MAP-I
Programa Doutoral em Informática

Source Code Analysis and Manipulation

Unidade Curricular em Paradigmas da Computação
Paradigms of Computation
(UCPC)

UMinho, FEUP

14 de Maio de 2010

Resumo

This document describes a Ph.D. level course, corresponding to a Curriculum Unit credited with 5 ECTS. It corresponds to a joint UMinho-FEUP proposal for UCPC (Paradigms of Computation) in the joint MAP-i doctoral program in Informatics, organized by three Portuguese Universities (Minho, Aveiro, and Porto).

LECTURING TEAM

UMinho:	João P. Fernandes, João Saraiva
FEUP:	João Cardoso, Rui Maranhão
Coordinator:	João Saraiva

A. Programmatic Component

1. Theme, Justification and Context

Motivation

This course concerns the *analysis* and/or *manipulation* of the *source code* of computer systems. While much attention in the wider software engineering community is properly directed towards other aspects of systems development and evolution, such as specification, design and requirements engineering, it is the source code that contains the only precise description of the behaviour of the system. Thus, the analysis and manipulation of source code remains a pressing concern and an active area of research.

In this course, we will use the term *source code*, to mean any description of a software system, being it a fully executable system or a non-executable one. Thus, it includes machine code, very high level language-based program specifications (for example, attribute grammars), general purpose programming languages (ie, Java, Scala, C, etc), domain specific languages (ie, MatLab, SQL, VHDL, etc), document programming languages (like, HTML/XML, LaTeX, etc), end-user programming environments (like, Spreadsheet systems), etc. The term *analysis* is taken to mean any automated or semi automated procedure which takes source code and yields insight into its meaning. The term *manipulation* is taken to mean any automated or semi-automated procedure which takes and returns source code.

This course encompasses many research areas, namely:

1. *Program Refactoring*: refactoring is a controlled transformation technique for improving the design of an existing code base. Its essence is applying a series of small behavior-preserving transformations.

2. *Program Specialization/Partial Evaluation*: Partial evaluation is a transformation technique in which a program is specialized to a part of the input that is known statically (at specialization time), where the specialized program may be much faster than the general one.

3. *Program Optimization*: A program optimization is a transformation that improves the run-time and/or space performance of a program. Examples of optimization are program fusion, inlining, constant propagation, common-subexpression elimination, and dead code elimination.

4. *Program Testing*: Testing is an important process that is performed to support quality assurance. These activities consist of designing test cases, executing the soft-

ware with those test cases, and examining the results produced by those executions.

5. *Program Slicing*: program slicing is the computation of the set of source code program statements, the source program slice, that may affect the values at some point of interest.

6. *Program Debugging and Fault Localization*: Program debugging is a methodical process of finding and reducing the number of defects (or faults) in software. Automatic fault localization techniques aid developers/testers to pinpoint the root cause of software faults, thereby reducing the debugging effort.

7. *Software Metrics*: Software metrics are used to indicate the complexity of a program. Software metrics measure measures program sizes, cohesion, number of linearly independent paths through a program's source code, etc.

8. *Document Analysis and Transformation*: Techniques for analysis and transformation of documents, described via Extensible Markup Languages like XML, plays an important role in the exchange of a wide variety of such documents on the Web.

9. *Generative Programming*: The purpose of generative programming is to replace manual search, transformation/adaptation, and assembly of components with the automatic generation of needed components on demand.

10. *Reverse Engineering*: The goal of reverse engineering is to infer a high-level specification (like, for example, the program data flow, the GUI behavioural model of an interactive application, the program documentation, etc) from a low-level program.

The course will emphasize the study of the analysis and manipulation techniques themselves, namely:

1. Generalized Top-Down and Bottom-Up Parsing Techniques.

2. Strategic Programming.

3. Bidirectional Transformations.

4. Program Calculation.

5. Attribute Grammars.

6. Techniques for Domain Specific Language Embedding.

7. Fault Localization techniques.

8. Program Transformations for embedding Systems.

It is our goal to address the research challenges in the above mentioned research areas, presenting the state-of-the-art in these fields. Moreover, we will invite several well-known researchers to give presentations of their work within the frame of this course (see section *Invited Talks*).

Course Context

ACM Computing Classification System subjects covered:

- D. Software / D.2 Software Engineering / D.2.1 Requirements/Specifications
- D. Software / D.2 Software Engineering / D.2.4 Software/Program Verification
- D. Software / D.2 Software Engineering / D.2.5 Testing and Debugging
- D. Software / D.3 Programming Languages / D.3.3 Language Constructs and Features
- D. Software / D.3 Programming Languages / D.3.4 Processors

2. Objectives and Learning Outcomes

The goal of this advanced course is to introduce the foundations of program analysis and transformation in software engineering.

We will present modern approaches to perform program analysis, namely the use of advanced generic parsing techniques, source code software metrics, source level testing techniques, and fault localization techniques. We will also present advanced techniques for program transformation, like the use of strategic programming, bidirectional program transformation techniques, modern attribute grammar extensions for program transformation, and the use of program calculational approaches to define such transformations.

We will study the use of these approaches not only for general purpose languages, but also for domain specific languages and document specification formalisms. Furthermore, we will discuss source code analysis and transformation techniques for both general and embedded computing.

At the end of this course, PhD students should be able

- To understand context-free and attribute grammars, and to specify powerful analysis and transformation source code systems within these two formalisms.

- To use source level software analysis techniques to extract high level abstract models from a (low level) software description, and to assess its quality;
- To use program refactoring and program slicing techniques to better understand low and high level source code programs;
- To identify hot-spots and the possible code optimizations and transformations that can be applied;
- To understand the impact of code transformations and optimizations in execution time, and in memory and power/energy consumption;
- to recognize mathematical properties in software and to transform and optimize such "naively"written programs by calculation;
- to advocate (semi-)automated software debugging techniques to quickly identify faulty software components;
- to be able to create test suites that are particularly suitable for automated software debugging;
- To acquire knowledge about embedded computing optimizations;
- To define, analyse and transform documents;
- To understand the challenges on developing tools to perform sophisticated code analysis, transformations and optimizations;

3. Course Structure and Contents

This course is structured in five units: the first two present the foundations of source code analysis and transformations. The next unit will present a formal setting where such analysis and transformations are defined. In unit 4, we will present analysis techniques for debugging and fault localization in software systems. Finally, in unit 5, we will study analysis and transformations of source code in the context of embedded computing.

Next, we present the content of each of these units.

1. **Source Code: Analysis**

   In general, software analysis extracts arbitrary properties of software source code. In this unit we will study the foundations of analysis of source code. We will present generic language technologies to analyse source code. These techniques are language paradigm and domain independent: they can be applied and reused to analyse source code defined in different programming languages (C, Haskell, XML), paradigms (OO, Functional, Imperative), and domains (general purpose or domain specific).

We start by studying powerful generic parsing techniques to rapidly define a parser/syntactic analyser for source code. We will also present the attribute grammar formalism in order to define a semantic analyser for programming languages. Domain specific Languages (DSL) will be used as case studies. Moreover, we will study the embedding of DSLs in general purpose languages. The languages for grammar specification (context-free and attribute grammars) will be used to introduced Embedded DSLs.

Software metrics are a special kind of analysis focused on the structure of the source code. Typical metrics report provide details on individual modules and summaries for subsystems. Such metrics are widely used to judge the quality of source code, enabling a software organization to more effectively focus its attention on the lower-quality portions of their portfolio. Software metrics are necessarily computed on the structure of the source code. This means metrics must be extracted from a parse of the program's source code.

We will also study software reverse engineering as yet another analysis technique which has two goals: first, to identify the system's components and their interrelationships; and, second to create representations of the system at a higher level of abstraction.

Finally, we will study program testing as an important analysis technique: reverse engineering techniques are used to extract a model of the underlying source code program, and test cases are (automatically) generated for the model. Model-based testing techniques are presented to check whether the model conforms with the program's implementation.

This unit focuses on the following topics:

- Generalized Parsing: Top-Down Generalized Parsing, Functional Parser Combinators, Bottom-Up Generalized Parsing, Disambiguation Filters, Scannerless Parsing.
- Domain Specific Languages: What is a DSL? When and how to define DSLs? How to implement DSLs? Building Domain Specific Processors for DSLs versus Embedded DSLs. A Parser Combinator Library as an embedding of BNF in a general purpose language.
- Attribute Grammars (AG) and their Extensions: Definition and implementation. Implementation of AGs in a strict/static and dynamic/lazy programming Language. Program analysis via Attribute Grammars. AG-based Pro-

gram transformation via the Higher-Order AG extension. The embedding of AG specification language (ie, a DSL) in Haskell.

- Source Level Software Metrics: Metrics Catalogue (from very simple Source Lines of Code to more complex measures such as Cyclomatic Complexity measurements). Computing metrics for GPLs and DSLs. Using metrics to detect bad smells. Assessing software quality via metrics.
- Reverse Engineering: software architecture extraction. Program model extraction (GUI behavioural models, spreadsheet business logic, program dependency graph). Document generation.
- Source Level Software Testing: Test case generation. Model-based testing. Testing automation. Performance testing for embedding systems.

2. **Source Code: Manipulation**

Program Analysis reduces a program to one aspect such as its control-flow. Thus, analysis can be considered a transformation to a sub-language. In this unit we will present generic techniques and tools for program transformation.

One of the aims of a general framework for program transformation is to define transformations that are reusable across as wide a range of languages as possible. Thus, we present strategic programming as a generic language-based technique for source code software transformation. We will also study bidirectional program transformations. A Bidirectional transformation defines two program transformations: one from a source program to a target program, and a second one from the target to the source program. A bidirectional transformation system allows users to edit/update both the source and the target programs: the two programs (source and target) are automatically synchronized after the user update.

In this course we present program refactoring as an important source code transformation: a technique for changing the structure of a program without changing its behaviour/semantics.

We will also study partial evaluation or program specialization as an important program optimization technique: program are specialized to a part of the input that is known statically. Two real examples will be studied: the specialization of the acceptance functions for non-deterministic finite automata and for table-driven top-down parsers.

In order to analyse/understand large software systems, it is convenient to focus on smaller parts of the system. Program slicing is a technique that allows us to slice out fragments of a larger software system that deserve our attention. We will present slicing techniques (backward and forward slicings) as algorithms that work on source code program dependency graphs.

Finally, we will present document analysis and transformation techniques based on the widely used extensible markup languages. We will relate these document techniques to context-free grammars and strategic programming.

The unit is structured into the following topics:

- Strategic Programming: Foundations. Rewriting systems. Strategic Combinators. Strategies across programming paradigms: functional, OO and imperative incarnations. Strategic-based source code transformation systems: TOM and RASCAL systems.
- Bidirectional Program Transformation: Backward and forward transformations. Deriving backward from forward transformation definition. Bidirectional attribute grammars.
- Program Refactoring: Source code refactorings. Refactorings catalogues. Bad smell elimination. Tool support for refactoring (The Hare refactorer)
- Program Specialization/Partial Evaluation: Foundations. Futamura projections. Partial evaluation as an optimization transformation. Study of two case studies.
- Program Slicing: Program dependency graph. Static versus dynamic slicing. Slicing Criteria. Forward and backward slicing, and chopping. Source code slicing systems: the CodeSurfer and the GuiSurfer systems.
- Document Definition and Transformation: Extensible markup languages. The XML Language. XML schema. XML Schema versus Context-Free Grammars. Document Transformation via XSLT. XSLT versus Strategic Combinators.

3. **Rigorous Approaches to Program Analysis and Transformations**

As software becomes more and more complex, it is more and more important to structure it well. Well-structured software is easy to write, easy to debug, and provides a collection of modules that can be re-used to reduce future programming costs. Conventional languages place conceptual limits on the way problems can be modularised. Functional languages push those limits back.

In this module, we show that two features of functional languages in particular, higher-order functions and lazy evaluation, can contribute greatly to modularity.

Furthermore, functional programs are "referentially transparent", which means that an expression can be replaced by its value without affecting the whole program. This freedom helps make functional programs more tractable mathematically than their conventional counterparts.

In this module, we will explore the mathematical properties of functional programs, and we will analyse and transformation such programs in a way that the correctness of the transformations can be formally established.

In this unit, we will focus on the following topics:

- Recursion Patterns: Definition and Identification in the source code of a (functional) program.
- Transforming functional programs by Calculation.
- Improving the efficiency of programs using Fusion Rules.
- Introducing Circular Programming, exploring their nice properties in program transformations.

4. **Program Debugging and Fault Localization**

   Software reliability/quality can generally be improved through extensive testing and debugging, but this is often in conflict with market conditions: software cannot be tested exhaustively, and of the bugs that are found, only those with the highest impact on the user-perceived reliability can be solved before the release. In this typical scenario, testing reveals more bugs than can be solved, and debugging is the bottleneck for improving software reliability/quality. Automated debugging techniques can help to reduce this bottleneck. These techniques give a diagnosis for failures that are detected during the execution of a program, which can help programmers to locate their root causes, and thus to reduce the effort spent on manual debugging.

   The unit is structured into the following topics:

   - Devise proper test suites which help in the subsequent debugging phase.
   - (Semi-) automatic fault localization.
   - Introducing to test sequencing for improving the time spent on debugging.

5. **Source Code Analysis and Transformations for Embedded Computing**

   Code transformations are of paramount importance in embedded computing. They may reduce execution time, power dissipation, and/or energy consumption.

   This module will focus on the analysis and identification of code optimization techniques best suited to achieve reductions related to those aspects on typical applications. As case studies, we will evaluate code transformations with applications running on mobile devices.

   Finally, challenges on the development of code transformation tools for embedded systems will be presented and possible solutions discussed.

   - Embedded Computing Idiosyncrasies.
   - Computing Engines used in Embedded Systems.

- Profiling, Performance Analysis and Amdahl's Law.
- Introduction to Power Dissipation and Energy Consumption.
- Source Code Transformations and Optimizations.
- Challenges regarding Tools for Code Transformations: Examples and Case Studies.

4. Teaching Methods and Student Assessment

This course will consist of theoretical and practical components.

In the theoretical component, a set of seminars will be delivered by the lecturing team, invited speakers, and the students themselves.

- The basic and advanced contents of this course will be presented by the lecturing team; We will be using the proceedings of the series of Ph.D. summer schools on *Generative and Transformation Techniques in Software Engineering* (GTTSE) as the reading material for the classes (see Bibliography section). The proceedings of the first three editions include several long and short tutorials (concretely 22 long tutorials and 19 short tutorials), written by well-known researchers, which cover most of the topics in the course. The members of the lecturing team are involved in the series of GTTSE Ph.D. schools as proceedings editors, local organizers and tutorialists.
- Advanced contents on specific topics will be presented by visiting researchers (see section *Invited Tutorials*)
- Students will give one presentation of proposed research papers. The presentation will follow a standard conference talk model. The research papers will be proposed by the lecturing team, and they consist of a selection of recently published papers in conferences and journals in the SCAM area (namely, PLDI, TOPLAS, SCAM, GPCE, and SLE).

The practical component consists of the development of an individual project. This project requires the use of a specific software system for the analysis and transformation of source code. The students are also expected to give a presentation, in the form of a tool demo, of the software system they decide to use. They are also supposed to write a report of the project as a research paper.

Invited Tutorials

We plan to have a series of tutorials given by researchers working in the field of SCAM. In the next scholar year we plan to have the following tutorials:

- Eric Van Wyk is an Associate Professor at Univ. Minnesota, USA. Prof. Eric Van Wyk will be visiting UMinho in the scholar year 2010-2011 (from October 2010 till June 2011) in the context of a sabbatical leave. His sabbatical research plan includes the study of "Bidirectional Transformation for Attribute Grammars". Thus, we will give a tutorial in the unit "Source Code: Transformations".

- Alberto Pardo is an Associate Professor at the Computing Science Department (InCo) of the Engineering School, Universidad de la República, Montevideo, Uruguay and we will be visiting UMinho in the context of the *Erasmus Mundus 17* exchange program, and as consultant of the FCT funded SSaaPP research project. We will give a tutorial in the unit "Rigorous Approaches to Program Analysis and Transformations", being this a research area that Alberto Pardo has already significantly contributed to, including with some of the lecturing team members.

- Janis Voigtlander is a Professor at Bonn University, Germany, conducting research on parametricity results (also called free theorems) for polymorphically typed languages. These results are used, for example, to establish formally the correctness of program transformations. In the context of the project *Strictification of Circular Programs*, (FCT/DAAD bilateral agreement 2010-2011), one visit is scheduled to UMinho, where Janis Voigtlander will give a tutorial in the unit "Rigorous Approaches to Program Analysis and Transformations".

- Ali Mesbah is a postdoctoral researcher at Delft University of Technology, the Netherlands. His research is about automatic testing of web systems. He has recently been awarded with the distinguished paper award at ICSE'09.

Student Assessment

During the this course the students will have to give a research talk, to develop a research project, to give a tool demo talk presenting the software system they adopt, and finally to write the project report as a scientific paper. Thus, the student will be assessed through these four activities.

5. Basic Bibliographic References

- *Generative and Transformation Techniques in Software Engineering I, II, III*, Ralf Laemmel, Joost Visser and João Saraiva editors, volumes 4143 and 5235 of LNCS Tutorials, proceedings of the summer schools GTTSE'05, GTTSE'07 and GTTSE'09 (to appear), Springer.

- *Compilation Techniques for Reconfigurable Architectures*, João Cardoso and Pedro Diniz, Springer Publishing Company, Incorporated, 2008.

- *Software Product Lines: Practices and Patterns*, Paul Clements and Linda Northrop, Addison-Wesley, August 2001.
- *Generative Programming - Methods, Tools, and Applications*, Krzysztof Czarnecki and Ulrich W. Eisenecker, Addison-Wesley, June 2000.
- *Partial Evaluation and Automatic Program Generation*, N.D. Jones, C.K. Gomard, and P. Sestoft, Prentice Hall International, June 1993
- *The Fun of Programming* , Jeremy Gibbons and Oege de Moor, editors. Cornerstones in Computing. Palgrave, 2003.
- *Domain Specific Languages*, Martin Fowler, Addison-Wesley Professional, September, 2010 (Estimated), already available at safari books online)
- *Refactoring: Improving the Design of Existing Code*, Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, Addison-Wesley, 2000.

B. Lecturing Team

1. Team Presentation

This course is supported by a team involving researchers form both the University of Minho, School of Engineering (João Paulo Fernandes and João Saraiva), and the University of Porto, FEUP (João Cardoso and Rui Maranhão).

All team members are working, and have worked actively in the past few years, on topics that are directly related to the subjects covered by this course, as detailed below.

2. Coordinator

The coordinator of the unit is João Saraiva.

3. Short Presentation of Team Members

In the sequel we introduce a brief presentation of each team member, which includes, for each of them, up to 5 key publications related to the scientific area in which this course is proposed.

João Saraiva is auxiliar professor at the Department of Informatics, Universidade do Minho, and a researcher member of CCTC. He obtained a MSc degree from University do Minho in 1993 and a Ph.D. degree in Computer Science from Utrecht University in 1999. His main research contributions have been in the field of program language design and implementation and in program analysis and transformation. He counts with over 50 international publications in this area and he has served in over 25 programme committees of international events.

He has experience in participating, coordinating and evaluating research projects in this area, both at national level with projects funded by FCT (projects: PURe, IVY, AMADEUS, CROSS, and SSaaPP) and at international level with projects: "Embedded Attribute Grammars"funded by EPSRC - The Engineering and Physical Sciences Research Council, UK, contract GR/S02266/01 (project developed at Oxford Computing Laboratory with Prof. Dr. Oege de Moor), and "Applied Semantics II"an European Union Thematic Network, EU contract IST-2001-38957 (where the he was the site coordinator). He was in the evaluation committees of the grant agencies: ANII *Agencia Nacional de Investigación e Innovación*, Uruguay (Fondo Clemente Estable 2007 (FCE 2007), and NWO - Netherlands Organisation for Scientific Research (Physical Sciences division of the Free

Competition, in astronomy, computer science or mathematics, 2009). Currently, he serves in the programme committee of GPCE'10, SBLP'10 , SBConf'10, and as an external reviewer of "Annual Prize IBM Belgium of informatics /F.R.S./FNRS 2010".

João Saraiva is one of the founders of the successful series of summer schools on Generative and Transformational Techniques in Software Engineering (GTTSE), organized in 2005, 2007 (volumes 4143,and 5235 of LNCS by Springer-Verlag) and 2009 (LNCS to appear), in Braga. The fourth instance of the school - GTTSE 2011, is already planned for July 2011 in Braga. He was the organizing chair of ETAPS'07, The European Joint Conferences on Theory and Practice of Software, organized in Braga in 2007, and he currently serves as treasurer on its steering committee.

Key Publications:

- J.C. Silva, C. Silva, R. Goncalo, João Saraiva, J.C. Campos, *The GUISurfer tool: towards a language independent approach to reverse engineering GUI code*, ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2010), June 2010 (accepted).

- Jácome Cunha, João Saraiva and Joost Visser, *Discovery-based Edit Assistance for Spreadsheets*, 25th IEEE Symposium on Visual Languages and Human-Centric Computing (VL-HCC'09), Corvallis, Oregon, September 2009, IEEE Press.

- João Carlos Silva, João Saraiva, Jose Creissac Campos, *A Generic Library for GUI Reasoning and Testing*, 24th Annual ACM Symposium on Applied Computing (SAC 2009), Honolulu, Hawaii, USA March 8 - 12, 2009, ACM Press.

- Jácome Cunha, João Saraiva and Joost Visser, *From Spreadsheets to Relational Databases and Back*, ACM SIGPLAN Symposium on Partial Evaluation and Program Manipulation (PEPM 2009), Savannah, Georgia, USA, January 19-20, 2009, ACM Press.

- Don S. Batory, Maider Azanza, João Saraiva, *The Objects and Arrows of Computational Design*, 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2008), Toulouse, France, September 2008, volume 5301 of LNCS, pag. 1-20, Springer.

Funded Projects (last 5 years):

- *Strictification of Circular Programs*, FCT/DAAD bilateral agreement 2010-2011. (Principal Investigator, recommended for funding)

- *SSaaPP : SpreadSheets as a Programming Paradigm*, under FCT contract PTDC/EIA-CCO/108613/2008. 2010-2013. (Principal Investigator)

- *CROSS: - An Infrastructure for Certification and Re-engineering of Open Source Software*, under FCT contract PTDC/EIA-CCO/108995/2008. 2010-2013.
- *AMADEUS - Aspects and Compiler Optimizations for Matlab System Development*, under FCT contract PTDC/EIA/70271/2006. 2008-2010.
- *IVY - A model-based usability analysis environment*, under FCT contract POSC/EIA/56646/2004. 2004-2007.
- *LerNet - Language Engineering and Rigourous Software Development*, European Comission ALFA Programme, 2006-2009.
- *PURe - Program Understanding and Re-engineering: Calculi and Application*, under FCT contract POSI/CHS/44304/2002. 2003-2006.
- *APPSEM - Applied Semantics II*, European Union - Thematic Networks Programme, under EU contract IST-2001-38957, 2003-2006. (Site Coordinator)

Supervision of PhD projects:

- Jácome Cunha, *Foundations of Spreadsheets*, FCT grant `SFRH/BD/30231/2006`, University of Minho, since 2006. (PhD supervisor)
- João Carlos Cardoso Silva, *Formal Methods and Reverse Engineering Applied to Interactive Systems*, FCT grant `SFRH/BD/30729/2006`, University of Minho, since 2005. (PhD supervisor)
- João Paulo Fernandes, *Design, Implementation and Calculation of Circular Programs*, FCT grant `SFRH/BD/19186/2004`, University of Minho, defended in March 2009. (PhD supervisor)
- Fábio Tirelo, *Semântica Multidimensional de Linguagens de Programação*, Departamento de Informática, Universidade do Federal de Minas Gerais, Belo Horizonte, Brazil, defended in March 2009. (PhD co-supervisor)

João Paulo Fernandes is a Postdoc researcher at the Informatics Department, University of Minho, a research member of CCTC and a Visiting Professor at the Polytechnic Institute of Porto (part-time position). He received a 5-year Degree in Computer Science in 2004 and a PhD degree in Computation Foundations in 2009, both from the University of Minho. Previously, he has been the Co-Chair of the Department of Systems Management and Information Science and Technology at Universidade Atlântica, a Portuguese private University (in the school year of 2008/2009), and a Visiting Professor at the same University.

João Fernandes has contributed mainly to the research area of program analysis, manipulation and transformation, specially under calculational form, having published several papers in top conferences in this field. He is most interested in exploiting the nice properties of circular lazy programs and their relation with Attribute Grammars. With his research, he has established different international cooperations, having worked at the University of Oxford, UK, de la Republica University, Uruguay and the University of Tokyo, Japan. A research visit to the Max Planck Institute for Software Systems in Germany is also in preparation.

João Fernandes has served in the organizing committee of the Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE'09), and has already served as an external reviewer for international conferences such as LDTA'09, APLAS'09, SLE'09 and PEPM'10.

Key Publications:

- João Paulo Fernandes, *Design, Implementation and Calculation of Circular Programs*, PhD thesis, defended in March, 2009. (The thesis is available as a book, ISBN 3639168968, published by VDM Verlag)
- Alberto Pardo, João Paulo Fernandes and João Saraiva, *Shortcut fusion rules for the derivation of circular and higher-order monadic programs*, ACM SIGPLAN Symposium on Partial Evaluation and Program Manipulation (PEPM 2009), Savannah, Georgia, USA, January 19-20, 2009, ACM Press. (selected paper for a special issue of the Journal HOSC, under reviewing)
- João Paulo Fernandes, Alberto Pardo and João Saraiva, *A shortcut fusion rule for circular program calculation*, ACM SIGPLAN workshop on Haskell (Haskell '07), pag. 95-106, Freiburg, Germany, January 2007, ACM Press.
- João Paulo Fernandes and João Saraiva, *Tools and Libraries to Model and Manipulate Circular Programs*, In Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM07, January 15-16, 2007, ACM Press.

Funded Projects (last 5 years):

- *Strictification of Circular Programs*, FCT/DAAD bilateral agreement 2010-2011. (recommended for funding)
- *SSaaPP : SpreadSheets as a Programming Paradigm*, under FCT contract PTDC/EIA-CCO/108613/2008. 2010-2013.

- *PURe - Program Understanding and Re-engineering: Calculi and Application*, under FCT contract POSI/CHS/44304/2002. 2003-2006.

João Cardoso received a 5-year Electronics Engineering from the University of Aveiro in 1993, and an MSc and a PhD degree in Electrical and Computer Engineering from the IST/UTL (Technical University of Lisbon), Lisbon, Portugal in 1997 and 2001, respectively. He is currently Associate Professor with tenure at the Department of Informatics Engineering, Faculty of Engineering of the University of Porto. Before, he was with the IST/UTL (2006-2008), a senior researcher at INESC-ID (2001-2009), and with the University of Algarve (1993-2006). In 2001/2002, he worked for PACT XPP Technologies, Inc., Munich, Germany.

He has participated in the organization of a number of conferences (e.g., RAW'10, FPL'03, '07, '08, ARC'05, ARC'06, '07) and he serves as a Program Committee member for various international conferences (e.g., IEEE FPT, IEEE SASP, FPL, IC-SAMOS, ACM SAC-EMBS, ARC). He serves(ed) as reviewer for various international scientific journals (e.g., IEEE Transactions on Computers, IEEE Transactions on VLSI, Elsevier Microprocessors and Microsystems, Elsevier Journal of Systems Architecture, IEEE Computer Magazine, IEEE Transactions on Education, Elsevier International Journal on Computers and Electrical Engineering, IEEE Transactions on Industrial Electronics, Elsevier Parallel Computing, IEEE Design & Test of Computers).

He is co-author of a Springer book and co-editor of two Springer LNCS volumes. He has (co-)authored over 80 scientific publications (including journal/conference papers and patents) on subjects related to compilers, embedded systems, and reconfigurable computing. He is a member of IEEE, IEEE Computer Society and a senior member of ACM.

Key Publications:

- Carlos Morra, João M. P. Cardoso, João Bispo, and Juergen Becker, *Retargeting, Evaluating, and Generating Reconfigurable Array-Based Architectures*, in 6th IEEE Symposium on Application Specific Processors (SASP 2008), 8-9 June 2008, Anaheim Convention Center, Anaheim CA, USA, pp. 34–41.

- João M. P. Cardoso, *Dynamic Loop Pipelining in Data-Driven Architectures*, in Proc. of the ACM International Conference on Computing Frontiers (CF'05), Ischia, Italy, 4-6 May 2005, ACM Press, pp. 106-115.

- Rui Rodrigues, João M. P. Cardoso, and Pedro C. Diniz, *A Data-Driven Approach for Pipelining Sequences of Data-Dependent Loops*, in 15th Annual IEEE Symposium on

Field Programmable Custom Computing Machines (FCCM'07), Napa Valley, CA, USA, April 23 - April 25, 2007, IEEE Computer Society Press.

- João M. P. Cardoso, and Markus Weinhardt, *From C Programs to the Configure-Execute Model*, in Proc. of the Design, Automation and Test in Europe Conference (DATE'03), Munich, Germany, March 3-7, 2003, IEEE Computer Society Press, pp. 576-581.
- João M. P. Cardoso, *On Combining Temporal Partitioning and Sharing of Functional Units in Compilation for Reconfigurable Architectures*, in IEEE Transactions on Computers, Vol. 52, No. 10, October 2003, pp. 1362-1375.

Funded Projects (last 5 years):

- Member of the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC), `http://www.hipeac.net/`
- *REFLECT: Rendering FPGAs to Multi-Core Embedded Computing*, Coordinator of WP4 and leader of FEUP partnership, Project Number 248976, FP7-ICT-2009-4, Activity: ICT-2009.3.6 Computing Systems. Duration in months: 36 (start: January 2010).
- *"AMADEUS: Aspects and Compiler Optimizations for Matlab System Development"*, Funded by FCT, PTDC/EIA/70271/2006. (Project coordinator).
- *"COBAYA: closing the compilation gap between algorithms and coarse-grained reconfigurable array architectures"*; Funded by FCT, PTDC/EEA-ELC/70272/2006. (Project coordinator)

Supervision of PhD projects:

- João Cardoso supervised 3 PhD projects and co-supervisor one PhD project.

Rui Maranhão (publishes as Rui Abreu) graduated in Systems and Computer Engineering from University of Minho, Portugal, carrying out his graduation thesis project at Siemens S.A., Portugal. Between September 2002 and February 2003, Rui followed courses of the Software Technology Master Course at University of Utrecht, the Netherlands, as an Erasmus Exchage Student. He was an intern researcher at Philips Research Labs, the Netherlands, between October 2004 and June 2005. He received his Ph.D. degree from the Delft University of Technology, the Netherlands, in November 2009. During his PhD studies period (2005-2009) he was also a research associate at the Embedded Systems Institute, the Netherlands. Currently, he is an assistant professor at the Faculty of Engineering of University of Porto, Portugal.

Key Publications:

- R. Abreu, P.Zoeteweij, and A.J.C. van Gemund, Spectrum-based Multiple Fault Localization. In Proceedings of the 24th International Conference on Automated Software Engeneering (ASE'09), pp. 88–99, Auckland, New Zealand, November 2009. IEEE Society.

- R. Abreu, P. Zoeteweij, R. Golsteijn, and A.J.C. van Gemund, A Practical Evaluation of Spectrum-based Fault Localization. Journal of Systems and Software (JSS), Elsevier, 2009.

- R. Abreu, P. Zoeteweij, and A.J.C. van Gemund, A New Bayesian Approach to Multiple Intermittent Fault Diagnosis. In Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI'09), pp. 653–658, Pasadena, CA, USA, July 2009. AAAI Press.

- R. Abreu, P. Zoeteweij, and A.J.C. van Gemund, On the Accuracy of Spectrum-based Fault Localization. In Proceedings of the Testing: Academia and Industry Conference - Practice And Research Techniques (TAIC PART'07), pp. 89–98, Windsor, United Kingdom, September 2007. IEEE Computer Society.

- T. Janssen, R. Abreu, and A.J.C. van Gemund, Zoltar: A Toolset for Automatic Fault Localization. In Proceedings of the 24th International Conference on Automated Software Engeneering (ASE'09) - Tools Track, pp. 662–664, Auckland, New Zealand, November 2009. IEEE Computer Society. (Best Demo Award)

Funded Projects (last 5 years):

- *SSaaPP : SpreadSheets as a Programming Paradigm*, under FCT contract PTDC/EIA-CCO/108613/2008. 2010-2013.

- *TRADER: Systems Reliability*, Embedded Systems Institute, the Netherlands.