SYLLABUS

1. Information regarding the programme

1.1 Higher education institution	**Babeş Bolyai University**
1.2 Faculty	**Faculty of Mathematics and Computer Science**
1.3 Department	**Department of Computer Science**
1.4 Field of study	**Computer Science**
1.5 Study cycle	**Master**
1.6 Study programme / Qualification	

2. Information regarding the discipline

2.1 Name of the discipline	**Resource Aware Computation**						
2.2 Course coordinator	**Assoc. Prof. PhD. Ing. Florin Craciun**						
2.3 Seminar coordinator	**Assoc. Prof. PhD. Ing. Florin Craciun**						
2.4. Year of study	**1**	2.5 Semester	**2**	2.6. Type of evaluation	**E**	2.7 Type of discipline	**compulsory**

3. Total estimated time (hours/semester of didactic activities)

3.1 Hours per week	3	Of which: 3.2 course	2	3.3 seminar/laboratory	1
3.4 Total hours in the curriculum	42	Of which: 3.5 course	28	3.6 seminar/laboratory	14

Time allotment:	hours
Learning using manual, course support, bibliography, course notes	30
Additional documentation (in libraries, on electronic platforms, field documentation)	30
Preparation for seminars/labs, homework, papers, portfolios and essays	70
Tutorship	14
Evaluations	14
Other activities:	-

3.7 Total individual study hours	158
3.8 Total hours per semester	200
3.9 Number of ECTS credits	8

4. Prerequisites (if necessary)

4.1. curriculum	• None
4.2. competencies	• Basic software development skills • Procedural and Object-oriented paradigms

5. Conditions (if necessary)

5.1. for the course	

6. Specific competencies acquired

Professional competencies	• Understanding and working with basic concepts in software engineering; • Knowledge, understanding and use of basic concepts of theoretical Computer Science • Capability of analysis and synthesis; • Proficient use of methodologies and tools specific tool software systems • Good programming skills in high-level languages
Transversal competencies	• Improved programming abilities: debugging and correcting compilers errors • Ability to apply compiler techniques to different real life problems

7. Objectives of the discipline (outcome of the acquired competencies)

7.1 General objective of the discipline	• To understand fundamental concepts of software quality. • To be able to apply basic methods for software analysis and software quality assurance.
7.2 Specific objective of the discipline	• To learn the methods of program verification and validation. • To become used with building correct programs from specifications • To acquire a modern programming style • To understand how the resources(memory, CPU, batery) are used by the programs

8. Content

8.1 Course	Teaching methods	Remarks
1. Semantics of sequential programs. Procedural paradigm. Object-oriented paradigm. Functional paradigm. Operational semantics. Denotational semantics. Small-Steps Semantics. Big-Steps Semantics.	Exposure,description, explanation, debate and dialogue, discussion of case studies	
2. Semantics of concurrent programs. Concurrency models. Processes & threads Atomic actions, interleaving model. Transition systems & diagrams,Safety and liveness.	explanation, debate and dialogue, discussion of case studies	
3. Semantics of concurrent programs. Critical regions, lock protocols. Barriers, Semaphores, Monitors, Deadlocks	Exposure,description, explanation	
4. Semantics of multicore programs. Cell processors. Parallel architectures. Parallel programming concepts.	Exposure,description, explanation	
5. Semantics of multicore programs. Parallel	Exposure,description,	

programming design patterns. StreamIt language. Parallelizing compilers.	explanation, discussion of case studies	
6. Semantics of resources usage in sequential, concurrent and parallel paradigms. Memory usage models.	Exposure,description, explanation, discussion of case studies	
7. Semantics of resources usage in sequential, concurrent and parallel paradigms. Memory usage models. CPU usage models. Battery usage models.	Exposure,description, explanation,	
8. Static analysis. Principles.Dataflow analysis.Type-based analysis	Exposure,description, explanation	
9. Static analysis. Symbolic execution. Abstract interpretation	Exposure,description, explanation, discussion of case studies	
10. Automatic verification. Hoare logic. Separation logic. Modular verification	Exposure,description, explanation, discussion of case studies	
11. Automatic verification. Theory solvers in SMT. Invariant inference	Exposure,description, explanation, discussion of case studies	
12. Analysis and verification of memory usage. Memory models. Shape analysis. Type based methods. Separation logic methods.	Exposure,description, explanation, discussion of case studies	
13. Analysis and verification of cpu usage. Cost analysis. Loop invariants.	Exposure,description, explanation, discussion of case studies	
14. Analysis and verification of battery usage. Energy-aware programming techniques. Approximate computations. Techniques to control the battery usage	Exposure,description, explanation, discussion of case studies	

Bibliography

1. 1.Benjamin Pierce: Foundational Calculi for Programming Languages, CRC Handbookof Computer Sc and Engineering, 1995.

2.Alex Aiken, Suhabe Bugrara, Isil Dillig, Thomas Dillig, Brian Hackett, and Peter Hawkins. An overview of the saturn project. In PASTE,pages 43--48, 2007.

3.Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. Arecognition algorithm for pushdown store systems.In Proceedings of the 1968 23rd ACM national conference, pages 597--604, New York,NY, USA, 1968. ACM Press.

4.Thomas Ball, Ella Bounimova, Byron Cook, Vladimir Levin, Jakob Lichtenberg, Con McGarvey, Bohus Ondrusek, Sriram K. Rajamani, and Abdullah Ustuner.Thorough static analysis of device drivers.In EuroSys '06: Proceedings of the

2006 EuroSys conference, pages 73--85, New York,NY, USA, 2006. ACM Press.

5.Bruno Blanchet, Patrick Cousot, Radhia Cousot, J\'{e}rome Feret, Laurent

Mauborgne, Antoine Min\'{e}, David Monniaux, and Xavier Rival.

A static analyzer for large safety-critical software.In PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, pages 196--207, New York, NY, USA, 2003.ACM Press.

6.Michael Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and

K. Rustan M. Leino. Boogie: A modular reusable verifier for object-oriented programs.

In FMCO, volume 4111 of Lecture Notes in Computer Science, pages 364--387, 2005.

7.Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte.The spec\# programming system: An overview.In CASSIS, volume 3362 of Lecture Notes in Computer Science, pages 49--69, 2005.

8.Randal E. Bryant.Symbolic boolean manipulation with ordered binary-decision diagrams.ACM Comput. Surv., 24(3):293--318, 1992.

9.Patrick Cousot and Radhia Cousot.Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.In POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium onPrinciples of programming languages, pages 238--252, New York, NY, USA, 1977.ACM Press.

10.Patrick Cousot and Radhia Cousot.Systematic design of program analysis frameworks.In POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium onPrinciples of programming languages, pages 269--282, New York, NY, USA, 1979.ACM Press.

11.Patrick Cousot and Nicolas Halbwachs.Automatic discovery of linear restraints among variables of a program.In POPL, pages 84--96, 1978.

12.Venkatesan T. Chakaravarthy.New results on the computability and complexity of points--to analysis.In POPL '03: Proceedings of the 30th ACM SIGPLAN-SIGACT symposium onPrinciples of programming languages, pages 115--125, New York, NY, USA, 2003.ACM Press.

13.Manuvir Das.Unification-based pointer analysis with directional assignments.In PLDI '00: Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation, pages 35--46, New York, NY, USA, 2000. ACM

Press.

14.Robert DeLine and Manuel Fahndrich.Enforcing high-level protocols in low-level

software.In PLDI, pages 59--69, 2001.

15.David Detlefs, Greg Nelson, and James B. Saxe.Simplify: a theorem prover for program checking.J. ACM, 52(3):365--473, 2005.

16.Jeffrey S. Foster, Robert Johnson, John Kodumal, and Alex Aiken.Flow-insensitive type qualifiers.ACM Trans. Program. Lang. Syst., 28(6):1035--1087, 2006.

17.Stephen Fink, Eran Yahav, Nurit Dor, G. Ramalingam, andEmmanuel Geay.

Effective typestate verification in the presence of aliasing.In ISSTA '06: Proceedings of the 2006 international symposium on Software testingand analysis, pages 133--144, New York, NY, USA, 2006. ACM Press.

18.Sumit Gulwani and Ashish Tiwari.Combining abstract interpreters.In PLDI '06: Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation, pages 376--386, New York, NY, USA, 2006.ACM Press.

19.Brian Hackett, Manuvir Das, Daniel Wang, and Zhe Yang.Modular checking for buffer overflows in the large.In ICSE '06: Proceeding of the 28th international conference on Software engineering,pages 232--241, New York, NY, USA, 2006. ACM Press.

20.Ben Hardekopf and Calvin Lin.The ant and the grasshopper: fast and accurate pointer analysis for millions of lines of code.In PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programminglanguage design and implementation, pages 290--299, New York, NY, USA, 2007.ACM Press.

21.C. A. R. Hoare.An axiomatic basis for computer programming.Commun. ACM, 12(10):576--580, 1969.

22.David Hovemeyer, Jaime Spacco, and William Pugh.Evaluating and tuning a static analysis to find null pointer bugs.In PASTE '05: Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop onProgram analysis for software tools and engineering, pages 13--19, New York, NY,USA, 2005. ACM Press.

23.John Kodumal and Alex Aiken.The set constraint/cfl reachability connection in practice.In PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on Programming language design and implementation, pages 207--218, New York, NY, USA, 2004.ACM Press.

24.James C. King and Robert W. Floyd.An interpretation oriented theorem prover over integers.In STOC '70: Proceedings of the second annual ACM symposium on Theory of computing, pages 169--179, New York, NY, USA, 1970. ACM Press.

25.Gary A. Kildall.A unified approach to global program optimization.In POPL '73: Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium onPrinciples of

programming languages, pages 194--206, New York, NY, USA, 1973.ACM Press.

26.James C. King.Symbolic execution and program testing.Commun. ACM, 19(7):385--394, 1976.

27.Xavier Leroy.Java bytecode verification: An overview.In CAV, volume 2102 of Lecture Notes in Computer Science, pages 265--285, 2001.

28.William Landi and Barbara G. Ryder.Pointer-induced aliasing: a problem taxonomy.

In POPL '91: Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 93--103, New York, NY, USA, 1991. ACM Press.

29.Monica S. Lam, John Whaley, V. Benjamin Livshits, Michael C. Martin,Dzintars Avots, Michael Carbin, and Christopher Unkel.Context-sensitive program analysis as database queries.In PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 1--12, New York, NY, USA, 2005.ACM Press.

30.Robert Muth and Saumya Debray.On the complexity of flow-sensitive dataflow analyses.In POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT symposium onPrinciples of programming languages, pages 67--80, New York, NY, USA, 2000. ACM Press.

31.Antoine Min{\'e}.A new numerical abstract domain based on difference-bound matrices.In PADO, pages 155--172, 2001.

32.Thomas Reps, Susan Horwitz, and Mooly Sagiv.Precise interprocedural dataflow analysis via graph reachability.In POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium onPrinciples of programming languages, pages 49--61, New York, NY, USA, 1995. ACMPress.

33.Manu Sridharan and Rastislav Bod\'{\i}k.Refinement-based context-sensitive points-to analysis for java.In PLDI, pages 387--400, 2006.

34.Alfred Tarski:A lattice-theoretic fixpoint theorem and its applications, Pacific J. Mathematics, 5,pages 285--309, 1955.

35. Flemming Nielson, Hanne Riis Nielson, Chris Hankin: Principles of ProgramAnalysis, Springer, 1999.

8.2 Seminar / laboratory	Teaching methods	Remarks
1. (2^{nd} week) First Project: Use K-framework to describe the semantics of a sequential program	Use practical tools to implement a small project	Seminar is organized as a total of 7 hours – 2 hours every second week

2. (4th week)Second Project: Use K-framework to describe the semantics of a concurrent program	Use practical tools to implement a small project	
3. (6th week) Third project:Use Hip/Sleek to verify sequential programs.	Use practical tools to implement a small project	
4. (8th week) Forth project:Use Verifast to verify concurrent programs	Use practical tools to implement a small project	
5. (10th week) Fifth project:Use Hip/Sleek to verify the memory usage	Use practical tools to implement a small project	
6. (12th week) Sixth project: Use Hip/Sleek to verify cpu and battery usage	Use practical tools to implement a small project	
7. (14th week) Evaluation of the projects		
Bibliography Students will use the following tools: K-framework, Hip/Sleek and Verifast		

9. Corroborating the content of the discipline with the expectations of the epistemic community, professional associations and representative employers within the field of the program

- The course respects the IEEE and ACM Curriculla Recommendations for Software Engineering studies;
- The content of the course is considered by the software companies as important for software development skills

10. Evaluation

Type of activity	10.1 Evaluation criteria	10.2 Evaluation methods	10.3 Share in the grade (%)
10.4 Course	- know the basic principle of the domain; - apply the course concepts - problem solving	Written exam	40.00%
10.5 Seminar/lab activities	– be able to implement course concept	-Practical examination	60.00%
10.6 Minimum performance standards			
➢ At least grade 5 (from a scale of 1 to 10) at both written exam and laboratory work.			

Date Signature of course coordinator Signature of seminar coordinator

..... Assoc. Prof. PhD. Ing. Florin CRACIUN Assoc. Prof. PhD. Ing. Florin CRACIUN

Date of approval Signature of the head of department

....