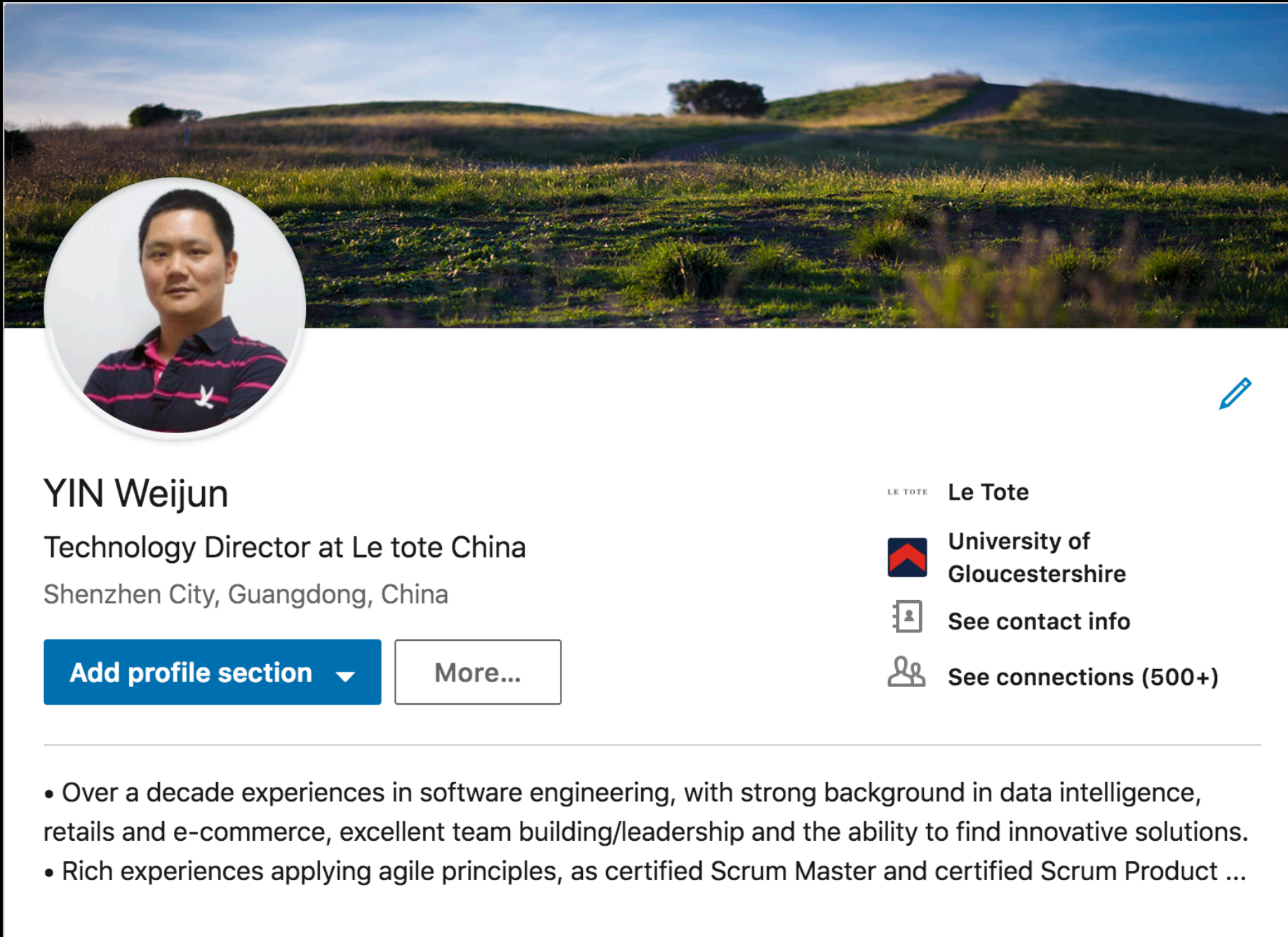


:Elixir

|> Property-based Testing




|> Day One



The image shows a LinkedIn profile for YIN Weijun. At the top is a banner image of a grassy field with hills in the background. Below the banner is a circular profile picture of a man with short dark hair wearing a dark polo shirt with pink stripes. To the right of the profile picture is a blue pencil icon. Below the profile picture, the name "YIN Weijun" is displayed, followed by the title "Technology Director at Le tote China" and the location "Shenzhen City, Guangdong, China". There are two buttons: a blue "Add profile section" button with a dropdown arrow, and a white "More..." button with a black border. To the right of these buttons, there are three items: "Le Tote" with a small logo, "University of Gloucestershire" with its logo, and "See contact info" with a contact card icon. Below these, there is a link "See connections (500+)" with a group of people icon. At the bottom, there is a section with two bullet points: "• Over a decade experiences in software engineering, with strong background in data intelligence, retails and e-commerce, excellent team building/leadership and the ability to find innovative solutions." and "• Rich experiences applying agile principles, as certified Scrum Master and certified Scrum Product ...".

YIN Weijun
Technology Director at Le tote China
Shenzhen City, Guangdong, China

[Add profile section](#) [More...](#)

LE TOTE **Le Tote**
 **University of Gloucestershire**
 **See contact info**
 **See connections (500+)**

- Over a decade experiences in software engineering, with strong background in data intelligence, retails and e-commerce, excellent team building/leadership and the ability to find innovative solutions.
- Rich experiences applying agile principles, as certified Scrum Master and certified Scrum Product ...

尹伟君

linkedin.com/in/yin-weijun-93a08816/

Agenda

- Introduction to Property-based Testing
- Unit + Property Testing
- Modules: StreamData, ExUnitProperties
- How to find properties
- Upcoming with StreamData

QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs

Koen Claessen
Chalmers University of Technology
koen@cs.chalmers.se

John Hughes
Chalmers University of Technology
rjmh@cs.chalmers.se

ABSTRACT

QuickCheck is a tool which aids the Haskell programmer in formulating and testing properties of programs. Properties are described as Haskell functions, and can be automatically tested on random input, but it is also possible to define custom test data generators. We present a number of case studies, in which the tool was successfully used, and also point out some pitfalls to avoid. Random testing is especially suitable for functional programs because properties can be stated at a fine grain. When a function is built from separately tested components, then random testing suffices to obtain good coverage of the definition under test.

1. INTRODUCTION

Testing is by far the most commonly used approach to ensuring software quality. It is also very labour intensive, accounting for up to 50% of the cost of software development. Despite anecdotal evidence that functional programs require somewhat less testing (‘Once it type-checks, it usually works’), in practice it is still a major part of functional program development.

monad are hard to test), and so testing can be done at a fine grain.

A testing tool must be able to determine whether a test is passed or failed; the human tester must supply an automatically checkable criterion of doing so. We have chosen to use formal specifications for this purpose. We have designed a simple domain-specific language of *testable specifications* which the tester uses to define expected properties of the functions under test. QuickCheck then checks that the properties hold in a large number of cases. The specification language is embedded in Haskell using the class system. Properties are normally written in the same module as the functions they test, where they serve also as checkable documentation of the behaviour of the code.

A testing tool must also be able to generate test cases automatically. We have chosen the simplest method, random testing [11], which competes surprisingly favourably with systematic methods in practice. However, it is meaningless to talk about random testing without discussing the distribution of test data. Random testing is most effective when the distribution of test data follows that of actual data, but when testing reusable code units as opposed to whole sys-



elixir

[HOME](#) [INSTALL](#) [GUIDES](#) [LEARNING](#) [DOCS](#)

StreamData: Property-based testing and data generation for Elixir

October 31, 2017 · by Andrea Leopardi · in [Releases](#)

In this blog post, we'll talk about property-based testing and sample data generation. We'll cover what these are, why we want them in Elixir, and what are are plans for the future. If you want to use the features discussed here or you want to read more formal documentation, head over to [stream_data](#), which is a library that currently provides both features (albeit in beta form) and which is where we are focusing our efforts.

Sample data generation

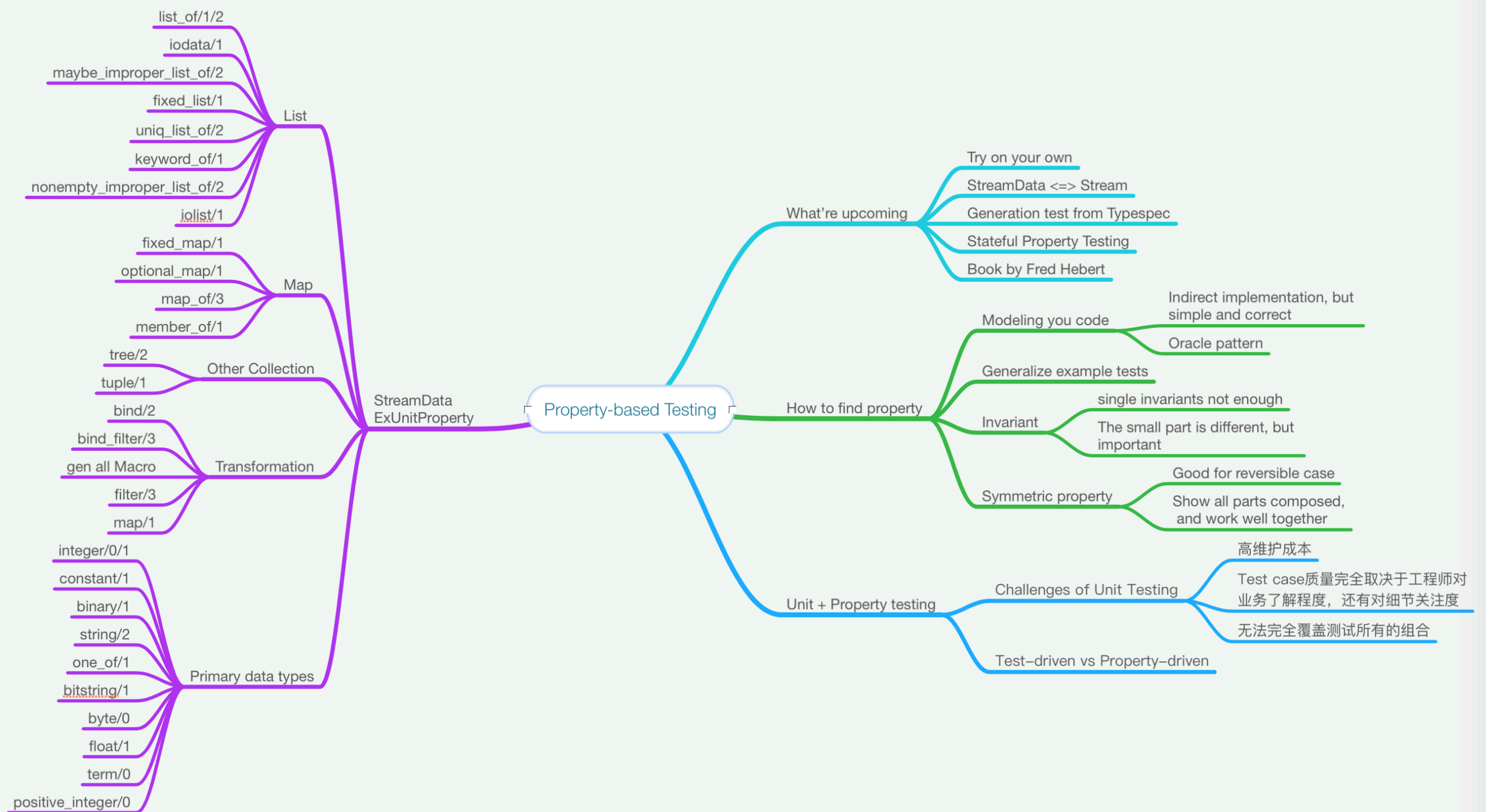
The core of the [stream_data](#) library is `StreamData`: this module provides all the functionalities related to generating sample data of many kinds. It includes both data generators for data types (like integers or booleans) as well as tools to combine other generators (such as `one_of(list_of_generators)`).

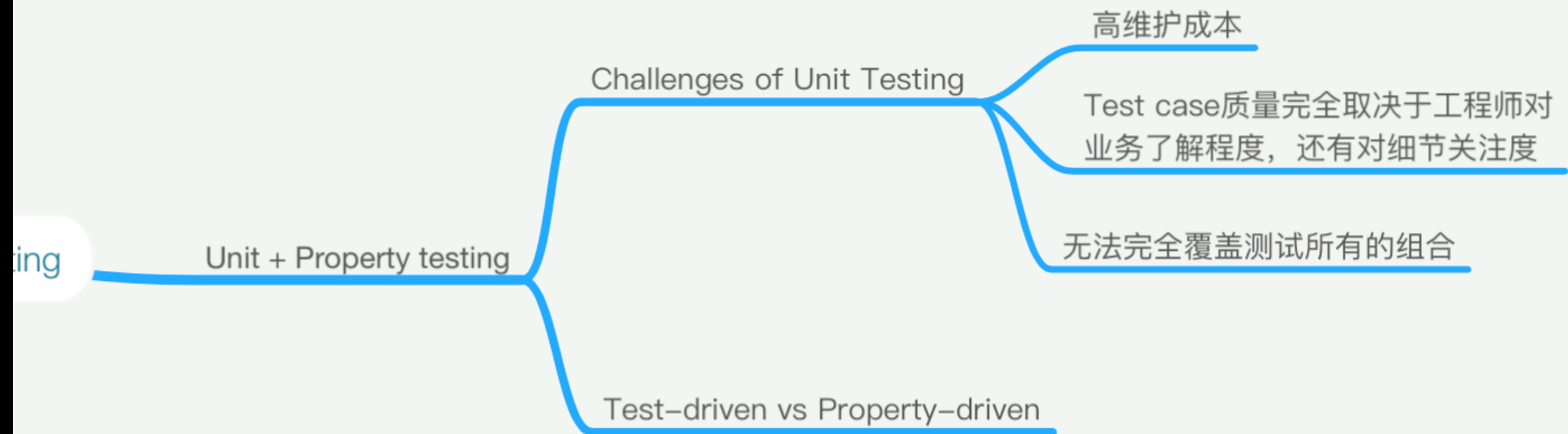
- <https://elixir-lang.org/blog/2017/10/31/stream-data-property-based-testing-and-data-generation-for-elixir/>

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

译文：程序测试可用于显示错误的存在，但永远不会显示它们的缺席！

反思Unit Testing





Unit + Property

“Property-based testing is not a replacement for all testing, just a new tool that can often improve results.”

合作共赢

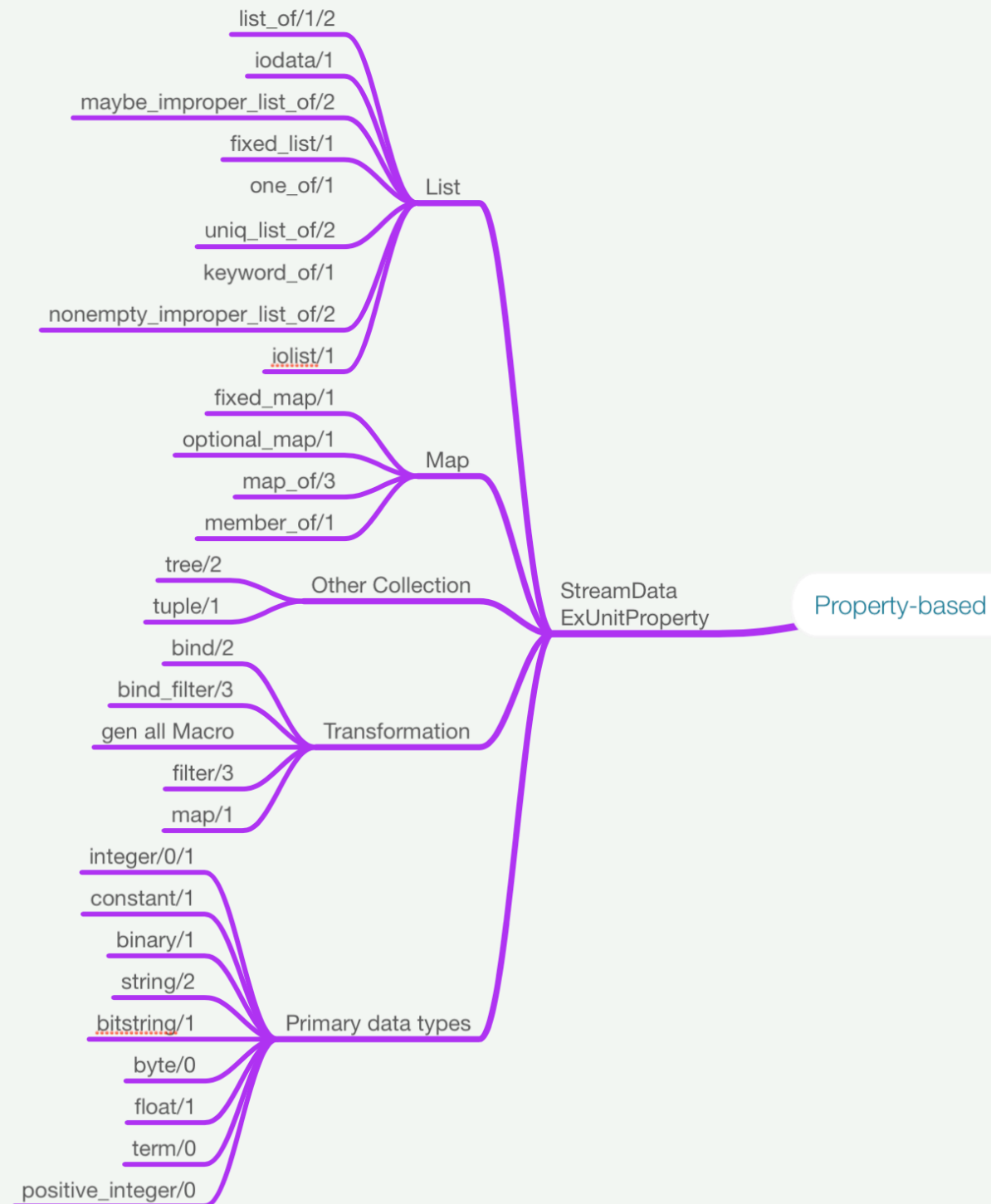
Challenges of Unit Testing

- 大量的测试代码有待维护
- Test case质量完全取决于工程师的业务了解和细节关注度
- 无法完全覆盖所有的测试组合

Test-driven vs. Property-driven

Test-driven	Property-driven
helps design code to have it conform to expectations and demands. design matches your expectation.	force the exploration of the program's behaviour to see what it can or cannot do. explore the program's behaviour.
Easy to write regular test case	guiding principles and expectations and use these directly as a test
Focus on details	Focus on high-level
Good on regression	Good in finding bugs

StreamData & ExUnitProperties



https://github.com/whatyouhide/stream_data

```
defmodule RevertTest do
  use ExUnit.Case
  use ExUnitProperties
```

```
  test "Test reverse" do
    assert reverse([]) == []
    assert reverse([1]) == [1]
    assert reverse([1, 2, 3]) == [3, 2, 1]
    assert reverse(0..10) == Enum.to_list(10..0)
  end
```

```
  property "Reverse a reverse doesn't change" do
    check all list <- list_of(integer()) do
      # IO.inspect(list)
      assert reverse(reverse(list)) == list
    end
  end
```

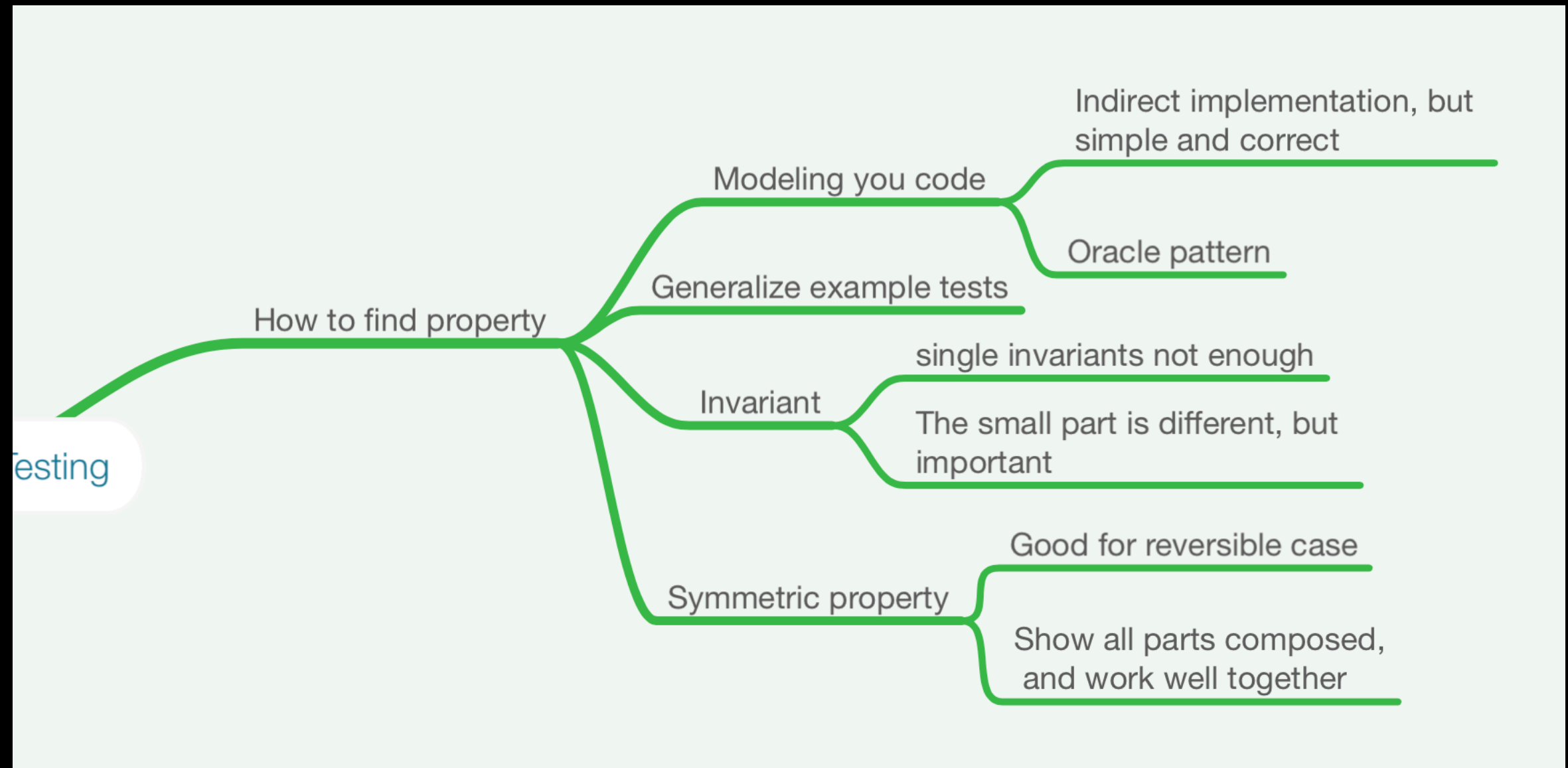
```
property "finds biggest element" do
  check all int_list <- nonempty(list_of(integer())) do
    assert Pbt.biggest(int_list) == model_biggest(int_list)
  end
end
```

```
def model_biggest(list) do
  list |> Enum.sort() |> List.last()
end
```



```
property "picks the last number" do
  check all {list, known_last} <- tuple_of_listNinteger() do
    known_list = list ++ [known_last]
    assert known_last == List.last(known_list)
  end
end
```

```
def tuple_of_listNinteger() do
  tuple({list_of(integer()), integer()})
end
```



How to find a property

老司机教路...

寻找您梦中的Property的一些技巧

- Modeling your codes - simpler implementation, or alternative implementation
- Generalize Example Tests - need more understanding your problem domain
- Invariant - 小, 简单, 恒久不变的
- Symmetric Property

Property-Testing Community

- javascript <http://jsverify.github.io>
- python <https://hypothesis.readthedocs.io>
- haskell <https://hackage.haskell.org/package/QuickCheck>
- go <https://github.com/leanovate/gopter>
- Scala http://www.scalatest.org/user_guide/property_based_testing
- Erlang <http://www.quviq.com/products/erlang-quickcheck>
- Elixir <https://github.com/alfert/propcheck>

Further Reading

[1]: <https://propertesting.com/index.html>

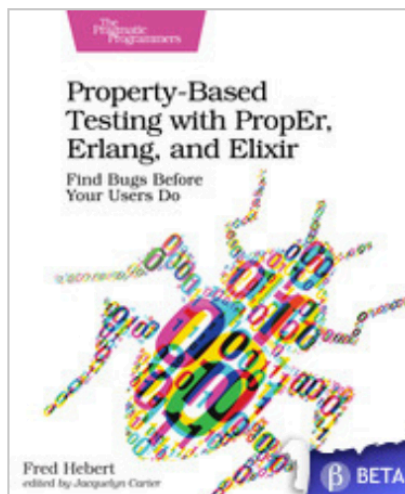
[2]: <https://blog.gopheracademy.com/advent-2017/property-based-testing/>

[3]: <https://people.eecs.berkeley.edu/~alexch/classes/CS294-F2016.html>

[4]: <https://youtu.be/p84DMv8TQuo>

[5]: <https://youtu.be/VhW9D0mbW1o>

[6]: <http://andrealeopardi.com/posts/the-guts-of-a-property-testing-library/>



Property-Based Testing with PropEr, Erlang, and Elixir

Find Bugs Before Your Users Do

by Fred Hebert

Property-based testing helps you create better, more solid tests with little code. By using the PropEr framework in both Erlang and Elixir, this book teaches you how to automatically generate test cases, test stateful programs, and change how you design your software for more principled and reliable approaches. You will be able to better explore the problem space, validate the assumptions you make when coming up with program behavior, and expose unexpected weaknesses in your design. PropEr will even show you how to reproduce the bugs it found. With this book, you will be writing efficient property-based tests in no time.