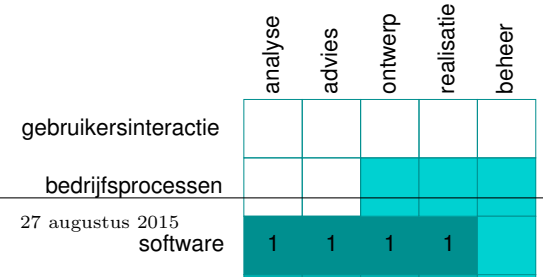
HOGESCHOOL ROTTERDAM / CMI

Security and Advanced Development Minor

TINFUN01

Aantal studiepunten: ???? ects
Modulebeheerder: Arne Padmos, Giuseppe Maggiore

Modulebeschrijving

Module name:	Security and Advanced Development Minor
Module code:	TINFUN01
Studie points and hours of effort:	This module gives ???? ectsstudie points, in correspondance with ??? hours: • ??? minutes frontal lecture • ??? minutes practicum • ??? minutes self-study • ??? minutes project work
Examination:	Projects and tests (open questions and multiple choice)
Course structure:	Lectures, self-study, and projects
Prerequisite knowledge:	All first and second year programming courses, and all courses in functional programming.
Learning tools:	For the security part: • Book: Everyday Cryptography ISBN-13: 978-0199695591 • Book: Cryptography Engineering ISBN-13: 978-0470474242 • Book: Threat Modeling Book • PDF: Security Engineering • Usable Security: History, Themes, and Challenges • Book: The art of software security assessment ISBN-13: 978-0321444424 • Book: Phishing Dark Waters: The Offensive and Defensive Sides of Malicious Emails ISBN: 978-1-118-95847-6 For the formal methods part: • Book: Friendly F# (Fun with game programming Book 1), authors: Giuseppe Maggiore, Giulia Costantini • Reader: Friendly F# (Fun with logic programming), authors: Giuseppe Maggiore, Giulia Costantini, Francesco Di Giacomo, Gerard van Kruining
Connected to competences:	

Content: For the formal methods part: • Functional programming, with special focus on F# • Meta-programming within F# • Meta-compilers, with special focus on the school's own implementation	For the security part: • ...
Modulebeheerders:	Arne Padmos, Giuseppe Maggiore
Date:	27 augustus 2015

1 Algemene omschrijving

1.1 Inleiding

Functioneel programmeren is een in opkomst zijnde manier van programmeren die nogal wat verschillen vertoont met de klassiek imperatieve manier van programmeren zoals dit bekend is in programmeertalen als C, C++, Java e.v.a.

Functionele talen zijn gebaseerd op de zogeheten λ *calculus*, waardoor deze talen fundamenteel anders werken dan de meeste programmeurs gewend zijn. Daar puur functionele talen geen side effects kennen en bovendien referential transparency gebruiken i.p.v. destructive update betekent dat programma's geschreven met functionele talen hebben hogere textitencapsulation van functies en data structuren, en vervolgens minder bugs en algemene fouten. Met het steeds populairder worden van deze platformen wordt ook het functioneel programmeren populairder.

Verder is te zien dat met het steeds verder toenemen van het abstractieniveau, waarop computerprogramma' s functioneren de behoefte aan talen, waarin deze abstracties compact kunnen worden uitgedrukt, toeneemt. Functionele talen lijken hier tot nu toe de betere papieren voor te hebben.

1.2 Relatie met andere onderwijseenheden

Op deze module wordt voortgebouwd in de module TINFUN02.

1.3 Leermiddelen

Verplicht:

- Presentaties die gebruikt worden in de hoorcolleges (pdf): te vinden op N@tschool
- Opdrachten, waaraan gewerkt wordt tijdens het practicum (pdf): te vinden op N@tschool

Facultatief:

- Boek: Types and Programming Languages, auteur: Benjamin Pierce
- Boek: Friendly F# (Fun with game programming Book 1), auteurs: Giuseppe Maggiore, Giulia Costantini
- Text editors: Emacs, Notepad++, Visual Studio, Xamarin Studio, etc.

2 Programma

De volgende lijst zijn de onderwerpen van de cursus:

1. Inleiding, interpreter, eerste programma;
2. Let, let-rec, basale flow-controle;
3. Primitive types, tuples, type sommen, pattern-matching;
4. Recursieve data-structuren, lijsten;
5. Records;
6. Hogere-orde-functies (HOF's);
7. Functie samenstelling: records-van-lambda's, continuation-passing-style (CPS);

3 Assessment

3.1 Procedure

3.2 Deliverables and deadlines

The assessment of the course is divided into two main parts. The first part is coupled with the background study of students, and is based on various questions and exercises. As the students are sufficiently prepared, then the second part comes into action and the students are asked to realize a series of projects.

Part I - background preparation The background preparation will make sure that all students have the required basic knowledge to tackle the projects later on. Each week, for the first month, students will prepare on security concepts, functional programming in F#, and the provided meta-compiler:

- **Week 1** (Friday 15:00 - 16:40) - score is 25%
- **Week 2** (Monday 08:30 - 10:10) - retake of week 1
- **Week 2** (Friday 15:00 - 16:40) - score is 25%
- **Week 3** (Monday 08:30 - 10:10) - retake of week 2
- **Week 3** (Friday 15:00 - 16:40) - score is 25%
- **Week 4** (Monday 08:30 - 10:10) - retake of week 3
- **Week 4** (Friday 15:00 - 16:40) - score is 25%
- **Week 5** (Monday 08:30 - 10:10) - retake of week 4

Part II - background preparation After students have completed their background preparation, we begin with the projects. To pass the background preparation, *the total score of Part I must be* ≥ 4. The projects are either based on security concepts, the application of formal methods to an actual compiler, or a mixture of the two. For a list of possible project thema's, see the appendix.

- **Week 5** (Friday - 16:00) - literature review and *plan van aanpak*
- **Week 6** (Monday - 08:30) - reparation week 5
- **Week 6** (Friday - 16:00) - weekly recap
- **Week 7** (Monday - 08:30) - reparation week 6
- **Week 7** (Friday - 16:00) - weekly recap
- **Week 8** (Monday - 08:30) - reparation week 7
- **Week 8** (Friday - 16:00) - weekly recap
- **Week 9** (Monday - 08:30) - reparation week 8
- **Week 9** (Friday - 16:00) - final project report or weekly recap for longer running projects
- **Week 10** (Monday - 08:30) - reparation week 9
- **Week 10** (Friday - 08:30) - individual presentation of personal project diary through GitHub check-in comments

Weeks 10 to 15 and 15 to 20 closely follow the structure of weeks 5 to 10.

3.3 Grading

Grading of Part I is based on correctness of the questions. The questions are either multiple choice, or are technical exercises with an either fully correct or fully incorrect answer. Each question adds one point to the total, so that the final grade each week is:

$$\frac{P}{N} \times 10$$

where P is the number of correct answers and N is the total number of answers.
Grading of Part II is based on adherence to the students plan van aanpak, and evaluation of the expected satisfaction of the client. **Handing-in** is completely based on GitHub, and the scale of grading is as follows:

Grade range	Process	Results
0 - 2	The students cannot document any significant work done on the project. There are few to no check-ins in the shared repository, and the repository itself is substantially empty.	There are no usable results to speak of.
2 - 4	The students have done very little work on the project. There are few check-ins in the shared repository, and the repository itself is poorly organized and has little useful content.	A few barely usable results are visible.
4 - 5,4	The students have done little work on the project. There are few check-ins in the shared repository, and the repository itself has some glimpses of organization and some useful content.	Limited and barely usable results are visible.
5,5 - 7	The students have done some work on the project. There are daily or weekly check-ins in the shared repository, and the repository itself is organized and contains useful content.	Primitive usable results are visible and clearly identified in a release accessible through the wiki pages of the repository.
7 - 10	The students have done significant amounts of work on the project. There are daily or weekly check-ins in the shared repository, and the repository itself is organized, efficient, and contains only useful content.	Concretely and highly usable results are visible and clearly identified in a release accessible through the wiki pages of the repository.

Bijlage 1: Toetsmatrijs

	Leerdoelen	Dublin descriptoren
1	editor, compiler, source, binary, statements, declaratieve/imperatieve talen	1,2,3,4
2	waarden: integer, float, double, boolean, char, String	1,2,3,4
3	selectie: if, if-else, if-else-if, match-with	1,2,3,4
4	recursie	1,2,3,4
5	lijsten en recursieve data structuren	1,2,3,4
6	HOF's en hun handeling	1,2
7	lambda calculus	1

Dublin-descriptoren:

1. Kennis en inzicht
2. Toepassen kennis en inzicht
3. Oordeelsvorming
4. Communicatie

Formal methods project - guidelines

An acceptable project for the formal methods aspect of the SAD minor covers the design of a programming language or comparable formalism with clearly specified useful properties.
The properties of interest for the minor include, but are not limited to:

- security aspects; for example, build a programming language which type system makes it impossible to send classified data without encryption over a network
- safety aspects; for example, build a programming language which type system makes it impossible to duplicate a unique value
- performance aspects; for example, build a programming language which uses code-generation to speed up some decisions by making them compile-time instead of run-time.

For a longer list of proposals, see `https://github.com/vs-team/metacompiler/issues/`. You may add your own proposal.

Structure of a programming language

An important word of warning concerns the use of tools and languages. It is, in theory, possible to build your own system from scratch, using a programming language made for other tasks (such as Java or C/C++). It is the teacher's estimation that doing so will almost always result in your failure, which for a whole minor has dire consequences. The reason why this will happen is simple. A project such as the ones described above has (very roughly) the following structure:

TESTING	SAMPLES*
BACKEND	• CODE-GEN • OPTIMIZER* • ANALYZER*
FRONTEND	• PARSER • LEXER

The only parts relevant and graded during the minor are those marked with an asterisk, that is the *analyzer* and the *optimizer* of the language. These are the parts where "real intelligence" can be put into a programming language, through knowledge of formal systems. Everything else is irrelevant for formal systems, and therefore not graded. The reason is simple: grading all of it would quickly put us out of scope and require way too much work on your behalf. Building a reliable front-end is a lot of work. In addition, building a well-working, reliable code generator that outputs useable code is a daunting task. Finally, whatever you choose to do, you will need to provide a series of basic samples that show that all aspects of your language work correctly.

Allowed tools and languages

To shield you all from this work, which would be significantly more than a single minor, we provide you with a ready-made meta-compiler that allows the definition of programming languages by focusing directly on the analyzer and the optimizer. The meta-compiler comes equipped with an extensive suite of tests that will help you understand if you broke something, and also how the program works.
You are very strongly encouraged to make use of the meta-compiler because you will get more done, and you will also get more help as the teachers are very expert in it. The meta-compiler is accessible as a GitHub project at `https://github.com/vs-team/metacompiler`.
Should you be interested in actually doing something within the meta-compiler itself, then it is also possible. The meta-compiler is written in F#, and therefore you may add features to it in the same language.

You may also suggest your own project (which should still be inspired from the existing open issues). In this case, use of F# is a still a strict requirement, and previous acceptance from the teacher is required.

To sum it up

If you are looking for a relatively easy task, then learn to use the meta-compiler and build your project in it. If you are looking for a challenging task, then learn to use both the meta-compiler and F#, and build a feature in the meta-compiler itself. If you are looking for a learning experience through very likely failure, then build the whole structure of your system in F# from scratch.