Description du module

Paradigmes de programmation avancée

Généralités

Nombres de crédits ECTS

3

Sigle du module

TSM_AdvPrPa

Version

2.12.2016

Responsable du module

Dr. Edgar Lederer, FHNW

Langue

	Lausanne	Berne	Zurich
Enseignement	☐ E ☒ F	☐ D ☐ E ☐ F	☒ D ☐ E
Documentation	☒ E ☐ F	☐ D ☐ E ☐ F	☒ D ☒ E
Questions d'examen	☒ E ☐ F	☐ D ☐ E ☐ F	☒ D ☐ E

Catégorie du module

☐ Bases théoriques élargies

☒ Approfondissement technique et scientifique

☐ Modules de savoirs contextuels

Périodes

☒ 2 périodes d'enseignement frontal et une période d'exercice par semaine

Brève description /Explication des objectifs et du contenu du module en quelques phrases

Au-delà de la programmation orientée-objets devenue une norme et du *testing* insuffisant, il existe tout un univers parallèle de technologies fascinantes. Ce module présente un panorama des diverses technologies émergentes du point de vue des langages de programmation.

Paradigmes Outre la programmation orientée objet qui est le paradigme de programmation le plus courant actuellement, d'autres *paradigmes* assez différents ont été amenés à maturité au cours des dernières décennies : en particulier la programmation *fonctionnelle,* la programmation logique ou encore la programmation par contraintes. Aucun de ces paradigmes (pas même la programmation orientée-objet) n'est adapté pour résoudre de manière optimale tous les types de problèmes, bien que chacun ait son domaine d'action où il peut faire valoir ses atouts. Toutefois, étant donné que les logiciels modernes englobent souvent bon nombre de ces domaines, l'application simultanée de plusieurs paradigmes semble appropriée. En conséquence, il existe depuis peu des langages de programmation multi-paradigmes.

Typage Les langages de programmation disposant d'un système de typage riche et consistant permettent d'identifier certaines erreurs dès l'instant de la traduction. Le système de typage permet de déclarer des invariances dans des types de données, qui seront ensuite contrôlées par le compilateur. La programmation dans et à l'aide d'un système de typage fort peut être considéré comme un premier pas vers la vérification d'un programme.

Exactitude Le choix du bon paradigme de programmation permet de simplifier la solution de tout problème donné mais ne garantit pas son exactitude, la première de toutes les qualités d'un logiciel. Une telle garantie requiert, en plus de l'implémentation (le «comment»), une spécification (le «quoi») ainsi qu'une preuve d'exactitude (le «pourquoi»). Les recherches menées depuis les origines de la science informatique ont abouti à une technologie de la vérification qui entre désormais dans sa phase d'application industrielle. Etant donné que la programmation orientée objet est omniprésente, sa spécification et sa vérification revêtent une importance particulière.

Le module donnera:

- un aperçu des concepts et des paradigmes sous-tendant les langages de programmation;
- une introduction à la programmation fonctionnelle (à l'aide de Haskell ou Scala);
- une introduction à la programmation multiparadigmes, avec un accent particulier donné aux types (utilisant Scala, un langage alliant programmation fonctionnelle et programmation orientée objet);
- une introduction à la théorie et à la pratique de la spécification et la vérification des programmes impératifs comme base de la vérification de programmes orientés objet (à l'aide de Dafny).

Objectifs, contenu et méthodes

Objectifs d'apprentissage et compétences visées

Les étudiants acquerront une vision des paradigmes émergents, des connaissances pratiques de la programmation fonctionnelle, type-full et multiparadigme, ainsi que les bases de la spécification et de la vérification, domaine en pleine expansion.

Contenu du module avec pondération des contenus d'enseignement

Programmation fonctionnelle (6 semaines)

- Concepts, paradigmes et langages de programmation
- Absence d'état, transparence référentielle, raisonnement à propos des programmes.
- Evaluation avancée ou retardée (*eager vs lazy*).
- Types et interférence du type.
- Fonctions de rang supérieur.
- Types de données concrètes et *pattern matching*.
- Une application : un interpréteur pour un petit langage de programmation impératif.

Programmation multiparadigmes et fortement typée (4 semaines)

- Les traits comme types (et composition Mixin comme variante à l'héritage traditionnel).
- Types génériques (covariance et contrevariance comme paramètres de typage).
- Classes de typage et paramètres implicites.
- Languages dédiés, DSL, (*Domain Specific Languages*) de typage sûr.

Vérification de programme (4 semaines)

- Fiabilité par les tests et la vérification.
- Logique de Hoare et calcul de plus faibles pré-conditions.
- Architecture des outils de vérification.
- Une application : générateur d'obligations de preuve VCG.
- Un outil de vérification actuel : Dafny.

Méthodes d'enseignement et d'apprentissage

- Cours ex-cathedra.
- Exercices de programmation et de vérification.

Connaissances et compétences prérequises

- Bonnes connaissances opérationnelles de la programmation orientée objet.

Bibliographie

- 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 (à titre de divertissement).

Mode d'évaluation

Conditions d'admission aux examens de fin de module (tests exigés)

Examen écrit de fin de module

Durée de l'examen:	120 minutes
Moyens autorisés:	Tout document écrit, mais aucun appareil électronique d'aucune sorte