Experimental Software Engineering
@JOIN'2005

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
- QUASAR Research Group
- 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**?

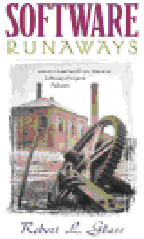
Why?
Abstract

- The Software [Engineering] Crisis
- The Scientific Method
- What is Experimental Software Engineering?
- Obstacles to Experimentations
- 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

Robert L. Glass: *Software Runaways: Monumental Software Disasters*, Prentice Hall PTR, 1998, ISBN: 0-13-673443-X



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"

- Object-oriented systems are more maintainable than procedural systems
- Software inspections are more efficient than testing
- Cohesion should be maximized and coupling should be minimized
- Accurate effort estimates can be produced without a detailed design (e.g. Function Points analysis)
- 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?

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 an 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

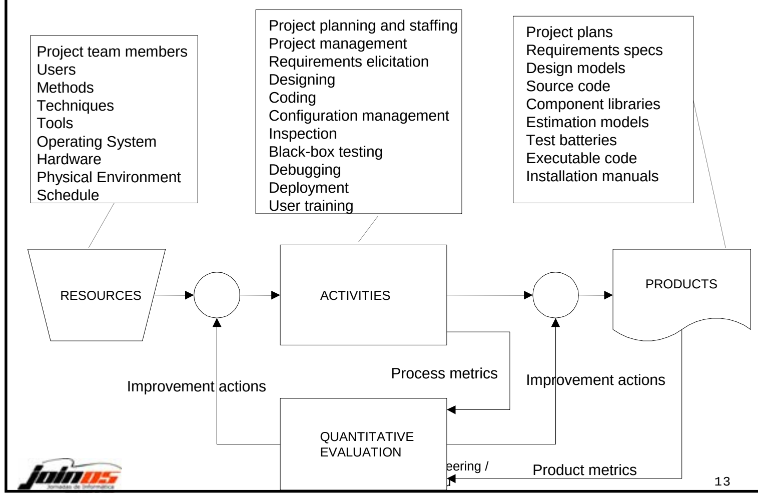
Conclusion:

- in Software Engineering we miss experimentation!

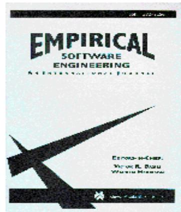
What is … Experimental Software Engineering ?

- Is a branch of Software Engineering where, by means of experimentation we want to validate hypothesis 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

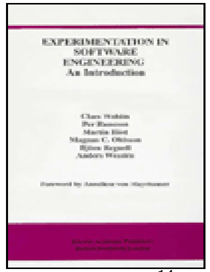
Sw development as a feedback loop …



Some landmark references?



- A journal ...
  - **Empirical Software Engineering:** *An International Journal*
  - http://www.kluweronline.com/issn/1382-3256
- ... and a book:
  - **Experimentation in Software Engineering: An Introduction** Claes Wohlin et al., Kluwer Academic Publishers, November 1999, ISBN: 0792386825



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/
- UNIVERSITY OF MARYLAND (EUA)
  - ESEG – Experimental Software Engineering Group
  - http://www.cs.umd.edu/projects/SoftEng/tame/

+ Competence centers in ESE

IESE
Fraunhofer Institut Experimentelles Software Engineering

- 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)**

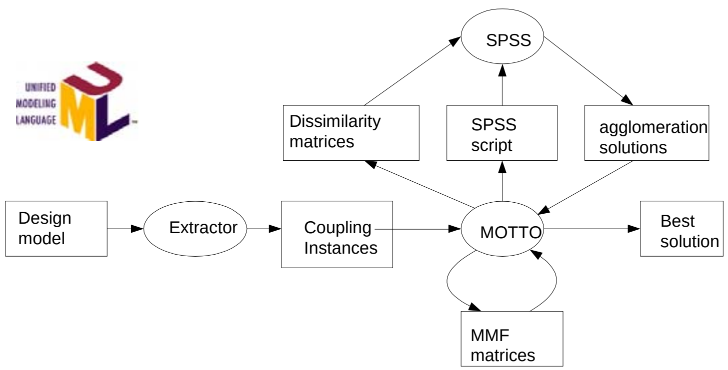
A brief overview of some case studies

- 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

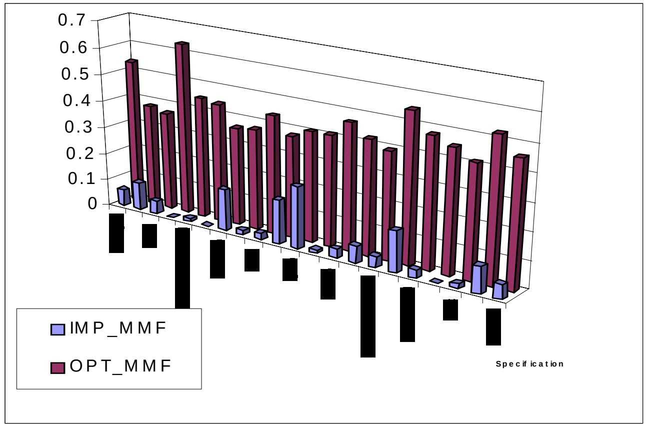
Case study 1: Motivation
Software refactoring

- **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



Case study 2:
Design Complexity Evaluation

Problems:

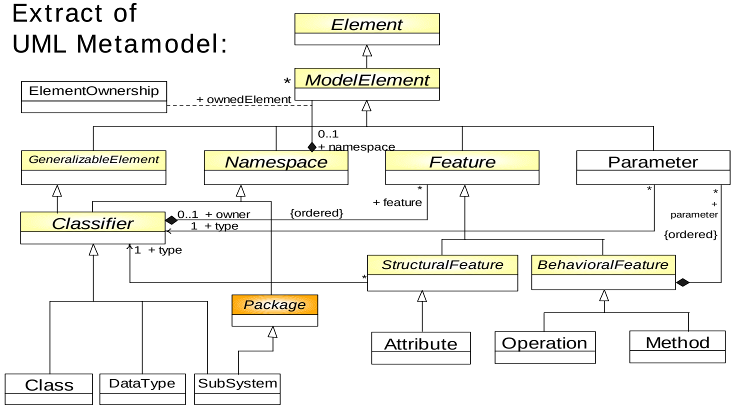
- Code metrics are too late!
- Formalization is required, but ...
  - **CSCO - C**ount of **S**ynchronization-based **C**oupled **O**bject types [Poels2001]

CSCO(P) =

$\#\{Q \in T - \{P\} \mid \exists e \in A: (\tau_1(e, P)=C \wedge \tau_1(e, Q)=E) \vee (\tau_1(e, P)=E \wedge \tau_1(e,Q)=C)\}$

Case study 2:
Design Complexity Evaluation

Extract of
UML Metamodel:



Case study 2:
Design Complexity Evaluation

***NOC( )** -- Number of Children*

Informal: number of classes inhering directly from current class

<u>Classifier:: NOC( ): Integer</u>

post: result = children()->size()

where

children(): Set(GeneralizableElement) = self.generalization -> collect(g | g.parent) -> excluding(self)->asSet

Case study 2:
Design Complexity Evaluation

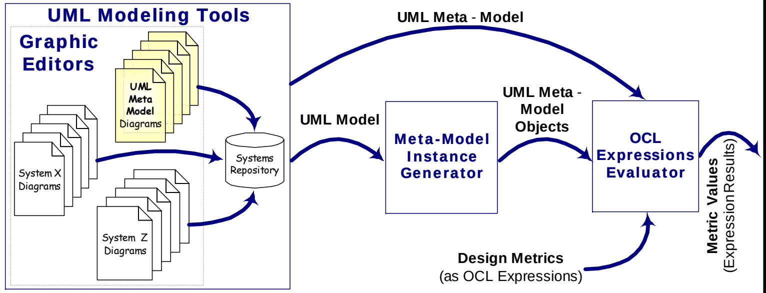
DIT( ) *– Depth in the Inheritance Tree*

Informal: Size of the longest inheritance chain until a root class

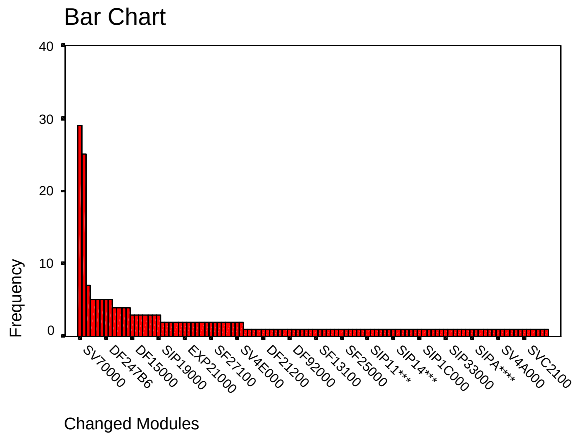
Classifier:: DIT (): Integer =

if self.isRoot then 0
else
if PARN() = 1 then
1 + self.parents()->iterate(elem: GeneralizableElement; acc:Integer = 0 | acc + elem.oclAsType(Class).DIT())
else
self.parents()->iterate(elem: GeneralizableElement; acc:Integer = 0 | acc + elem.oclAsType(Class).DIT())
endif
endif

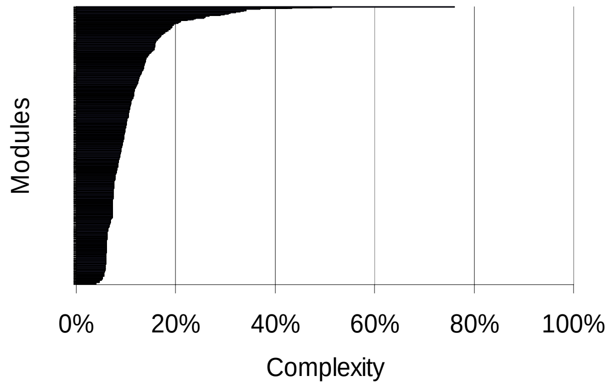
Case study 2:
Design Complexity Evaluation



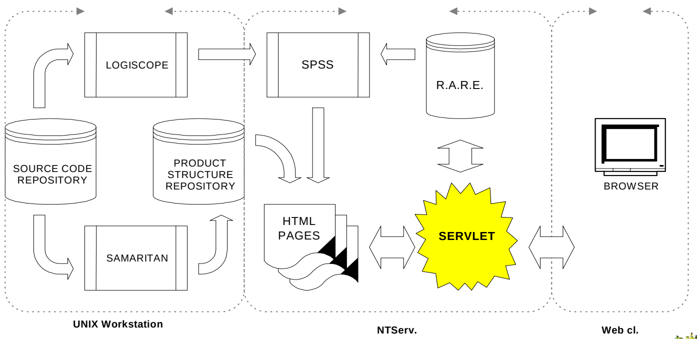
Case study 3:
Controlling the Evolution of Legacy Systems



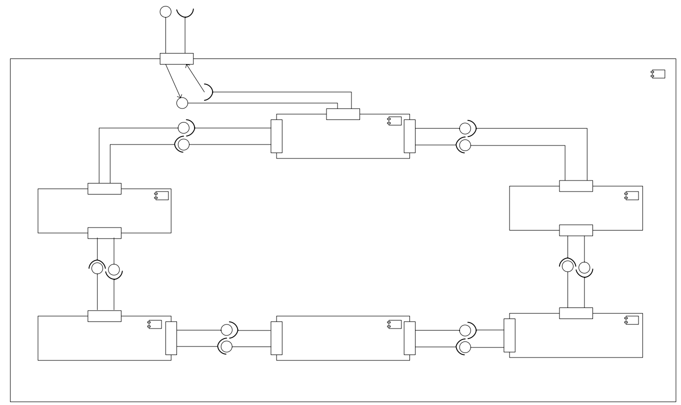
Case study 3:
Controlling the Evolution of Legacy Systems



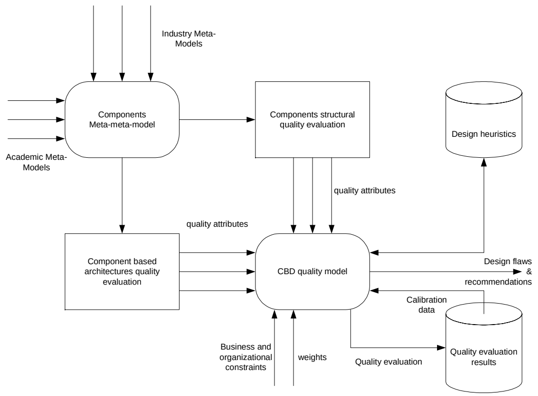
Case study 3:
Controlling the Evolution of Legacy Systems



Case study 4:
Evaluating components and component assemblies



Case study 4:
Evaluating components and component assemblies



Back to the summary …

- Introduction to ESE
- Teaching ESE
- QUASAR Research Group
- 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!
- Which kind of experiments do we propose?

Work assignment A:

Hypothesis:
"Top American universities have more elaborated sites than top European universities"

Steps:
- Produce/adapt a MM in UML for web site architecture and contents (export in XMI format)
- Define/adapt a set of complexity metrics for web site architecture and contents and formalize them in OCL
- Select a representative sample of top American and European university sites and capture the info on their architecture and contents with a crawler
- Generate relevant meta-data (objects and relations) out of the captured info (USE tool "cmd" format) and load the MM with them
- Obtain the metrics for each site and export them to a statistical tool
- Test the hypothesis
- Write a technical report describing each of the steps above

Work assignment B:

Hypothesis:
"Eclipse plug-ins are all of similar pluggable complexity"

Steps:
- Produce/adapt a MM in UML for eclipse plug-in interfaces by reverse engineering of the corresponding XSD or DTD file (export in XMI)
- Define/adapt a set of complexity metrics for eclipse plug-ins interface complexity and formalize them in OCL
- Collect as much plug-in interface definition files (XML files) as possible
- Generate relevant meta-data (objects and relations) out of the XML files (USE tool "cmd" format) and load the MM with them
- Obtain the metrics for each of the plug-ins and export them to a statistical tool
- Test the hypothesis
- Write a technical report describing each of the steps above

Work assignment C:

Hypothesis:
"Academic presentations have more visual clarity than industry ones"

Steps:
- Produce/adapt a metamodel (MM) in UML for presentation contents and format (export in XMI format)
- Define/adapt a set of complexity metrics for presentations' visual clarity and formalize them in OCL *(hints in PowerPoint: see option "Tools / Options / Spelling and Style / Style Options / Visual Clarity")*
- Select a representative sample of academic and industry presentations with a search engine (e.g. ppt files found in ".edu" and ".com" web-sites)
- Generate relevant meta-data (objects and relations) out of the captured files (USE tool "cmd" format) and load the MM with them
- Obtain the metrics for each of the presentations and export them to a statistical tool
- Test the hypothesis
- Write a technical report describing each of the steps above

Work assignment D:

Objective:
"Construct, calibrate and validate a multivariate linear or non-linear correlation model for effort and schedule estimation at the OO design phase"

- **Sample**: a set of Java projects collected in the web
- **Dependent variables**: effort and schedule for model calibration
  - COCOMO II estimates extracted from source code, considering an hypothetical scenario for work environment conditions
- **Independent variables**: MOOD2 metrics (OCL definitions available)
  - extracted from UML design models
  - UML design models are reengineered from Java code
- **Validation:** performed by using the "*Jack Knifing"* technique

Some alternative assignments …

- Replicate our experiments on modularity assessment and reengineering with a different sample
- Compare the coupling and cohesion of systems built with or without design patterns
- Build a defect estimation model based upon past data on defects and system evolution
- Assess to what extent collecting additional software metrics is required
- Evaluate the amount of reuse for given systems
- Verify the distribution of structural design complexity at class and package level (using C&K and MOOD metrics, respectively)
- Verify the independence of UML design metrics

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

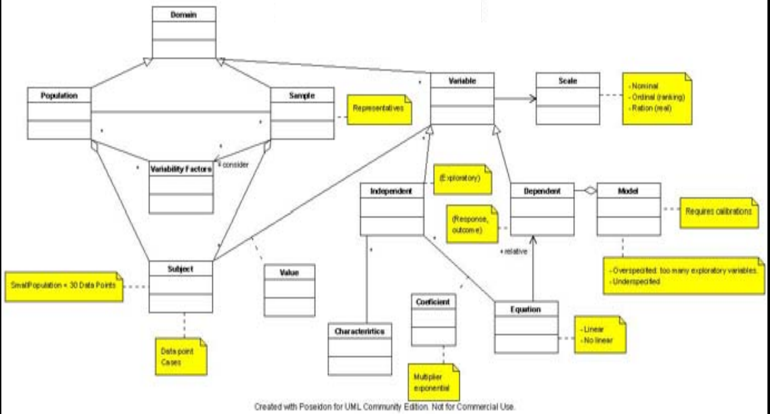
A proposed curricula on ESE

- 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?

An ontology of data analysis …



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

A proposed curricula on ESE

- 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 (eg: 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:
  - University of Maryland (USA)
  - Oregon State University (USA)
  - Florida Atlantic University (USA)
  - Lund University (Sweden)
  - Kaiserlautern University (Germany)
  - NTNU (Norway)
  - Technical University of Sydney (Australia)
  - ... (Japan)

Back to the summary …

- Introduction to ESE
- Teaching ESE
- QUASAR Research Group
- Wrap up

The 6 'W's …

*"I have six loyal servants
(that taught me everything I know);
their names are: what, why and
when, how, where and who"*

*Rudyard Kipling
(Literature Nobel Prize, 1907)*

- **W**here?
- **W**ho?
- **W**hen?
- **W**hy?
- **W**hat?
- ho**W**?

Where?

Who?

The forest and its trees …

- Lisbon New University
- Faculty of Sciences and Technology
- Computer Science Department
- QUASAR Research Group
  - (QUASAR = **QU**antitative **A**pproaches on **S**oftware Engineering **A**nd **R**eengineering)

Who?
Within QUASAR ...


Miguel Goulão


Aline Baroni


FBA


Sérgio Bryton


Ricardo Santos


Eduardo Miranda


Sofia Brás


Luís Ochôa


Pedro Catelas


Rita Esteves

When?

- **1993 - 1998**
  - MOOD Team
    - Software Engineering Group / INESC
- **1996 - …**
    - Pioneers in the introduction of ESE practices in the Portuguese software industry (e.g. Portugal Telecom, Portuguese Navy, etc)
- **1999 - …**
  - QUASAR Research Group
    - DI/FCT
- **2002 - …**
  - We started at FCT/UNL the first systematic effort to teach ESE at both the graduate and undergraduate levels in Portugal

Why?

QUASAR Mission Statement

“We want to contribute to the progress in the state of the art of experimental software engineering, as well as to disseminate best practices in experimentation, both in academia and industry”

What?
Our research threads

- Evaluation of CB architectures (PhD thesis)
- Assessment of UML 2 behavioral design (PhD thesis)
- Modularity improvement with aspects (SOFTAS Project)
- Ontology-based requirements elicitation (STACOS Project)
- Metamodel enlightening support (MSc thesis)
- Evaluation of contracts complexity
- Evaluation of web-sites complexity
- Visualization of quantitative Software Engineering data
- Estimation models for OO development
- Legacy systems reengineering (e.g. modularity improvement)

hoW?

Our strategy

- Results dissemination
  - Continuing participation in international scientific events, either as organizers or active role participants
- Knowledge transfer
  - Producing academic materials in the areas of software quality and experimental software engineering
- Automated software engineering
  - Producing tools to support the conception of controlled experiments.
- Increase participation in research networks
  - Foster idea cross-fertilization, searching for complementary research strengths and interchanging researchers.
  - In 2003 we joined the ESERNET

Back to the summary …

- Introduction to ESE
- Teaching ESE
- QUASAR Research Group
- Wrap up

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