Introduction to Software Engineering

12. An Empirical Software Engineering Primer (and a bit on type systems)

Mircea F. Lungu

Partially using materials by Massimiliano Di Penta
Almost done…

> Next (week): Project Presentation
  — Public (invite your friends)
  — 7 minutes of presenting the project
> Next (next (week)): Exam Preparation
Dragos: Functional and OO can co-exist
Zeller: Testing can be automated
Dustdar: We must learn how to design systems with humans included
Di Penta: Empirical studies for detecting bad code
Gamma: Monaco is the new editor for Typescript
Empirical Studies
Empirical = Observation | Experimentation
Kinds of empirical studies

> **Quantitative**: to get numerical relations among variables
  - Are programmers more productive with Java than with C#?
  - Are defects correlated with cyclomatic complexity?

> **Qualitative**: to interpret a phenomenon just observing it in its context
  - E.g. by using explanations obtained by interviewing developers
  - I interview developers to know why a given method improves their productivity
  - By observing some software artifacts
Quantitative Studies

Evaluating state of the art and practice
No user involvement
Tool selection and tailoring

Evaluating specific aspects of a technology in a controlled environment
Careful design
Replication

Evaluating the whole technology on a realistic project
Lower level of control than experiments
Examples of Empirical Studies in SE

- API Design at Microsoft
- UML In Practice
- API Deprecation
- Influence of Type Systems
- Pair Programming
- Is Code Duplication Good or Bad?
- How Developers use Reflection
- Comparing Programming Languages
- Which metrics correlate better with perceived complexity?
Case Study: Static vs. Dynamic Typing
Type systems

> Goal: assigning meaning to bits

> Multiple aspects
  — Weak
  — Strong
  — Static
  — Dynamic

> Automate boring checks
Weak Typing

- When one can “coerce” a variable of one type to be used in stead of a variable of another type
- Pointer Arithmetic
- Languages: C
Strong Typing

- A type system which prevents the possibility of unchecked runtime errors
- Languages: Haskell, Java
- Advantages: Tool support
Static Type Checking

- Verifying the type safety of a program based on the text of the program
- Executed by the compiler
- Languages: Java, C++
Dynamic Typing

- Type checks are executed at compile time
- Is not excluded by static typing
- Languages: Smalltalk, Ruby, Python
- Advantages: faster round trip
Dynamic Typing Enables Duck Typing

> Duck.quack()
Specifying types is extra work
Type inference can require less specification

object InferenceTest1 extends Application {
  val x = 1 + 2 * 3         // the type of x is Int
  val y = x.toString()      // the type of y is String
  def succ(x: Int) = x + 1  // method succ returns Int values
}
Anything that tells you about a mistake earlier not only makes things more reliable because you find the bugs, but the time you don't spend hunting bugs is time you can spend doing something else

James Gosling

http://www.artima.com/intv/strongweak.html
Dynamic is Great!

The flexibility of dynamically typed languages makes writing code significantly easier. There are no build time issues at all. Life in a dynamically typed world is fundamentally simpler.

Robert Martin

http://www.artima.com/weblogs/viewpost.jsp?thread=4639
Impact on Development Time

> 49 subjects
> developing a parser
> 27 hours of work time
> Purity language (16 hours training)
> 200 test cases

> static type system has no impact on development time
Intermezzo: Designing Controlled Experiments

- Hypothesis formulation
- Controlling Variables
- Threats to validity
- Replication
Hypothesis formulation

> The experiment aims at rejecting a null hypothesis
> We can reject the null hypothesis $\rightarrow$ we can draw conclusions
> Hypothesis must be specific
Controlling the variables

[Wohlin et al., 2000]
Null Hypothesis H0

- There do not exist trend/patterns in the experimental setting: **the occurred differences are due to chances**
- Example: there is no difference in code comprehension with the new technique and the old one H0 $\mu_{Nold}=\mu_{Nnew}$
Alternative Hypothesis Ha

> In favor of which the null hypothesis is rejected
> Example: the new technique allows a better level of code comprehension than the old one $H_0 \, \mu_{\text{Nold}} < \mu_{\text{Nnew}}$
Important!

- An experiment does not prove any theory, it can only fail to reject an hypothesis.
- The logic of scientific discovery [Popper, 1959]
  - Any statement made in a scientific field is true until anybody can contradict it.
- In practice we could do it after several replications…
Average time is 20% higher in the control group than in the experimental group. Can we conclude that the experimental treatment is better?
Can you eat the cake and have it too?

```typescript
function twoMoreThanYou(calculateANumber: Function): number {
    return calculateANumber(4) + 2;
}

function double(n: number): number {
    return n*2;
}

console.log("TWO MORE", twoMoreThanYou(double))
```

Typescript, Dart add *optional static type annotations*

http://en.wikipedia.org/wiki/TypeScript
Bonus: An Experiment About Comparing Languages

- 80 implementations
- in 7 languages
- task: string manipulation and search in a dictionary

- dynamic languages are more productive
- C/C++ use less memory
- differences between programmers are larger than between languages

An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl
Lena Prochel
Fakultät für Informatik, Universität Karlsruhe
76128 Karlsruhe, Germany
+49(721)608-4566, Fax: +49(721)608-7434
http://www.pci.informatik.uni-karlsruhe.de/prochel/
March 14, 2000

Introduction
When it comes to the pros and cons of various programming languages, programmers and computer scientists often are not highly optimistic. This paper will provide empirical data on the performance of C, C++, Java, Perl, Python, Rexx, and Tcl, by showing the following facts.

Results
The programs were evaluated on four different input sizes: 1000, 2000, 5000, 10000. Each input size was evaluated 100 times. The input size for the evaluation was chosen to be 100 arbitrary strings.

Summary
The difference in run time for the four different input sizes is significant. The difference in run time is larger than the differences between programming languages. The difference in run time is larger than the differences between programming languages. The difference in run time is larger than the differences between programming languages.

Memory Consumption
The memory consumption of the four different input sizes is significant. The memory consumption is larger than the differences between programming languages. The memory consumption is larger than the differences between programming languages. The memory consumption is larger than the differences between programming languages.
What you should know

> What are type systems and what are some of the advantages of the different approaches
> What kind of empirical studies can be run in software engineering
> What is the difference between qualitative and quantitative experiments?
> Can an experiment prove a hypothesis?
Attribution-ShareAlike 3.0

You are free:
- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

**BY:** Attribution. You must attribute the work in the manner specified by the author or licensor.

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

[http://creativecommons.org/licenses/by-sa/3.0/](http://creativecommons.org/licenses/by-sa/3.0/)