premises, but drew a different conclusion.

Some skipped points:

* Someone argued that bugs found by static typing aren't important.
I can give simple counter-arguments. Ny code has fallen down in production because it executed an untested line where I misspelled an identifier (especially logging code inside an exception handler... Grrrr!)

* Farrer says he also saw logical bugs, but ignored them as out of scope for this argument.