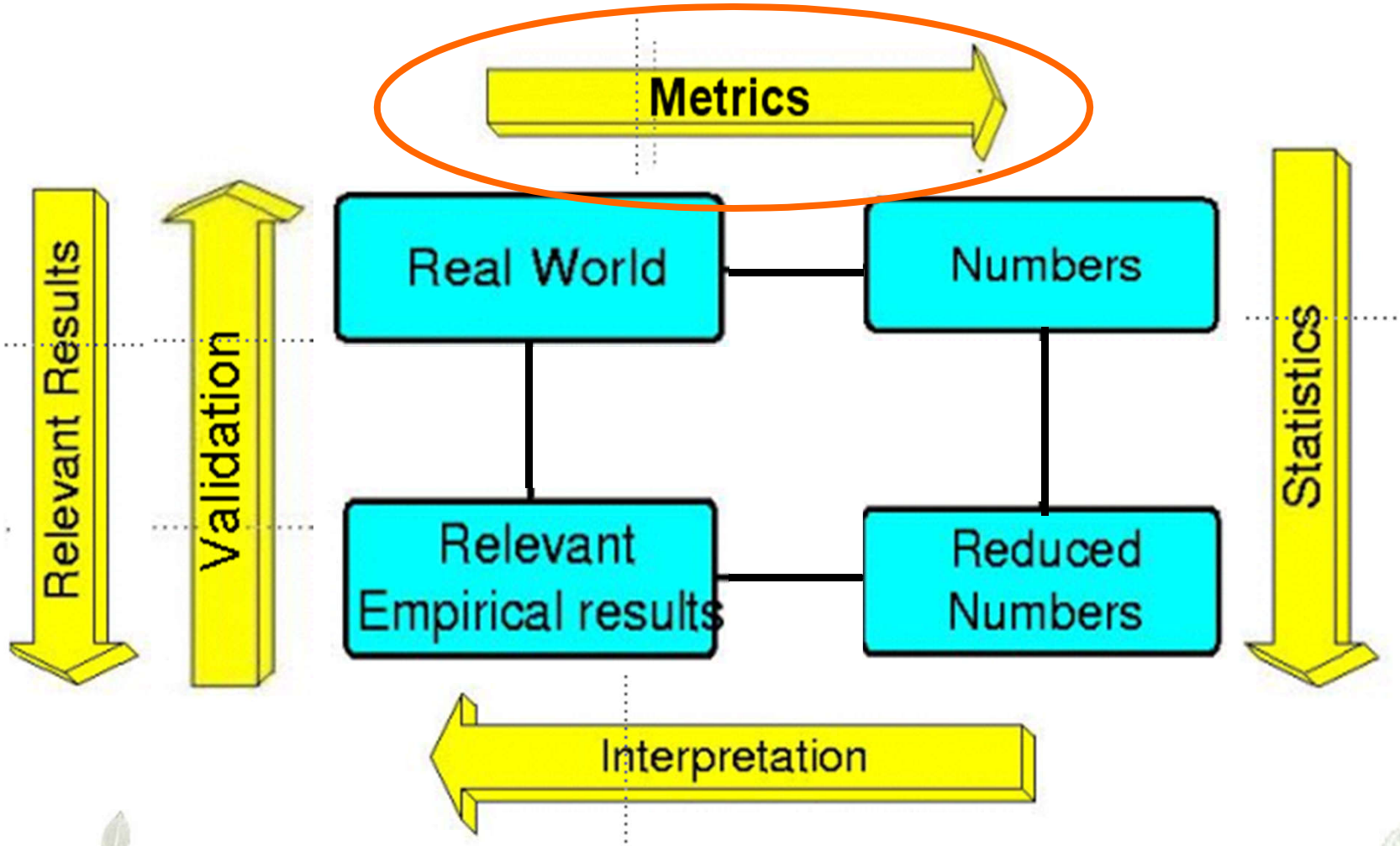
Software Metrics

Lecture 4

Data collection

Yuming Zhou

Measurement Theory



Contents

- Public data sets
- Analyzing source codes
- Mining software repositories
- Conducting controlled experiments

Section 1

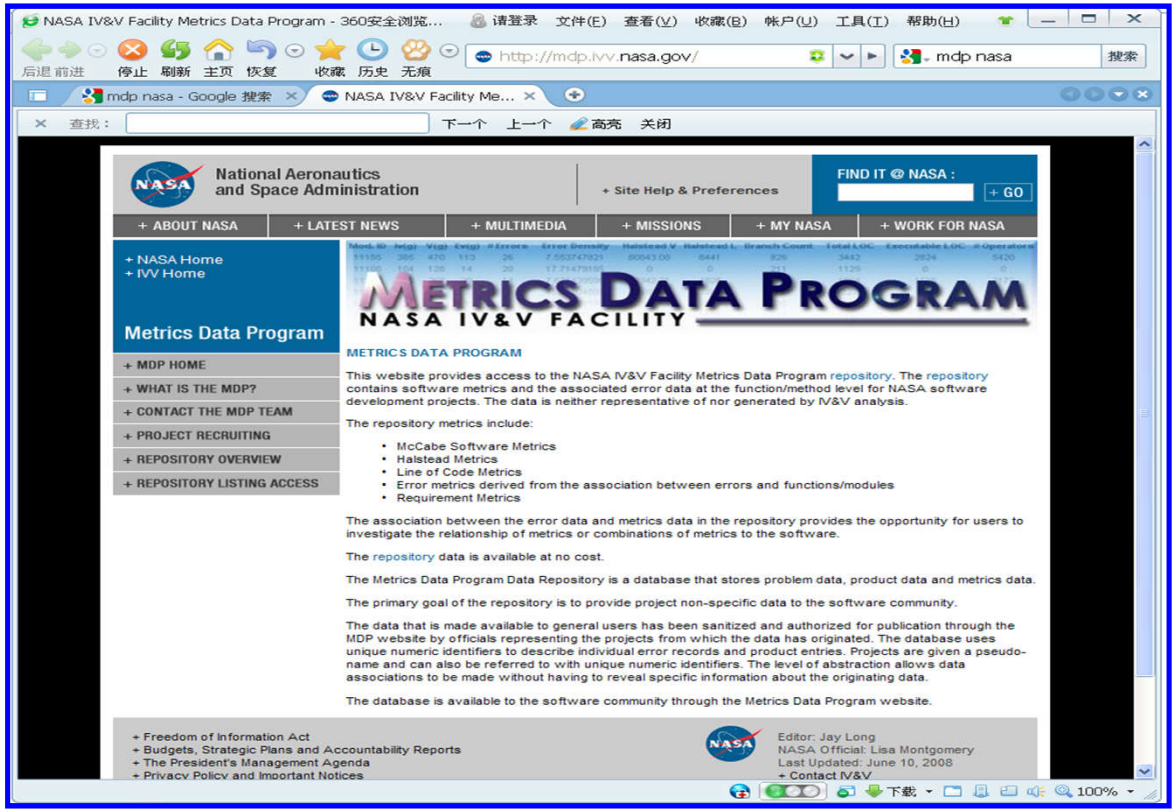
Public data sets

- NASA data sets
- tera-Promise data sets
- ISBSG data sets

NASA data sets

数据集 (http://mdp.ivv.nasa.gov/)

缺陷预测：13



http://openscience.us/repo/

NASA data sets

Data Set	Cases		Features	
	MDP	Promise	MDP	Promise
CM1	505	498	43	22
JM1	10878	10885	24	22
KC1	2107	2109	27	22
KC2	n.a.	522	n.a.	22
KC3	458	458	43	40
KC4	125	n.a	43	n.a
MC1	9466	9466	42	39
MC2	161	161	43	40
MW1	403	403	43	38
PC1	1107	1109	43	22
PC2	5589	5589	43	37
PC3	1563	1563	43	38
PC4	1458	1458	43	38
PC5	17186	17186	42	39

NASA data sets

数据集中的主要度量

LOC counts	LOC_total
	LOC_blank
	LOC_code_and_comment
	LOC_comments
	LOC_executable
	Number_of_lines
Halstead attributes	content
	difficulty
	effort
	error_est
	length
	level
	prog_time
	volume
	num_operands
	num_operators
	num_unique_operands
	num_unique_operators
McCabe attributes	cyclomatic_complexity
	cyclomatic_density
	design_complexity
	essential_complexity
Miscellaneous	branch_count
	call_pairs
	condition_count
	decision_count
	decision_density
	design_density
	edge_count
	essential_density
	parameter_count
	maintenance_severity
	modified_condition_count
	multiple_condition_count
	global_data_complexity
	global_data_density
	normalized_cyclomatic_compl.
	percent_comments
	node_count

NASA data sets

- M. Shepperd, Q. Song, Z. Sun, et al. Data quality: some comments on the NASA software defect datasets. IEEE TSE, 2013
- Q. Song, et al. A general software defect-proneness prediction framework. IEEE TSE, 2011
- T. Menzies, et al. Data mining static code attributes to learn defect predictors. IEEE TSE, 2007
- S. Lessmann, et al. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE TSE, 2008
- Y. Liu, et al. Evolutionary optimization of software quality modeling with multiple repositories. IEEE TSE, 2010
- Y. Zhou, et al. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM TOSEM, 23(1), 2014
- …

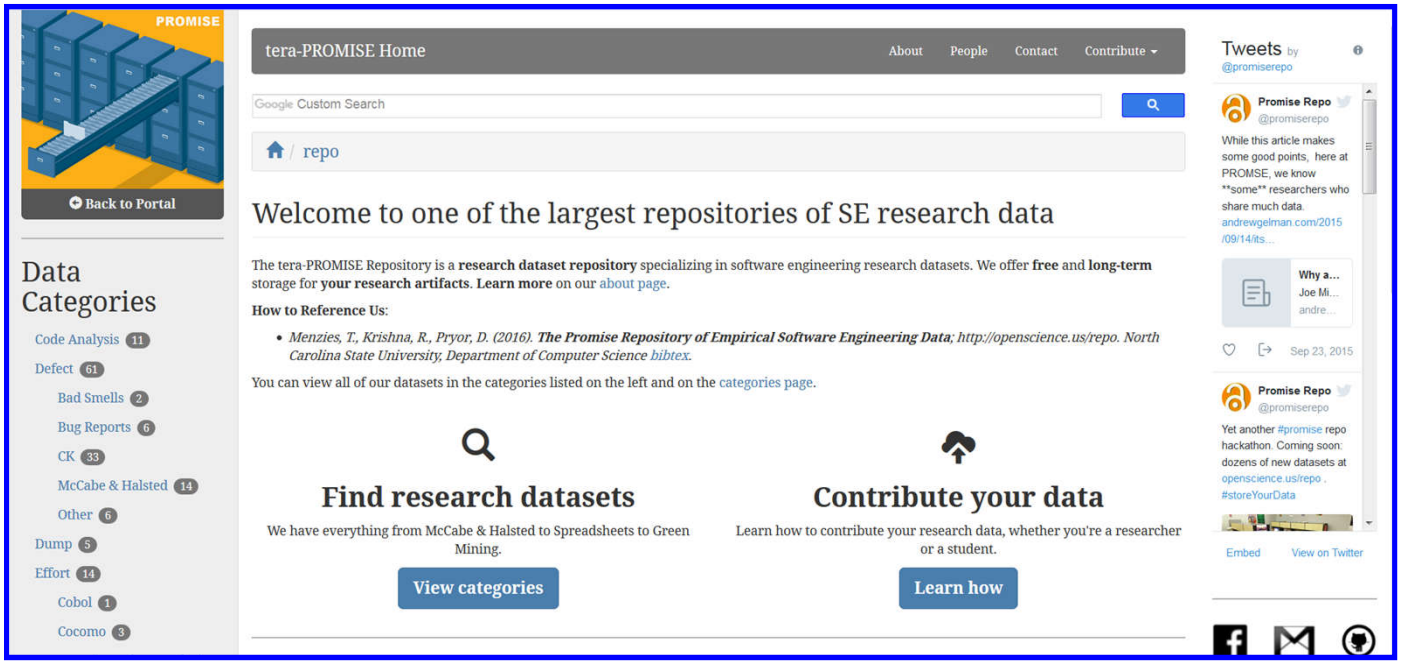
PROMISE data sets

PRedictOr Models In Software Engineering

数据集 http://openscience.us/repo/

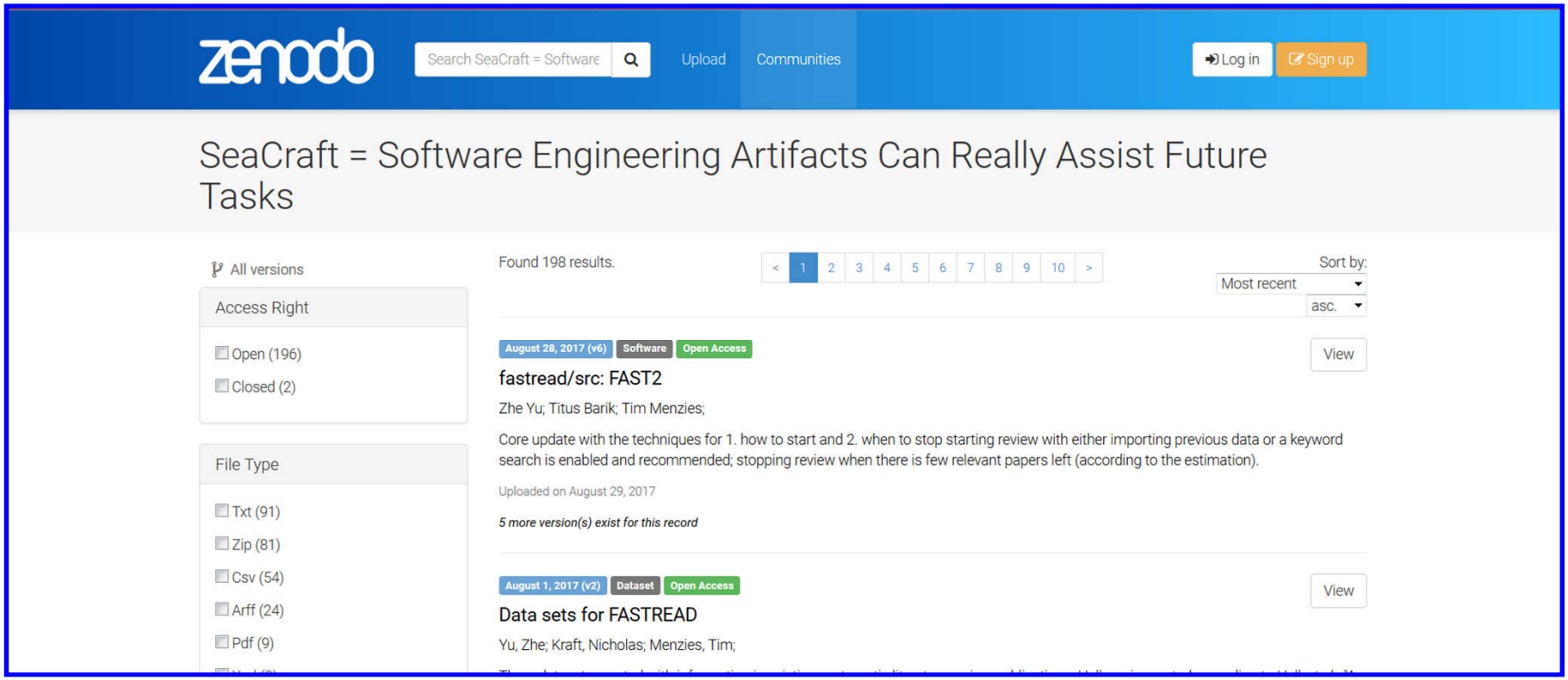
200+

代码分析
缺陷预测
工作量估算
能耗挖掘
...



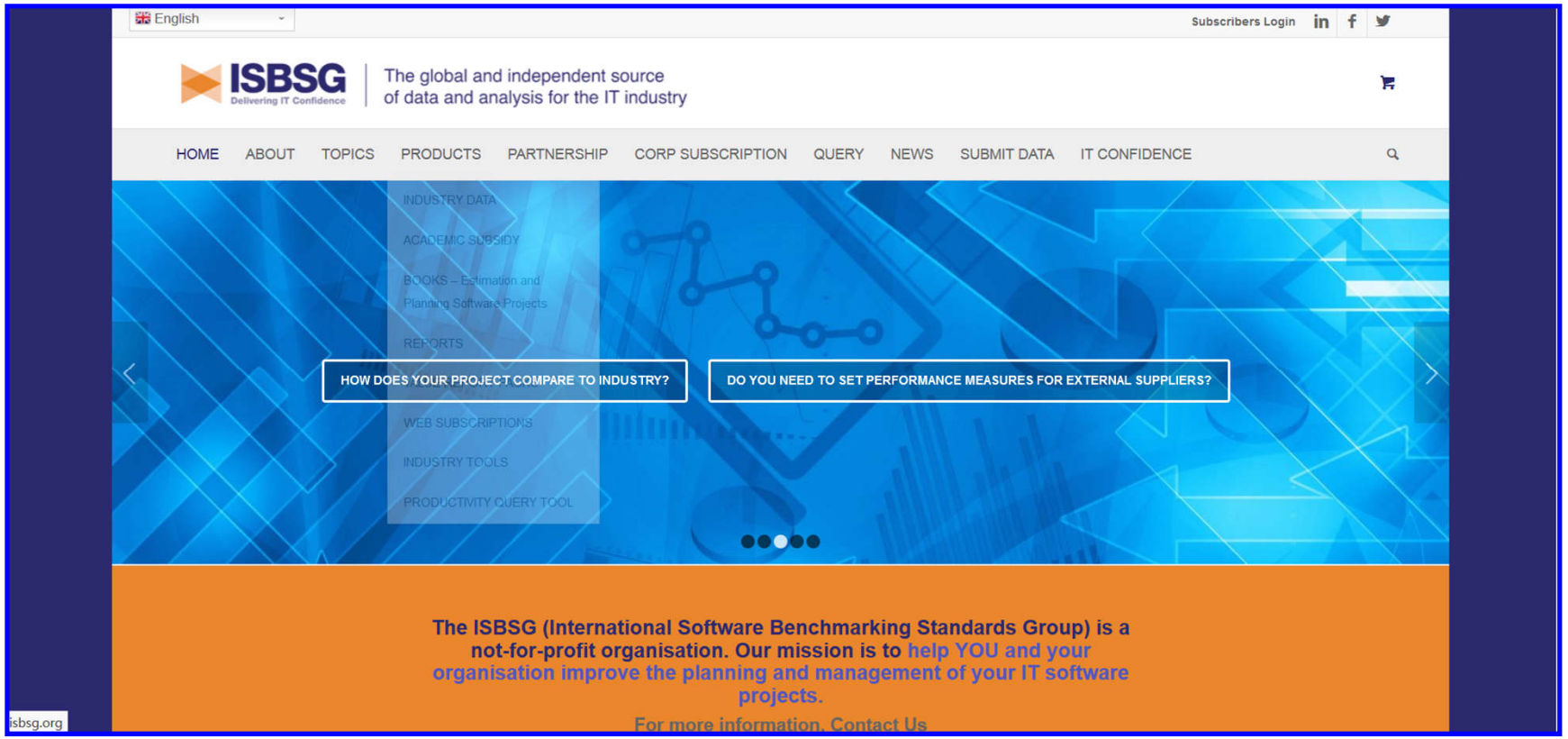
SeaCraft data sets

198



ISBSG data sets (花钱买!)

http://www.isbsg.org



其他数据来源

(1) 主流期刊

- TOSEM: ACM Trans. Software Eng. Methodology
- TSE: IEEE Trans. Software Eng.

(2) 主流会议

- ICSE: International Conference on Software Eng.
- FSE: Foundations of Software Eng.

(3) 专题会议

- ESEM: Empirical Software Eng. and Measurement
- PROMISE: Predictor Models in Software Eng.
- MSR: Mining Software Repositories

其他数据来源

(4) 一般期刊

- JSS: Journal of Systems and Software
- IST: Information and Software Technology
- ESE: Empirical Software Eng.
- IEEE Software
- CACM: Communication of the ACM
- SQJ: Software Quality Journal

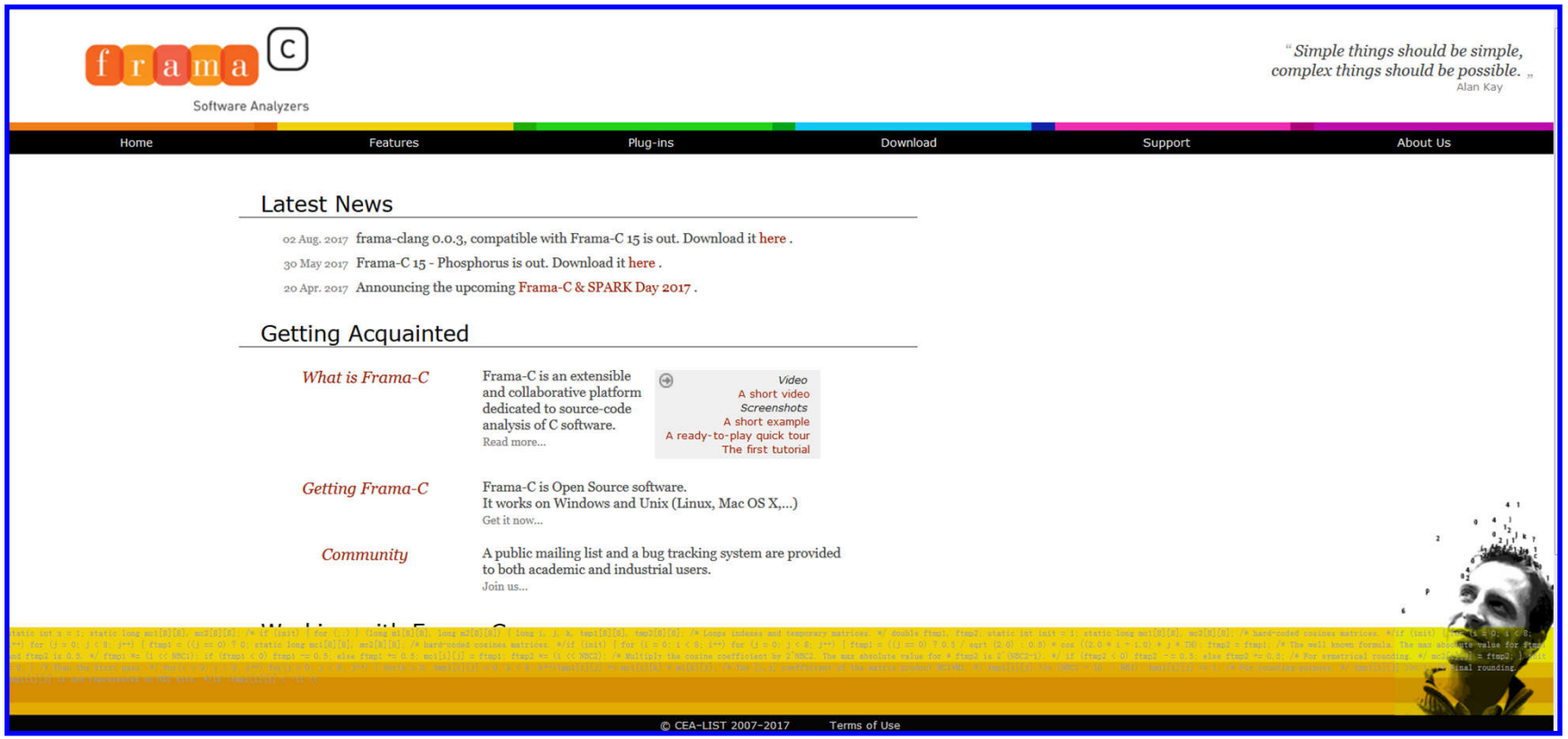
工欲善其事 必先利其器

Section 2

Analyzing source codes

- **Understand**
- **Frama-C**
- **Soot**

Frama-C


frama C
Software Analyzers
"Simple things should be simple, complex things should be possible."
Alan Kay
Home
Features
Plug-ins
Download
Support
About Us
Latest News
02 Aug. 2017 frama-clang 0.0.3, compatible with Frama-C 15 is out. Download it here .
30 May 2017 Frama-C 15 - Phosphorus is out. Download it here .
20 Apr. 2017 Announcing the upcoming Frama-C & SPARK Day 2017 .
Getting Acquainted
What is Frama-C
Frama-C is an extensible and collaborative platform dedicated to source-code analysis of C software.
Read more...
Video
A short video
Screenshots
A short example
A ready-to-play quick tour
The first tutorial
Getting Frama-C
Frama-C is Open Source software.
It works on Windows and Unix (Linux, Mac OS X,...)
Get it now...
Community
A public mailing list and a bug tracking system are provided to both academic and industrial users.
Join us...
© CEA-LIST 2007-2017
Terms of Use

Soot: A framework for analyzing and transforming Java and Android Applications

http://sable.github.io/soot/

Originally, Soot started off as a Java optimization framework.

By now, researchers and practitioners from around the world use Soot to analyze, instrument, optimize and visualize Java and Android applications.

View on GitHub

Soot

A framework for analyzing and transforming Java and Android Applications

Please help us improve Soot!

You are using Soot and would like to help us support it in the future? Then please support us by filling out this little web form.

That way you can help us in two ways:

- By letting us know how we can improve Soot you can directly help us prioritize newly planned features.
- By stating your name and affiliation you help us showcasing Soot's large user base. Thanks!

What is Soot?

Originally, Soot started off as a Java optimization framework. By now, researchers and practitioners from around the world use Soot to analyze, instrument, optimize and visualize Java and Android applications.

What input formats does Soot provide?

Soot: A framework for analyzing and transforming Java and Android Applications

What input formats does Soot provide?

Currently, Soot can process code from the following sources:

- **Java** (bytecode and source code up to Java 7), including other languages that compile to Java bytecode, e.g. Scala
- **Android** bytecode
- **Jimple** intermediate representation (see below)
- **Jasmin**, a low-level intermediate representation.

What output formats does Soot provide?

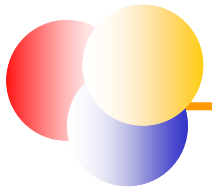
Soot can produce (possibly transformed/instrumented/optimized) code in these output formats:

- **Java** bytecode
- **Android** bytecode
- **Jimple**
- **Jasmin**

Soot: A framework for analyzing and transforming Java and Android Applications

What kind of analyses does Soot provide?

- Call-graph construction
- Points-to analysis
- Def/use chains
- Template-driven Intra-procedural data-flow analysis
- Template-driven Inter-procedural data-flow analysis, in combination with heros
- Taint analysis in combination with FlowDroid



Soot: A Java Optimization Framework

PLDI Tutorial

(still relevant despite being old!)




Section 3

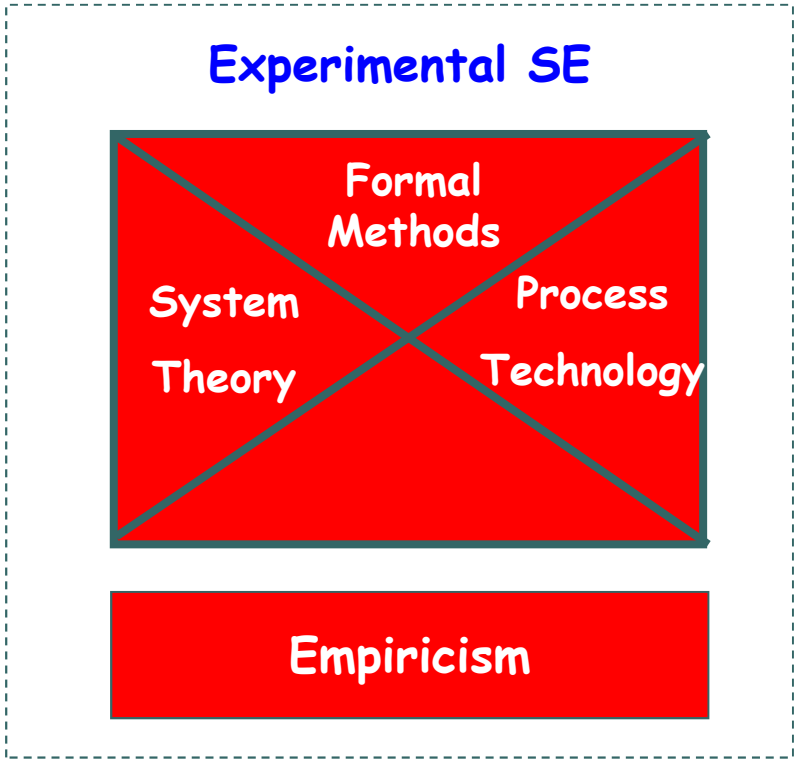
Mining software repositories



Section 4

Conducting controlled experiments

Conducting controlled experiments



- **Software Engineering comprises**
  - ✓ **(formal) methods (e.g., modeling techniques, description languages)**
  - ✓ **system technology (e.g., architecture, modularization, OO, product lines)**
  - ✓ **process technology (e.g., life-cyle models, processes, management, measurement, organization, planning QS)**
  - ✓ **empiricism (e.g., experimentation, experience capture, experience reuse)**

Conducting controlled experiments

验证假设：Java编写的代码比C语言编写的代码包含更少缺陷

- 用班上的40位同学做实验
- 测试20个Java程序+20个C程序(相同功能)
- 坐在前面的20个人测Java程序，后面20人测C程序
- 比较它们的缺陷数目，验证假设是否成立

实验方案合理吗？为什么？

Conducting controlled experiments

Definitions

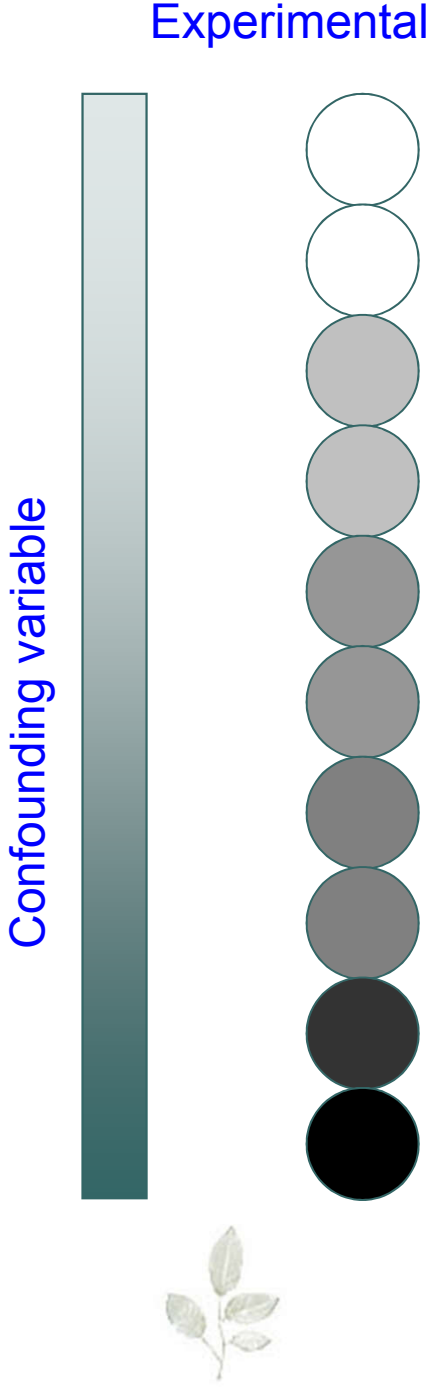
- Factor: Independent variables we can control 程序设计语言
- Levels: value of a factor Java/C
- Treatment: combination of levels Java/C
- Response: the variable measured 缺陷数目
- Experimental Units: The object upon which the response Y is measured 程序
- Parameter: any characteristic we want to be invariable throughout the experiment 程序的复杂性
- Undesired variations (block variables): Independent variables we can not control 同学的测试经验
- Subjects: people in the experiment 同学

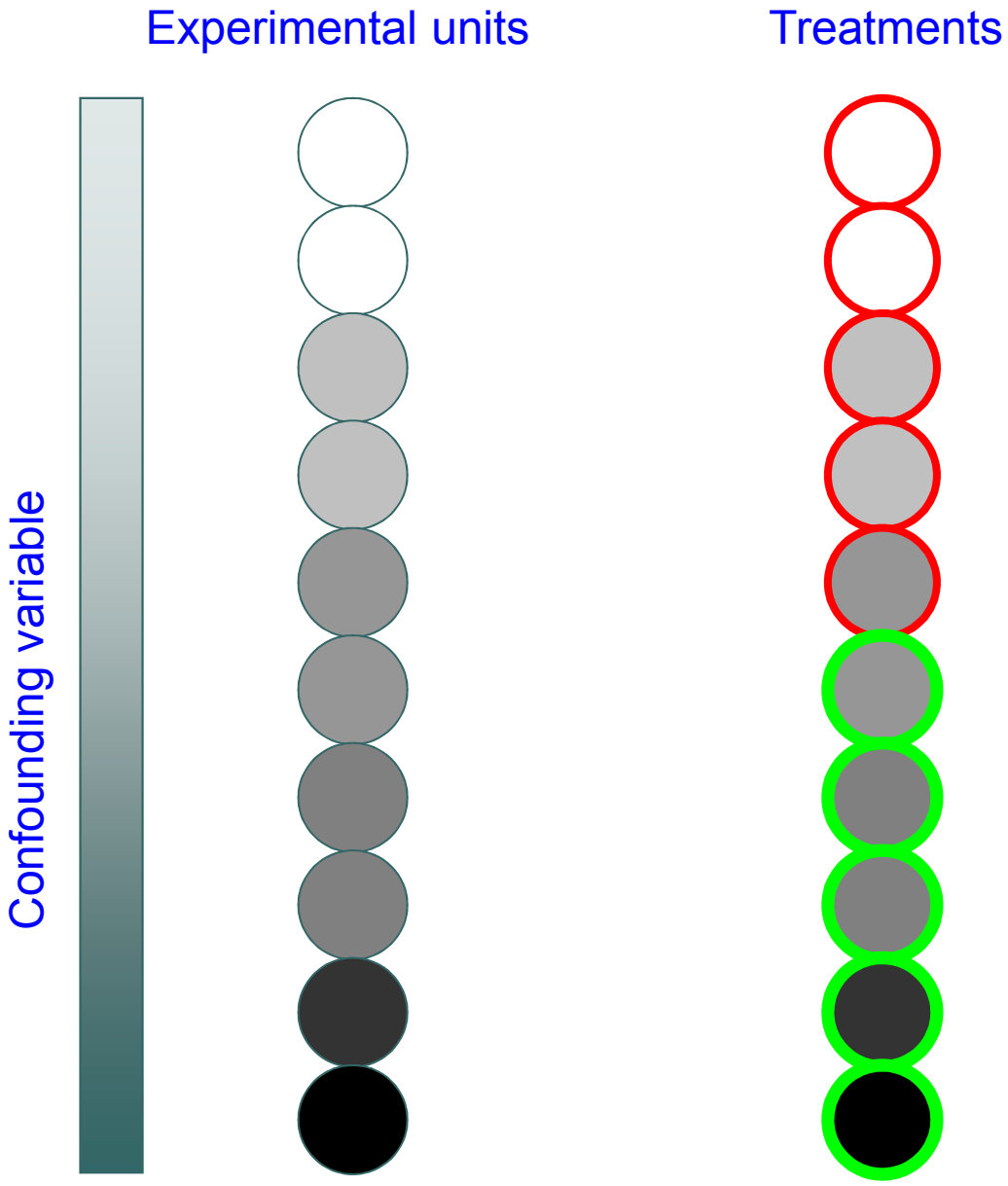
Table 4.1. Examples of factors and parameters in real experiments

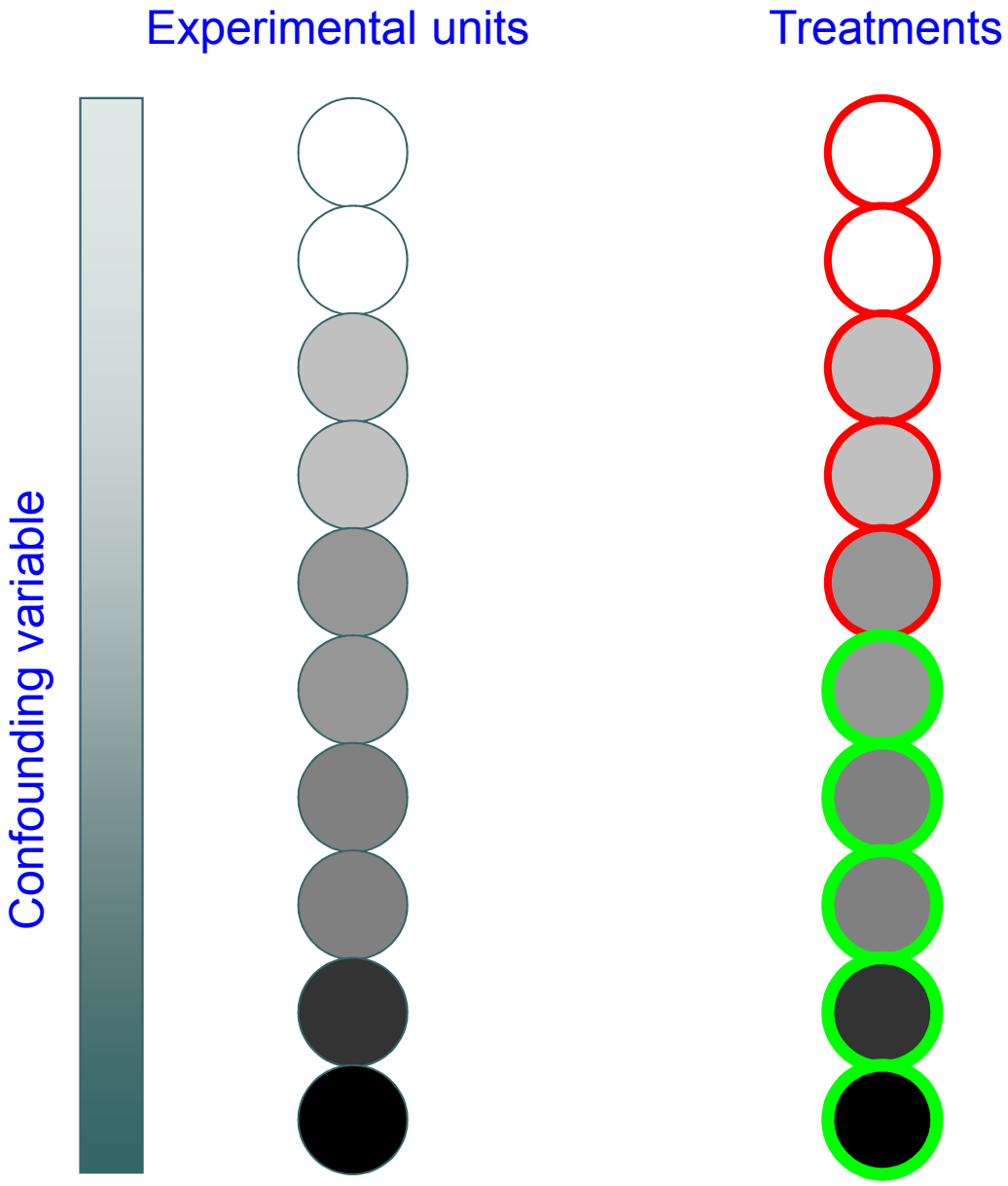
GOAL	FACTORS	PARAMETERS	REFERENCE
Studying the effect of different testing techniques on the effectiveness of the testing process	• Software testing techniques (code reading, functional testing, structured testing) • Program types: three different programs • Subject level of expertise (advanced, intermediate, junior)	• Testing process (first training, then three testing sessions and then a follow-up session) • Program size • Familiarity of subjects with editors, terminal machines and programs implementation language (good familiarity) • High-level language for implementing programs	(Basili, 1987)
Studying the ease of creating a program using an aspect-oriented approach and an OO approach	• Programming approach (Aspect J, Java)	• Problem complexity (low) • Application type (program with concurrence) • Subjects from a university course	(Murphy, 1999)
Studying the quality of code produced using a functional language and an OO language	• Programming language (SML, C++)	• Problem domain (image analysis) • Specific development process • Subjects experienced in both programming languages	(Harrison, 1996)

Principles of Experimentation

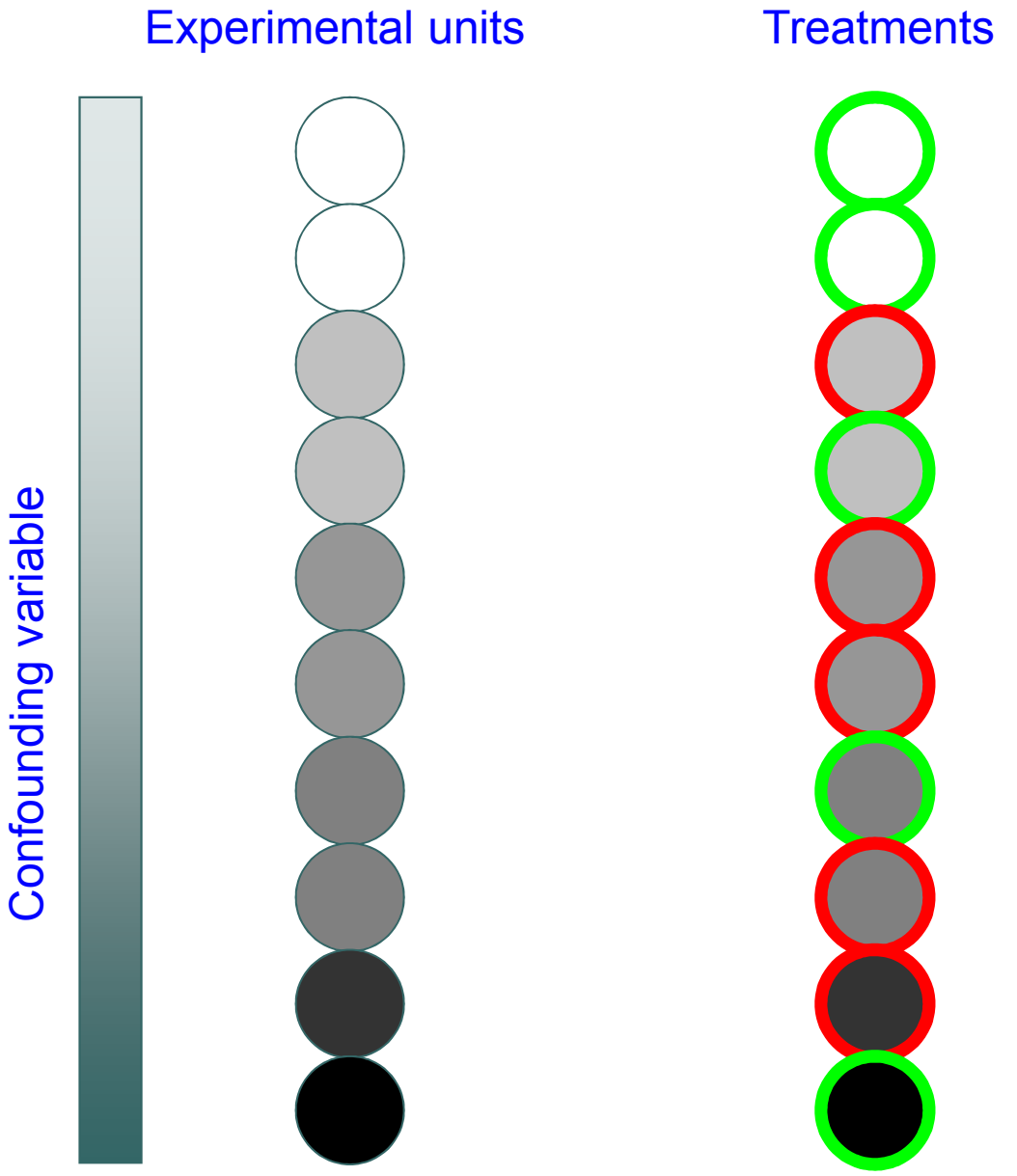
- Randomization - to satisfy independence of error observations, to decrease likelihood of systematic bias, to improve validity of casual inferences
- Blocking - to remove extraneous variation
- Completeness - to give balance and improve accuracy of error measurements

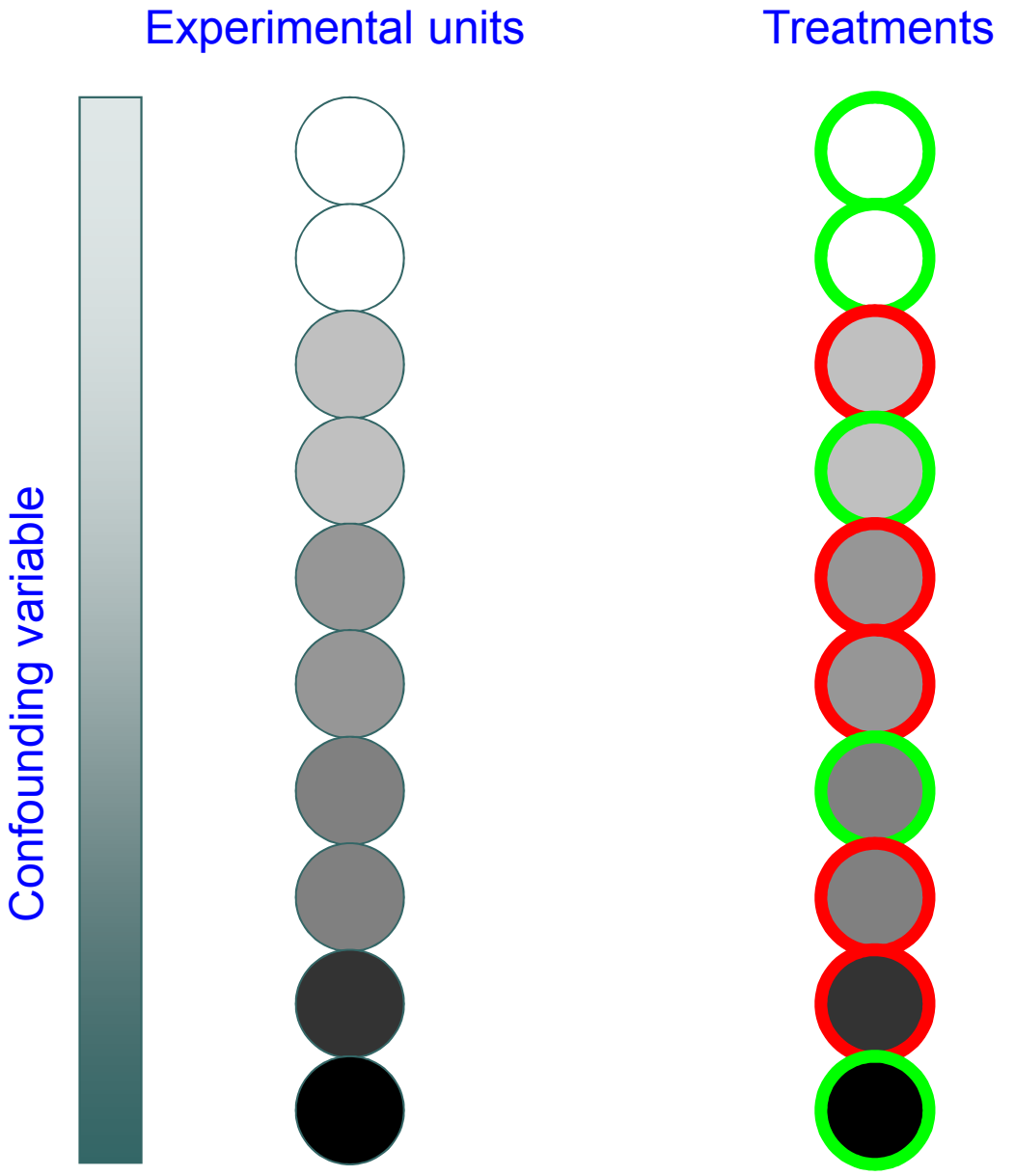

Experimental units
Confounding variable


Experimental units
Treatments
Confounding variable



Without randomization, the confounding variable differs among treatments


Experimental units
Treatments
Confounding variable



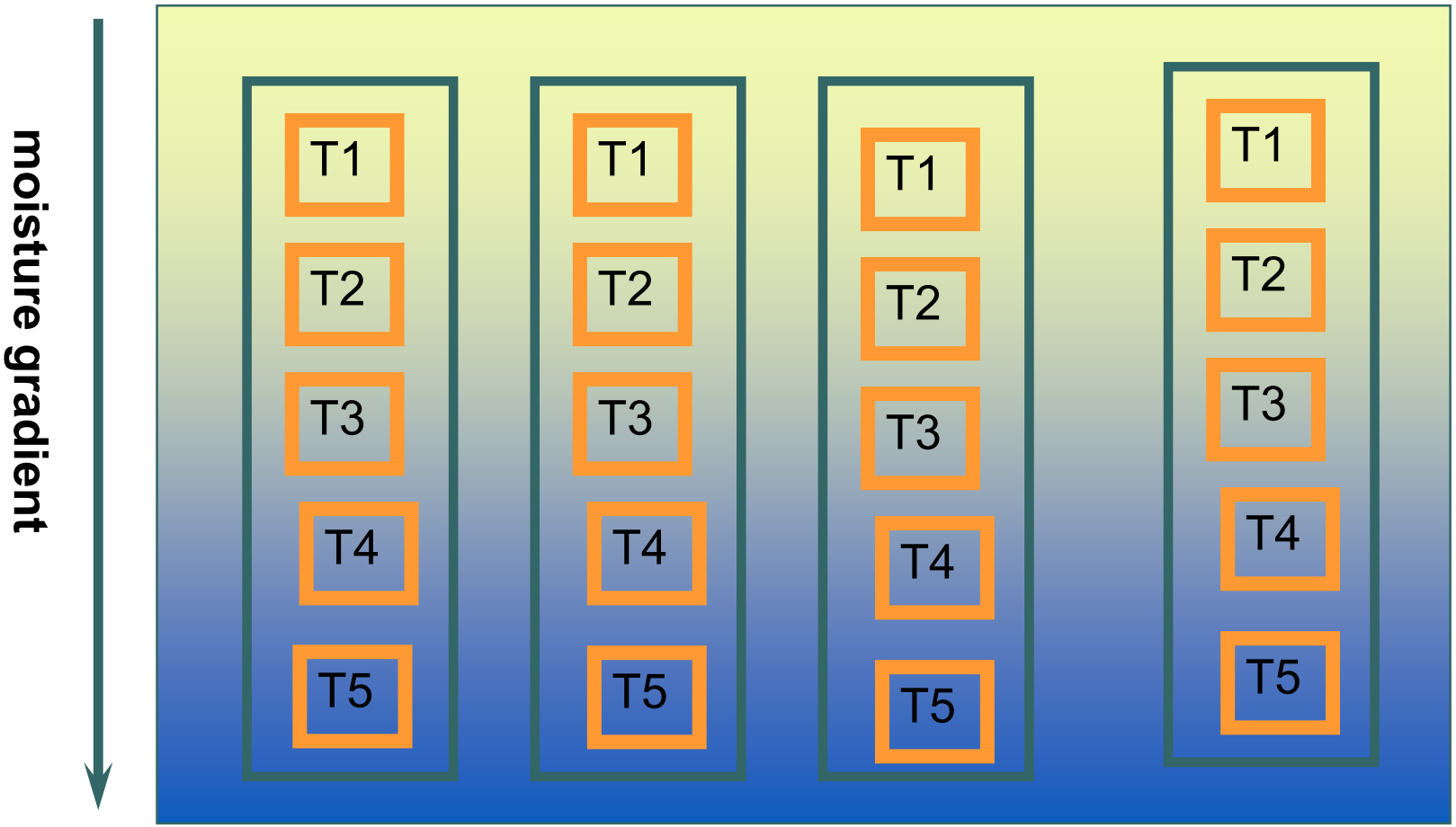
With randomization, the confounding variable does not differ much among treatments

Conducting controlled experiments

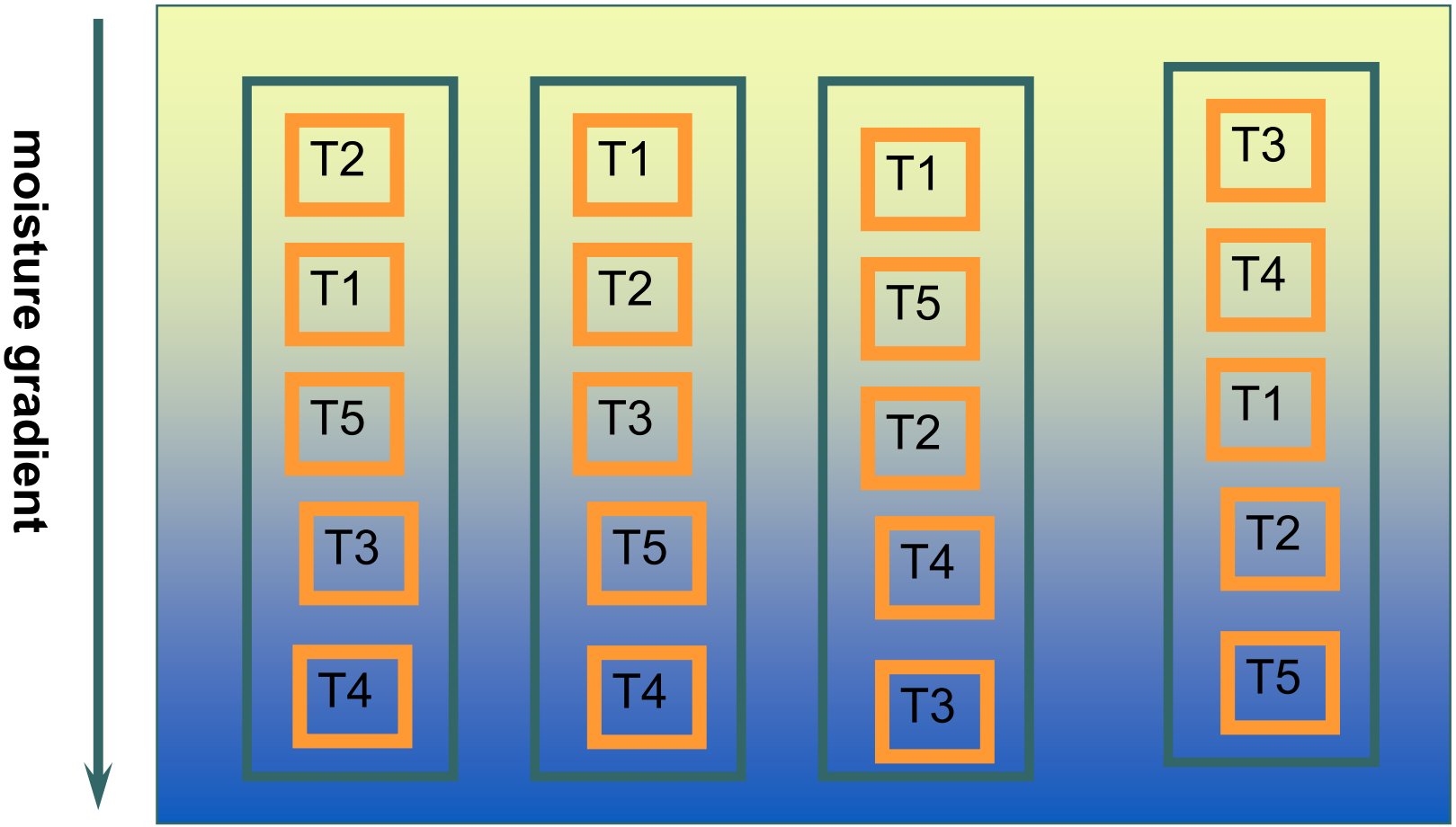
验证假设：Java编写的代码比C语言编写的代码包含更少缺陷

- 用班上的40位同学做实验
- 测试20个Java程序+20个C程序(相同功能)
- 坐在前面的20个人测Java程序，后面20人测C程序
- 比较它们的缺陷数目，验证假设是否成立

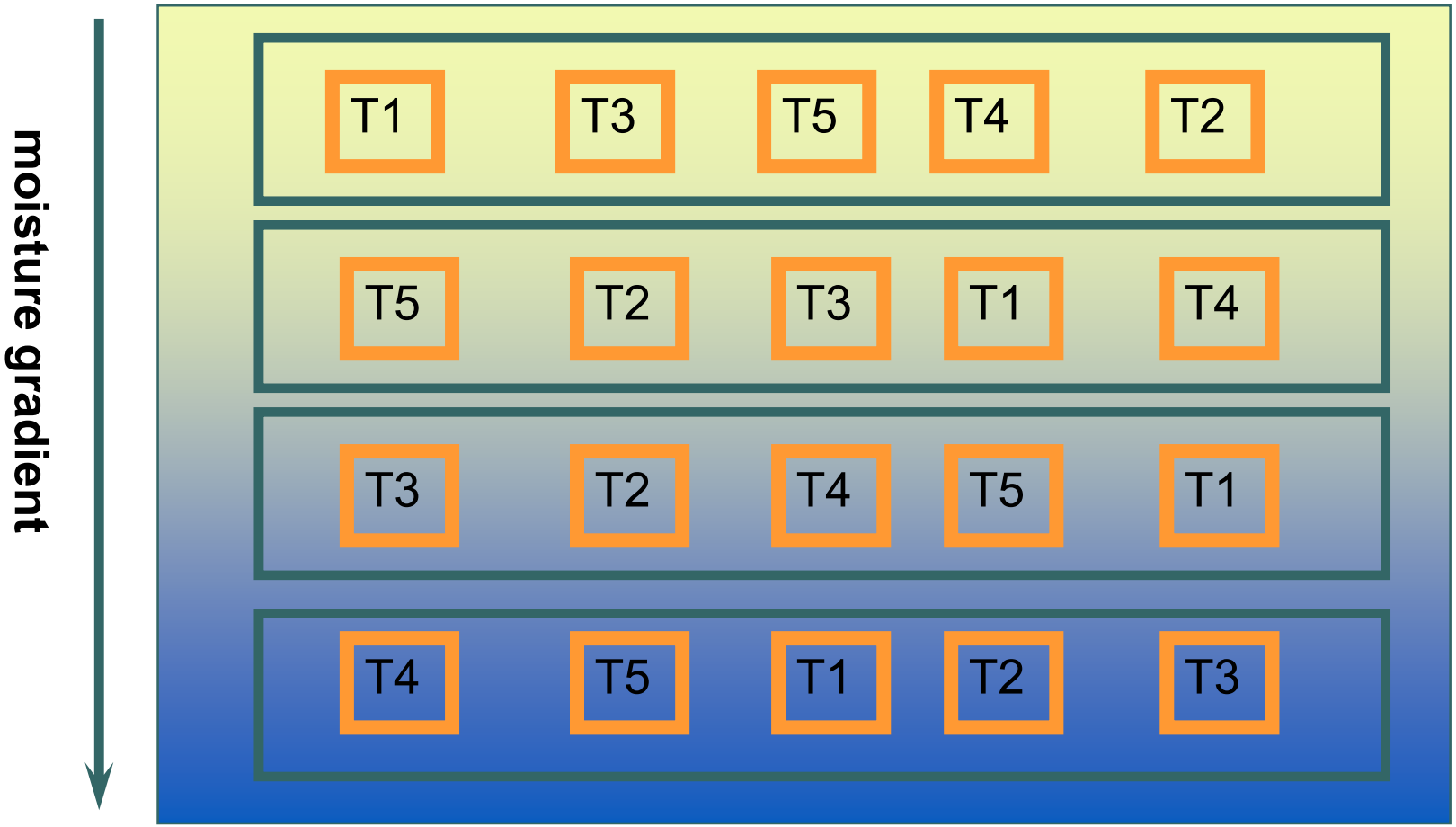
没有随机化！



Treatment effects confounded with moisture effect!



Treatment effects confounded with moisture effect!



Block effect now removes moisture effect, fair comparisons among treatments.

Randomized Block Design (RBD)

Any experimental design in which the randomization of treatments is restricted to groups of experimental units within a predefined block of units assumed to be internally homogeneous is called a **randomized block design**. Blocks of units are created to control known sources of variation in expected (mean) response among experimental units.

There are two ***classifications*** or ***factors*** in an RBD: block "effects" and treatment "effects".

Rules for blocking:

- Carefully examine the situation at hand and identify those factors which are known to affect the proposed response.
- Choose one or two of these factors as the basis for creating blocks.

Blocking factors are sometimes referred to as *disturbing factors*.

Examples of Typical Blocking Factors

Disturbing Variable	Experimental Unit
Nutrient gradient Water moisture gradient Slope differences Soil composition	Field Plot
Orientation to sun Flow of air Distribution of heat	Location in Greenhouse
Age Local density	Tree
Gender Age Socio-demographics	Person

Blocking Importance

How blocks are formed is critical to the effectiveness of the analysis.

- With field plots, blocks are laid out so that they are perpendicular to the maximum direction of change in the disturbing factor to be controlled.
- Wide border (discard) areas are used to overcome interference between neighboring plots (i.e. to maintain independence of responses) within blocks and between blocks.
- Time blocks may need discard times between “replications”.

This approach maximizes *within block homogeneity* while simultaneously maximizing *among block heterogeneity*.

Conducting controlled experiments

验证假设：Java编写的代码比C语言编写的代码包含更少缺陷

- 用班上的40位同学做实验
- 测试20个Java程序+20个C程序(相同功能)
- 坐在前面的20个人测Java程序，后面20人测C程序
- 比较它们的缺陷数目，验证假设是否成立

没有根据同学们的测试经验分组实验！

Complete or Incomplete Designs

Can all treatments be accommodated in each block?

Complete Block Design: Every treatment occurs in each block.

Incomplete Block Design: Not every treatment occurs in each block.

Complete

A	B	C	D
B	D	C	A
D	C	A	B

Incomplete

A	B	C
B	D	A
D	C	A

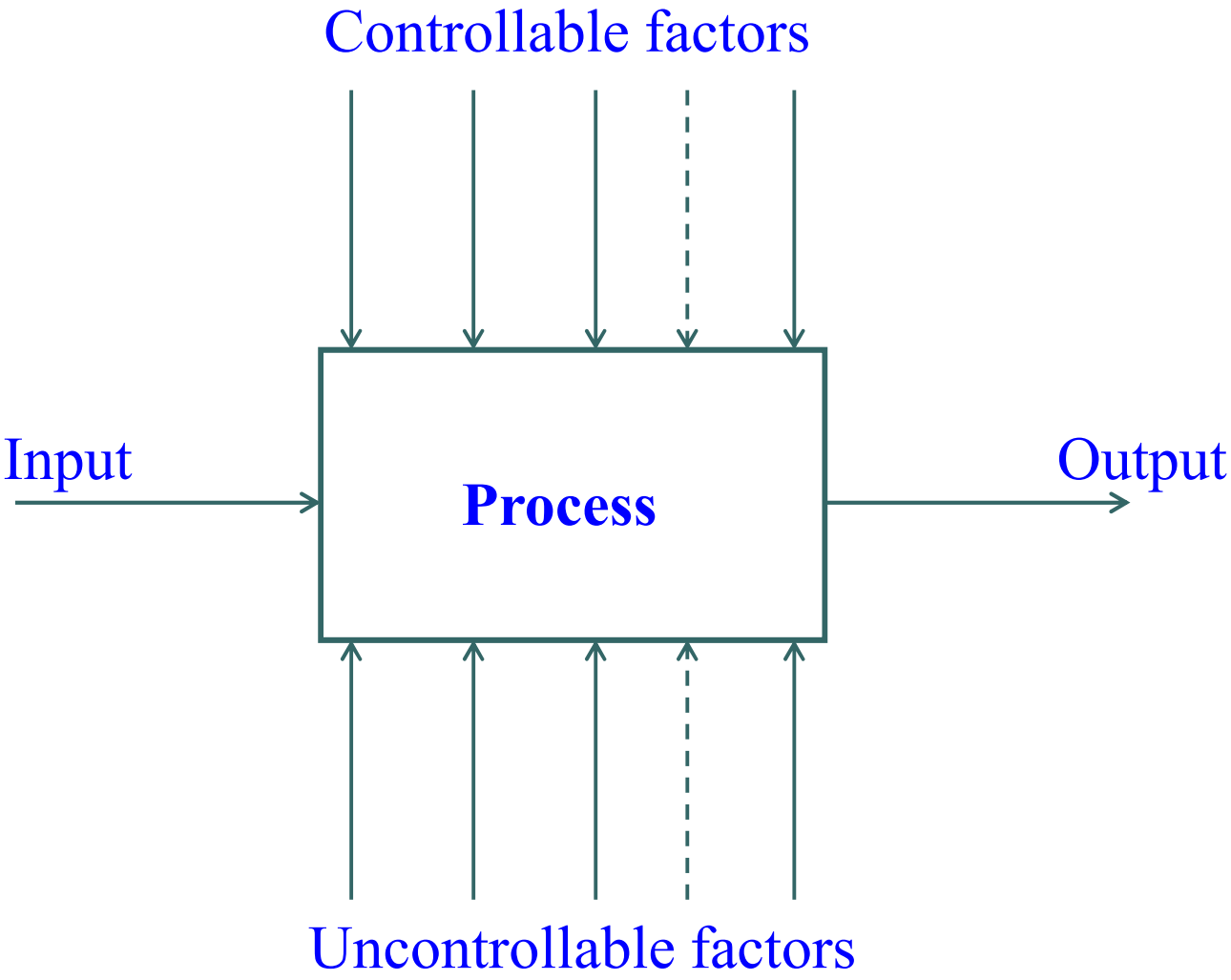
Balance in Designs

Balancing refers to the specific assignment of treatments to experimental units such that *comparisons of treatment effects are done with equal precision*. This is usually accomplished by equally replicating each treatment.

Balanced Block Design: The variance of the difference between two treatment means is the same regardless of which two treatments are compared. This usually implies that the overall replication (disregarding which blocks they are in) for the comparison of two treatments is the same for all pairs of treatments.

Partially Balanced Design: The variance of the difference between two treatments depends on which two treatments are being considered. This usually implies different replication for different treatments.

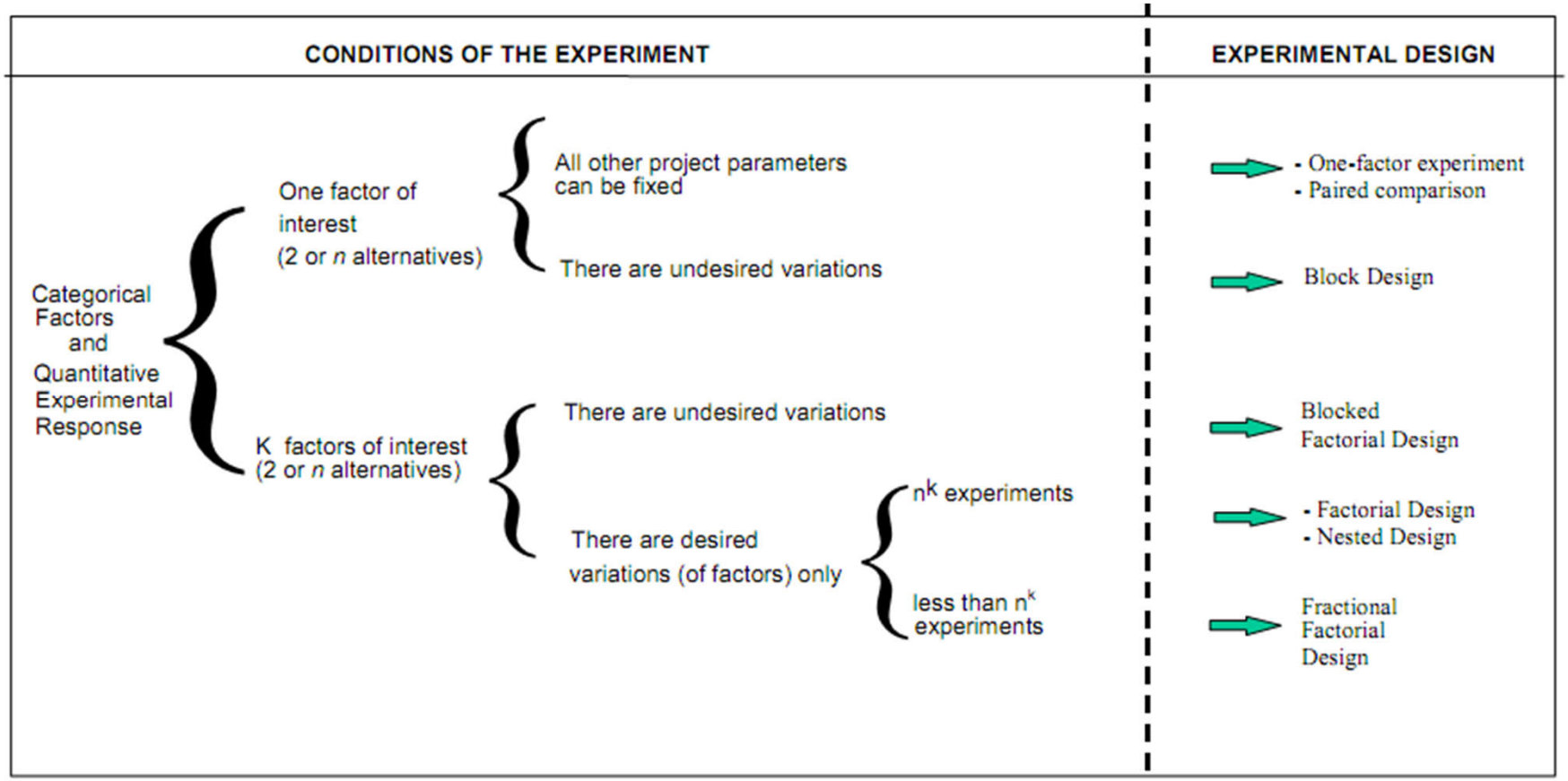
Unbalanced Designs: Unequal replication in each block - usually what one ends up with.


Controllable factors
Input
Process
Output
Uncontrollable factors

Principles of Experimentation

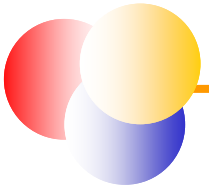
- Blocking - to remove extraneous variation
- Completeness - to give balance and improve accuracy of error measurements
- Randomization - to satisfy independence of error observations, to decrease likelihood of systematic bias, to improve validity of casual inferences

Different experimental designs



- 第5章. Basics of Software Engineering Experimentation. Natalia Juristo, Ana M. Moreno, Kluwer Academic Publishers, 2001

Summary



- How to analyze source codes?
- How to mine software repositories?
- How to conduct controlled experiments?




Further reading

- Basics of Software Engineering Experimentation. Natalia Juristo, Ana M. Moreno, Kluwer Academic Publishers, 2001

Thanks for your time and attention!

用Understand收集数据

```
my @funcEnts = $db->ents("function ~unknown
                                   ~unresolved");

foreach my $fun(@funcEnts){
        my $funname=$fun->name();
        my $fileref = $fun->ref("definein");
        next if (!defined $fileref);
        my $abspath=$fileref->file()->relname();
        my $countline=$fun->metric("CountLineCode");
        my $countpath=$fun->metric("CountPath");
        …
}
```

从patch收集bug数据

```
unsigned int __fastcall sub_10002(int a1)
{
  signed int v1; // ecx@1
  int v2; // edi@1
  bool v4; // zf@3

  v2 = a1;
  v1 = -1;
  do
  {
    if ( !v1 )
      break;
    v4 = *(BYTE *)v2++ == 0;
    --v1;
  }
  while ( v4 );
  return ~v1;
}
```

源代码