UNIVERSITÄT BERN

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

Report from the CHOOSE Forum

- 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

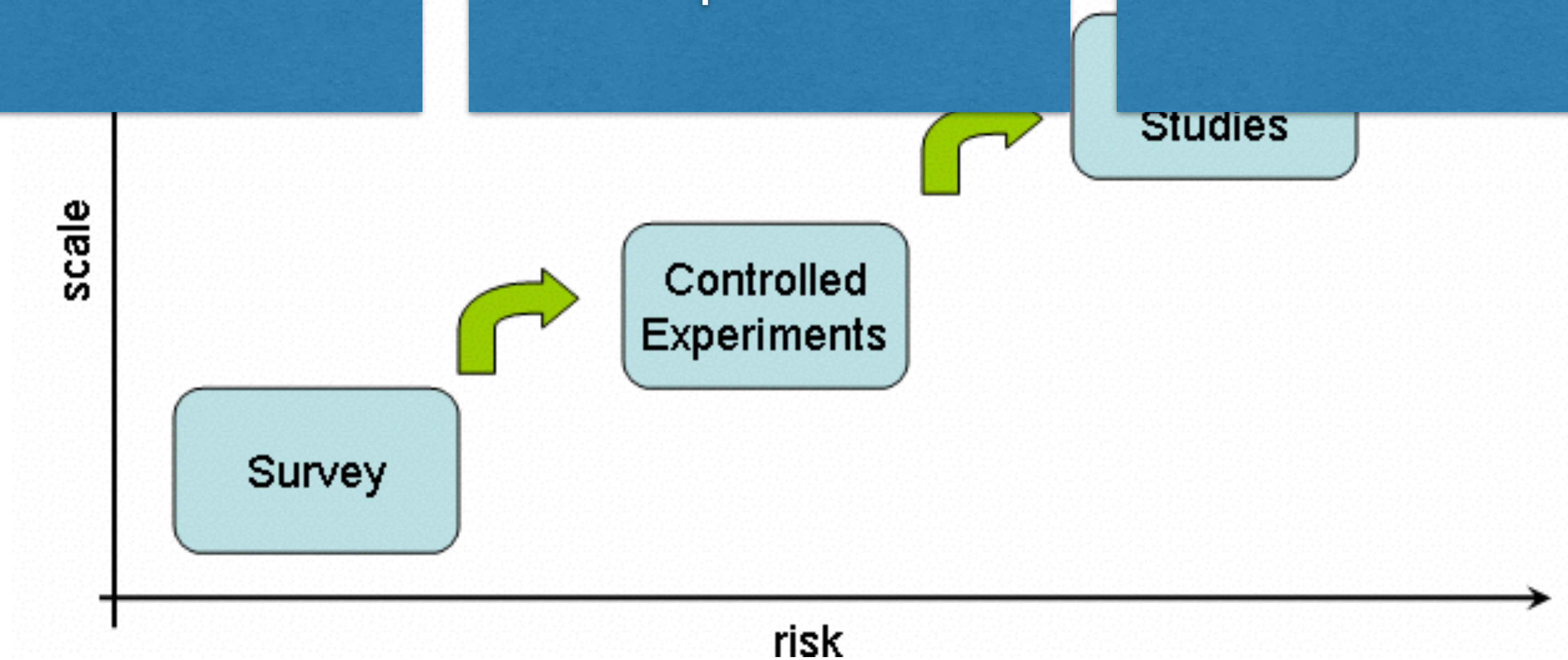
- **Quantitative**: to get numerical relations among variables
  - Are programmers more productive with Java than with C#?
  - Are defects correlated with cyclomatic complexity?
- **Qualitative**: to interpret a phenomenon just observing it in its context
  - E.g. by using explanations obtained by interviewing 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

JAVA (BEFORE VERSION 1.5)

```
public Vector aList        = new Vector;
public int    aNumber      = 5;
public int    anotherNumber;

aList.addElement(new Integer(aNumber));
anotherNumber = ((Integer)aList.getElement(0)).intValue();
```

PYTHON

```
aList = []
aNumber = 5

aList.append(aNumber)
anotherNumber = aList[0]
```

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
}
```

http://www.scala-lang.org/old/node/127

Static is Great!

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

An Experiment About Static and Dynamic Type Systems

Doubts About the Positive Impact of Static Type Systems on Development Time

Stefan Hanenberg

Institute for Computer Science and Business Information Systems
University of Duisburg-Essen
Schützenbahn 70, D-45117 Essen, Germany
stefan.hanenberg@icb.uni-due.de

Abstract

Although static type systems are an essential part in teaching and research in software engineering and computer science, there is hardly any knowledge about what the impact of static type systems on the development time or the resulting quality for a piece of software is. On the one hand there are authors that state that static type systems decrease an application's complexity and hence its development time (which means that the quality must be improved since developers have more time left in their projects). On the other hand there are authors that argue that static type systems increase development time (and hence decrease the code quality) since they restrict developers to express themselves in a desired way. This paper presents an empirical study with 49 subjects that studies the impact of a static type system for the development of a parser over 27 hours working time. In the experiments the existence of the static type system has neither a positive nor a negative impact on an application's development time (under the conditions of the experiment).

Categories and Subject Descriptors D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms Experimentation, Human Factors, Languages.

Keywords *Type Systems, Programming Languages, Empirical Study, Dynamically Typed Languages*

1. Introduction

Static type systems (see for example [1, 19]) are one of the major topics in research, teaching as well as in industry. In research, new type systems appear frequently either for existing programming languages (such as for example the introduction of Generics in Java) or new programming languages are constructed that provide a new static type system.

In teaching, students are educated in the formal notation of static type systems as well as in proofs on static type systems (see for example [1, 19]). In industry, type systems become important for different reasons. Possibly, a programming language in use evolves by introducing a new static type system. If this new static type system should be applied, developers need to be educated, which causes additional costs. Maybe existing libraries or products should be adapted to match the new type system which also causes additional costs. Finally, additional tools might be required due to the new type system (such as tools that measure the current state of the software product) which potentially also cause additional costs.

An example for such a static type system that evolves in a programming language with industrial relevance is Java: since the introduction of Generics in Java there is still a huge number of applications and APIs available that do not use Generics. Hence, for industry a main question is whether it is rewarding to educate developers to use the new static type system and whether there will be a return on investment for migrating existing non-generic APIs to the new static type system.

In general, for industry it is important to determine whether such an investment is reasonable, i.e. whether the expected benefit of static type systems represents some future revenues. This means, it is necessary to understand what the advantages and maybe additional costs of using a static type system are.

In industry, it is also observable that dynamically typed programming languages such as PHP or Ruby become more and more important for specific domains such as website engineering. Also, dynamically typed programming languages such as Tcl or Perl are still used in software development. For future developments of such languages it seems valid to ask, whether new releases of such languages should provide a static type system – assuming that a static type system has a positive impact on software development.

However, while static type systems are well-studied from the perspective of theoretical computer science, there is hardly any knowledge about whether a static type system plays a relevant role in the practical application of a

- static type system has no impact on development time

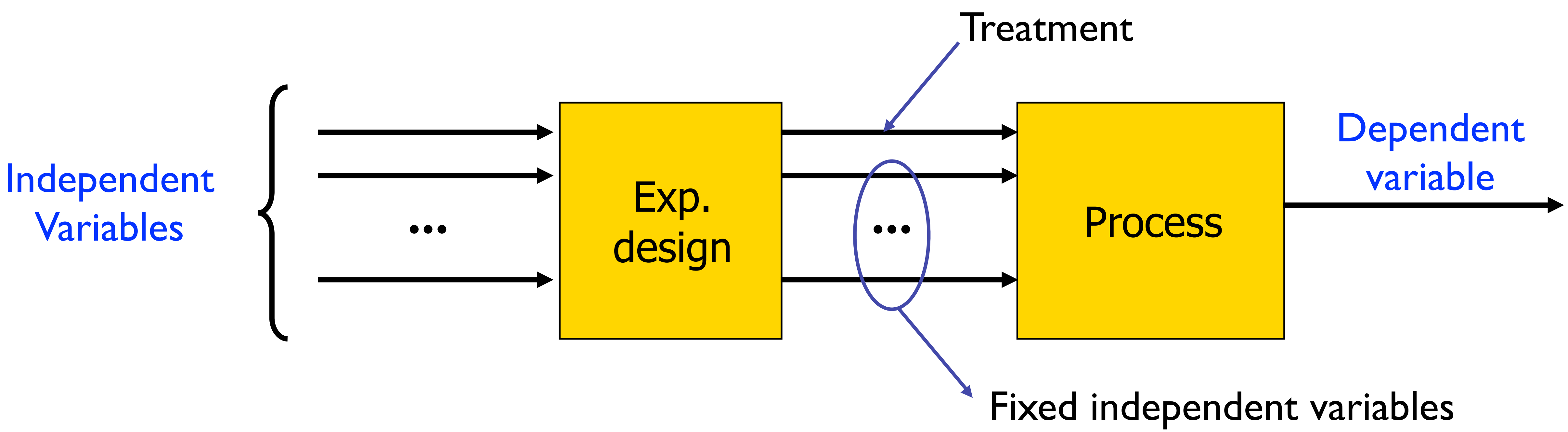
Intermezzo: Designing Controlled Experiments

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

Hypothesis formulation

- The experiment aims at rejecting a null hypothesis
- We can reject the null hypothesis → 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 µNold= µNnew

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 H0 µNold< µNnew

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…

Quiz

- 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?

```
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


Submission to IEEE Computer

An empirical comparison of
C, C++, Java, Perl, Python, Rexx, and Tcl

Lutz Prechelt (prechelt@ira.uka.de)
Fakultät für Informatik, Universität Karlsruhe
D-76128 Karlsruhe, Germany
+49/721/608-4068, Fax: +49/721/608-7343
http://wwwipd.ira.uka.de/˜prechelt/

March 14, 2000

Summary

80 implementations of the same set of requirements are compared for several properties, such as run time, memory consumption, source text length, comment density, program structure, reliability, and the amount of effort required for writing them. The results indicate that, for the given programming problem, which regards string manipulation and search in a dictionary, "scripting languages" (Perl, Python, Rexx, Tcl) are more productive than "conventional languages" (C, C++, Java). In terms of run time and memory consumption, they often turn out better than Java and not much worse than C or C++. In general, the differences between languages tend to be smaller than the typical differences due to different programmers within the same language.

Introduction

When it comes to the pros and cons of various programming languages, programmers and computer scientists alike are usually highly opinionated. The present work provides some objective information comparing several languages, namely C, C++, Java, Perl, Python, Rexx, and Tcl. It has the following features:

- The same program (i.e. an implementation of the same set of requirements) is considered for each language. Hence, the comparison is narrow but homogeneous.
- For each language, we analyze not a single implementation of the program but a number of separate implementations by different programmers. Such a group-wise comparison has two advantages. First, it smoothes out the differences between individual programmers (which threaten the validity of any comparison based on just one implementation per language). Second, it allows to assess and compare the variability of program properties induced by the different languages.
- Several different aspects are investigated, such as program length, programming effort, run time efficiency, memory consumption, and reliability.

We will consider the languages both individually and combined into groups: Perl, Python, Rexx, and Tcl and often called *scripting languages* and will form one group called the *script group*. The name scripting language commonly refers to languages that are for instance (more or less) interpreted rather than compiled, at least during the program development phase, and that do typically not require variable declarations. The alternative are the more conventional programming languages which I will call the *non-script group*. These languages (C, C++, and Java) are more or less compiled and require typed variable declarations. We will sometimes consider C and C++ as one group and Java as another.

The number of programs considered for each language and the execution platforms are described in Table 1. See the sidebars for a discussion of setup and validity of the study. A more detailed description of this study can be found in a technical report [4].

Results

The programs were evaluated on three different input files: z1000 contains 1000 non-empty random phone numbers, m1000 contains 1000 arbitrary random phone numbers (with empty ones allowed), and z0 contains no phone number at all (for measuring dictionary load time alone).

Run time

We will first investigate the total run time and then examine the initialization phase and the search phase separately.

Total: z1000 data set. The global overview of the program run times on the z1000 input file is shown in Figure 1. We see that for all languages a few very slow programs exist, but except for C++, Java and Rexx, at least three quarters of the programs run in less than one minute. We can make several interesting observations:

1

> 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

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?