Module Description

Advanced Programming Paradigms

General Information

Number of ECTS Credits

3

Abbreviation

TSM_AdvPrPa

Version

19.02.2015

Responsible of module

Dr. Edgar Lederer, FHNW

Language

	Lausanne	Bern	Zürich
Instruction	☒ E ☒ F	☐ D ☐ E ☐ F	☒ D ☐ E
Documentation	☒ E ☒ F	☐ D ☐ E ☐ F	☒ D ☒ E
Examination	☐ E ☒ F	☐ D ☐ E ☐ F	☒ D ☐ E

Module category

☐ Fundamental theoretical principles
☒ Technical/scientific specialization module
☐ Context module

Lessons

☒ 2 lecture periods and 1 tutorial period per week
☐ 2 lecture periods per week

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 (including object-orientation) 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 *multiparadigm* 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.

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 a 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 into functional programming (using Haskell or Scala);
- an introduction to multiparadigm 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 (with Dafny).

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.
- An application: Interpreter for a small imperative programming language.

Multiparadigm 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

Teaching and learning methods

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

Prerequisites, previous knowledge, entrance competencies

- Good working knowledge of object-oriented programming.

Literature

- Graham Hutton, Programming in Haskell, Cambridge, 2008.
- 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 and 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)

Written module examination

Duration of exam :	120 minutes
Permissible aids:	Any written material, but no electronic devices whatsoever.