Experimental Software Engineering
– Introduction –

Fernando Brito e Abreu (fba@di.fct.unl.pt)
Universidade Nova de Lisboa (http://www.unl.pt)
QUASAR Research Group (http://ctp.di.fct.unl.pt/QUASAR)

Summary

- Introduction to ESE
  - Teaching ESE
  - Wrap up
Why should we care about … Experimental Software Engineering?

- Because it is a new hype?
- Because others are doing it?
- Because it is a good topic for publishing papers?
- Because “Engineering” requires “Experimentation”?
- Because we want to understand phenomena in software development?
- Because “traditional” Software Engineering is only a statement of ambition, not of accomplishment?
- Because we must convince senior management that something must be done?
- Because … it is yet another silver bullet to the crisis?

Why?

Abstract

- The Software [Engineering] Crisis
- The Scientific Method
- What is Experimental Software Engineering?
- Obstacles to Experimentation
- Case studies
Crisis? What crisis?

- After around 5 decades the software community is still unable to consistently produce reliable software on time and within budget that completely satisfies customers.


Software Engineering: crisis solution?

- There are no “Laws” in Software Engineering.

- Software Engineering is traditionally *qualitative*.

- Software engineering is full of:
  - “Theories” about effectiveness of software engineering practices, methods and techniques.
  - Unsubstantiated claims about efficiency of engineering practices, methods and techniques.
Some Software Engineering “theories”

ABOUT THE PRODUCT:
- Object-oriented systems are more maintainable than procedural systems
- Cohesion should be maximized and coupling should be minimized
  - Is this what practitioners are doing in practice?
- The complexity of a software system increases non linearly with its age (“Lehman’s “Law” of Software Evolution”)
- Aspect-oriented languages improve systems modularity

- How can we can assess these “theories” or evaluate these claims?

More Software Engineering “theories”

ABOUT THE PROCESS:
- Accurate effort estimates can be produced without a detailed design (e.g. using Function Points analysis)
- Software inspections are more efficient than testing
- Agile processes lead to shorter development cycles in the long term (until final deployment is fully achieved)
  - Is there a significant difference in the profitability of development projects that can be explained by the adopted process?
- More controlled processes lead to an higher effectiveness in defect removal

- How can we can assess these “theories” or evaluate these claims?
More Software Engineering “theories”

ABOUT INCIDENT/DEFECT MANAGEMENT:
- Some defect types take longer to correct than others
- The distribution of defect types is not uniform
- Aspects such as culture/nation, business area, licensing level (customer value) or size of the user community influence the bug solving process
  - Is there a concordance in the ordering between fault impact (user perspective) and urgency (support perspective)?
  - Does that concordance vary from country to country?
  - Does business area (e.g., public administration, banking, military, education, utilities) have an influence on incident priority?
  - Is there a relation between licensing levels and the volume of incidents? (by country, by incident type, by business area)

- How can we can assess these “theories” or evaluate these claims?

The Scientific Method

- It is a fundamental technique used by scientists to raise hypothesis and produce theories

**Assumption:** world is a cosmos not a chaos
- Scientific knowledge is **predictive**
- **Cause and effect** relationship exist
- Knowledge in an area is expressed as a set of **theories**
- Theories are raised based upon **hypothesis**

- The scientific method progresses through a series of **steps**
Steps in the Scientific Method (i/iii)

1. Observe facts
   - Fact means the “quality of being actual” or “a piece of information presented as having objective reality.”

2. Formulate hypothesis
   - An hypothesis is a tentative theory that has not been tested (knowledge before experimental work performed).
   - Formulation can be performed through:
     - Induction (generalization of observed facts)
     - Abduction (suggestion that something could be)

Steps in the Scientific Method (ii/iii)

3. Test the hypothesis (experimentation)
   - Build experiments to see if the hypothesis holds:
     - Experiments must be performed methodically
     - The hypothesis is used to make predictions
     - Predictions are compared with newly observed facts
     - Experiments can only prove that an hypothesis is false
     - If unsure revise hypothesis (step 2) in light of new experiments or observations

   - Experiments replication is required for wide acceptance of theories
     - Pharmaceutical industry (regulated by the F&DA)
     - Surgery techniques
     - Software world?

   - Experimentation evolves in steps
     - In vitro (controlled experiments)
     - Quasi-experiments (replication of experiments)
     - In vivo (widespread experiments after deployment)
Steps in the Scientific Method (iii/iii)

4 - Raise a theory
- After extensive experimentation corroborating the hypothesis
- A theory is a conceptual framework that explains existing facts or predicts new facts

5 – Express a law
- Law is a theory or group of theories that has been widely confirmed
- Confirmation can be obtained with intensive “in vivo” evidence
- A law should delimit its own application scope
  - e.g. Newton’s laws (holds for velocities much less than speed of light)
- Laws (as well as theories) are open to rebuttal

Conclusion:
- in Software Engineering we miss experimentation!

What is …
Experimental Software Engineering?

Is a branch of Software Engineering where, by means of experimentation we want to validate hypotheses raised by induction (and abduction), aiming at building theories that will allow us to:

- help understand the virtues and limitations of methods, techniques and tools, namely by assessing current SE claims
- express quantitatively the cause–effect relationships among sw process characteristics (resources and activities) and sw product characteristics
Sw development as a feedback loop …

**ACTIVITIES**
- Project planning and staffing
- Project management
- Requirements elicitation
- Designing
- Coding
- Configuration management
- Inspection
- Black-box testing
- Debugging
- Deployment
- User training

**PRODUCTS**
- Project plans
- Requirements specs
- Design models
- Source code
- Component libraries
- Estimation models
- Test batteries
- Executable code
- Installation manuals

**QUANTITATIVE EVALUATION**
- Process metrics
- Improvement actions

**RESOURCES**
- Project team members
- Users
- Methods
- Techniques
- Tools
- Operating System
- Hardware
- Physical Environment
- Schedule

... and a book:
- Experimentation in Software Engineering: An Introduction

Some landmark references?

- A journal ...
  - Empirical Software Engineering: An International Journal
Competence centers in ESE

- Lund University (Sweden)
  - SERG - Software Engineering Research Group
  - http://serg.telecom.lth.se/

- Florida Atlantic University (EUA)
  - ESEL – Empirical Software Engineering Laboratory
  - http://www.cse.fau.edu/research/ESEL/

- Florida Atlantic University (EUA)
  - ESEG – Experimental Software Engineering Group

+ Competence centers in ESE

- Fraunhofer IESE
  - http://www.iese.fhg.de/

- Fraunhofer USA
  - Center for Experimental Software Engineering, Maryland
  - http://fc-md.umd.edu
USA Government support

- CeBASE – sponsored initiative by the National Science Foundation

- University of Maryland College Park
- University of Southern California
- Fraunhofer Center for Experimental Software Engineering - Maryland
- University of Nebraska-Lincoln
- Mississippi State University

- http://www.cebase.org

Why is experimentation not common practice in Software Engineering? (1/2)

- Experiments cost too much
- Demonstrations will suffice
- There’s too much noise in the way
- Experimentation will slow progress
- Technology changes too fast
- Software developers not trained in importance of scientific method are not sure how to analyze data

(Walter Tichy, 1998)
Why is experimentation not common practice in Software Engineering? (2/2)

- The five fears of experimental validation

  - Shareholders - Commercial fear
  - Project managers - Budgeting fear
  - Team members - Evaluation fear
  - Software engineers - Misinterpretation fear
  - Researchers - Apathy fear (not any more 😊)

  (F. Brito e Abreu, 1997)

Cautions in Experimentation

- Experimenting in areas involving people is difficult

  - Not easy to repeat an experiment under the same conditions, if the human factor has a strong influence

  - However, if the sample is large, the individual influences get averaged and then cancelled
Software Experimentation Requirements

- Requires samples of considerable size relative to real world software development projects
  - namely process data (efforts, schedules, defect data, etc)
  - this implies a relation among university and sw companies
  - Researchers need to publish their results
    - non-disclosure agreements ...

- Real world data samples are meaningless unless they are in relation to a theoretical **model of the phenomenon**
  - This gap can be fulfilled through the expression of an adequate **domain ontology (e.g a metamodel)**

A brief overview of some case studies

- Some concrete examples from our own research on the field
  - Modularity reengineering
  - Design complexity assessment
  - Controlling the evolution of legacy systems
  - Component architecture evaluation

- Objective: show how Software Engineering is becoming more quantitative and automated
Case study 1: Motivation
Software refactoring

Objective:
- Improve quality characteristics of existing software systems (without modifying its functionality) to facilitate their evolution and, as such, to extend its life-time

Examples:
- Redocumentation
- Design refactoring - identifying and applying design patterns, modularity refactoring, ...
- Code refactoring - eliminating redundancies, identifying reusable components and enforcing their reuse, verifying compliance with naming standards, automatic reformatting for readability

Case study 1: Modularity Refactoring
Case study 1:
Modularity Refactoring

![Bar chart for modularity refactoring](chart)

Case study 2:
Design Complexity Evaluation

Problems:
- Code metrics are too late!
- Formalization is required, but ...
  - \textbf{CSCO - C}ount of \textbf{S}ynchronization-based \textbf{C}oupled \textbf{O}bject types \cite{Poels2001}

\[
\text{CSCO}(P) = \# \{ Q \in T - \{ P \} \mid \exists e \in A: (\tau_1(e, P) = C \land \tau_1(e, Q) = E) \lor (\tau_1(e, P) = E \land \tau_1(e, Q) = C) \}
\]
Case study 2: Design Complexity Evaluation

Extract of UML Metamodel:

NOC() -- Number of Children
Informal: number of classes inhering directly from current class

Classifier:: NOC(): Integer
post: result = children()->size()

where

children(): Set(GeneralizableElement) = self.generalization -> collect(g | g.parent) -> excluding(self)->asSet
Case study 2: Design Complexity Evaluation

\[ \text{DIT}(\cdot) \] – Depth in the Inheritance Tree
Informal: Size of the longest inheritance chain until a root class

**Classifier:: DIT (): Integer** =

\[
\begin{align*}
\text{if } \text{self.isRoot} & \text{ then 0} \\
\text{else} & \\
\text{if } \text{PARN()} = 1 & \text{ then} \\
1 + \text{self.parents()}.\text{iterate}(\text{elem: GeneralizableElement}; \text{acc:Integer} = 0 | \text{acc} + \text{elem.oclAsType(Class).DIT()} ) \\
\text{else} & \\
\text{self.parents()}.\text{iterate}(\text{elem: GeneralizableElement}; \text{acc:Integer} = 0 | \text{acc} + \text{elem.oclAsType(Class).DIT()} ) \\
\text{endif} & \\
\text{endif}
\end{align*}
\]
Case study 3: Controlling the Evolution of Legacy Systems

Bar Chart

- Changed Modules
- Frequency

Complexity

- Modules
- 0% 20% 40% 60% 80% 100%

7-Apr-08 Experimental Software Engineering / Fernando Brito e Abreu
Case study 3: Controlling the Evolution of Legacy Systems

LOGISCOPE SOURCE CODE REPOSITORY

SPSS PRODUCT STRUCTURE REPOSITORY

R.A.R.E. HTML PAGES

UNIX Workstation NTServ. Web cl.

Case study 4: Evaluating components and component assemblies


7-Apr-08 Experimental Software Engineering / Fernando Brito e Abreu
Case study 4:
Evaluating components and component assemblies

Bibliography

Empirical Methods and Studies in Software Engineering

Lecture Notes on Empirical Software Engineering

Empirical Studies in Software Engineering

Basics of Software Engineering Experimentation

Experimentation in Software Engineering: An Introduction
Online Bibliography

Statistics

- Electronic Statistics Textbook (StatSoft, Inc.)

- A New View of Statistics (Will G. Hopkins)

- PROPHET StatGuide (Glossary)
  - [http://www.basic.nwu.edu/statguidefiles/sg_glos.html](http://www.basic.nwu.edu/statguidefiles/sg_glos.html)

Back to the summary …

- Introduction to ESE

- Teaching ESE

- Wrap up
Are professors to blame for the crisis?

- “Real world” - Programming in the large
  - large teams, large overheads, deliverables often late, over budget
  - no assessment by peers ⇒ quality doesn’t pay!

- Universities - Programming in the small
  - small teams, small overheads, schedules are met, budget is not a constraint
  - assessment by “graduate peers” ⇒ quality pays!

- We have a paradigm mismatch!

ESE – A step forward …

- Maybe we cannot teach programming in the large, …
  - … but we can teach how to assess technology in the large!

- Let us see how we can achieve this!
A proposed curricula on ESE

- Formalization of domain knowledge

- Ontologies and meta-modeling (MM) with examples
  - Motivation
  - Software process MM
  - Object-relational database schemas MM
  - The OCL MM
  - UML 2.0 MM

- lab session: building and loading a metamodel

A proposed curricula on ESE

- Understanding the state of the art of ESE
  - Revision of related work

- Producing hypothesis
  - about the population and about the sample
    - requires some descriptive statistics
  - requires careful observation of facts

- Measurement concepts
  - scale types, objectivity and subjectivity, direct and indirect measuring, …
A proposed curricula on ESE

- Experimental design
  - Variables: independent and dependent
  - Outlier and extremes detection and removal
  - Plan to systematically vary the independent variables
  - Identify threats to results validity
    - construction validity - is the relation between theory and observation adequate?
    - Internal - do results show a cause-effect relationship in the sample?
    - external - can results be generalized to the population?

A proposed curricula on ESE

- The Object Constraint Language (OCL)
  - Motivation
  - OCL syntax and semantics
  - Modeling business constraints with OCL

- lab session: design instantiation and testing of a constrained UML model
A proposed curricula on ESE

Data collection
- Objectivity, non intrusiveness and efficiency aspects
- Tool integration

Expressing and obtaining complexity metrics with OCL upon a MM
- Software process complexity
- Object-relational database schema complexity
- Complexity of UML static and dynamic models

lab session: expressing and calculating complexity metrics for UML designs

Data analysis
- testing hypotheses
  - tests to verify adherence to a given distribution
  - tests to verify independence (parametric and non-parametric)
  - evaluation of tests’ significance

Results presentation
- results interpretation (hypothesis hold or not?)
- practical significance (requires domain knowledge)
A proposed curricula on ESE

- **Advanced topics**
  - Principal component analysis
    - for reducing the number of independent factors (e.g. in building a quality model)
  - Cluster analysis
    - for assessing and improving modularity
  - Multivariate linear and nonlinear regression analysis
    - for building estimation models
  - Moving average methods (e.g. ARIMA)
    - for expressing the evolution of large legacy systems

Who’s teaching ESE?

Several CS departments have recently started dedicated ESE courses or include this topic very strongly in their SE courses:

- Colorado State University (USA)
- George Mason University (USA)
- University of Texas at Austin (USA)
- University of Maryland (USA)
- Walden University (USA)
- Worcester Polytechnic Institute (USA)
- University of Calgary (Canada)
- Oregon State University (USA)
- Florida Atlantic University (USA)
- University of Otago (New Zealand)
- University of Sannio (Italy)
- University of Oulu (Finland)
- Linköpings Universitet (Sweden)
- Lund University (Sweden)
- University of Skövde (Sweden)
- Kaiserlautern University (Germany)
- NTNU University (Norway) – 2005 version
- Technical University of Sydney (Australia)
- ...
Conclusions

- Being able to obtain “proofs” of SE claims may be a very important ability for future engineers

- For justifying technology migration …
  - e.g. assessing and comparing software artifacts (designs, code, tools, processes)
- For justifying budgets to upper management …
- For negotiating with clients on a sound basis …
  - building estimation models (development effort, reliability, maintainability)
- For achieving consensus in technical teams …
Conclusions

- Teaching ESE requires:
  - Formalization of domain knowledge
    - We teach how to develop ontologies with UML
  - Data samples of considerable size
    - To allow reaching meaningful results
  - Precise collection of quantitative data
    - This often requires building or adapting some tools
    - We teach how to formalize and automate collection with OCL
  - Data analysis
    - This requires learning some specific statistical techniques
    - We adopt an “hands-on/tool-guided” approach to its learning

The END!

- Thanks for your attention 😊

- For more information:
  - fba@di.fct.unl.pt
  - http://ctp.di.fct.unl.pt/QUASAR
Recommended readings

- Should Computer Scientists Experiment More? 16 Excuses to Avoid Experimentation
- Introducing empirical software engineering methods in education

Bibliography

OCL

- UML Standards 1.* and 2.0 (include OCL)
  - [http://www.omg.org](http://www.omg.org)

- The Object Constraint Language: Getting Your Models Ready for MDA (2nd Ed.)

- Object Modeling with the OCL: The Rationale Behind the Object Constraint Language

- The Object Constraint Language: Precise Modeling with UML
Bibliography

Experimentation

- Empirical Methods and Studies in Software Engineering

- Lecture Notes on Empirical Software Engineering

- Empirical Studies in Software Engineering

- Basics of Software Engineering Experimentation

- Experimentation in Software Engineering: An Introduction

Software metrics

- Software Metrics: A Rigorous & Practical Approach
Bibliography
Statistics

- Electronic Statistics Textbook (StatSoft, Inc.)

- A New View of Statistics (Will G. Hopkins)
  - http://www.sportsci.org/resource/stats/

- PROPHET StatGuide (Glossary)
  - http://www.basic.nwu.edu/statguidefiles/sg_glos.html

- Introdução à Teoria dos Testes em Ciências Humanas