MSE | MASTER OF SCIENCE IN ENGINEERING

Module Description

Advanced Programming Paradigms

General Information

Number of ECTS Credits

3

Module code

TSM_AdvPrPa

Responsible of module

Edgar Lederer, FHNW

Language

Explanations regarding the language definitions for each location:

- Instruction is given in the language defined below for each location/each time the module is held.
- Documentation is available in the languages defined below. Where documents are in several languages, the percentage distribution is shown (100% = all the documentation).
- The examination is available 100% in the languages shown for each location/each time it is held.

	Berne	Lausanne			Lugano	Zurich		
Instruction	☐ E 100%	☑ E 100%		☐ F %	☐ E 100%	☑ E 100%		☐ D 100%
Documentation	☐ E 100%	☑ E 100%	☐ E %	☐ F %	☐ E 100%	☑ E 100%	☐ E %	☐ D %
Examination	☐ E 100%	☑ E 100%	☐ E %	☐ F %	☐ E 100%	☑ E 100%	☐ E 100%	☐ D 100%

Module category

☐ FTP Fundamental theoretical principles

☑ TSM Technical/scientific specialization module

☐ CM Context module

Lessons

2 lecture periods and 1 tutorial period per week

Entry-level competencies

Prerequisites, previous knowledge

Good knowledge of object-oriented programming.

Brief course description of module objectives and content

A wealth of fascinating technologies exists alongside the ubiquitous object-oriented programming and the inadequate testing methods. This module introduces students to the most relevant of these emerging technologies from a general programming-language point of view.

Paradigms Besides object-orientation as today's mainstream programming paradigm, other quite different paradigms have been developed and brought to maturity during the last decades: in particular functional programming, but also logic and constraint programming. None of these paradigms is well suited to solving all the different kinds of problems but each has its own particular strengths in specific areas. Since modern software encompasses many such areas, simultaneous application of several paradigms seems appropriate, and consequently, their seamless integration into multi-paradigm languages.

Types Programming languages with a rich and consistent type system make it possible to detect certain errors at the time of compilation already. Using the type system, invariants can be declared in one's own data types, which are then checked by the compiler. Programming in and with a strong type system can be regarded as the first step in the direction of program verification. With even stronger type systems, it is possible to formulate complete program specifications. The compiler does then, however, generally require support for verification.

Correctness Choosing the right programming paradigm for a given problem simplifies its solution, but does not guarantee its *correctness* – the most important of all software qualities. Such a guarantee requires, in addition to the actual implementation (the "How?"), a specification (the "What?") and proof of correctness (the "Why?"). Continuous research right from the very outset of computing has now led to a verification technology that is entering industrial application. Since *object-oriented programming* is ubiquitous, its specification and verification is of particular importance.

This module will provide:

- an overview of programming concepts, paradigms and languages;
- a comprehensive introduction to functional programming (using Haskell or Scala);
- an introduction to multi-paradigm programming, with special emphasis on types (using Scala, which is a combination of functional and object-oriented programming);
- an introduction to the theory and practice of specification and verification of imperative programs (as a basis for the verification of object-oriented programs - example Dafny) and/or of functional programs (example, Coq).

Aims, content, methods

Learning objectives and acquired competencies

Students will acquire an understanding of the emerging paradigms, practical skills in modern functional, multiparadigm and type-full programming, and a basic understanding of the increasingly important field of software specification and verification.

Contents of module with emphasis on teaching content

Functional programming (6 weeks)

- Programming concepts, paradigms and languages
- Absence of state, referential transparency, reasoning about programs.
- Eager versus lazy evaluation.
- Types and type inference, higher-order functions, concrete data types and pattern matching
- Functors, applicative functors, monads
- An application: interpreter for a small imperative programming language (IML)

Multi-paradigm and strong typed programming (4 weeks)

- Trait types (and Mixin composition as a variant on classical inheritance).
- Generic types (co- and contravariance for type parameters).
- Type classes and implicit parameters.
- Type-secure DSLs (*Domain Specific Languages*).

Program verification (4 weeks)

- Reliability via testing and verification.
- Hoare logic and weakest preconditions.
- Architecture of verification tools.
- An application: verification condition generator.
- A current verification tool: Dafny and/or Coq

Teaching and learning methods

- Ex-cathedra teaching.
- Programming and verification exercises.

Literature

- Graham Hutton, Programming in Haskell, Second Edition, Cambridge, 2016.
- Miran Lipovaca, Learn You a Haskell for Great Good!, No Starch Press, 2011.
- Martin Odersky, Lex Spoon and Bill Venners, Programming in Scala, Artima, 2008.
- David Gries, The Science of Programming, Springer, 1981 (a classical text).
- José Bacelar Almeida et al., Rigorous Software Development, Springer, 2011.
- Federico Biancuzzi und Shane Warden, Masterminds of Programming: Conversations with the Creators of Major Programming Languages, O'Reilly, 2009 (for recreation).

Assessment

Certification requirements for final examinations (conditions for attestation)

None

Basic principle for exams:
All the standard final exams for modules are written exams.
The repetition exams can be either written or oral.

Standard final exam for a module and written repetition exam

Kind of Exam	written
Duration of exam	120 minutes
Permissible aids	☐ no aids
	☑ permissible aids:
	☑ Electronic aids: none
	☑ In hardcopy form: all written documents
	☐ ________________________________

Special case: Repetition exam as an oral exam

If an oral exam is set (only possible for ≤ 4 students), the following applies:

Kind of Exam	oral
Duration of exam	30 minutes
Permissible aids	no aids