18th of October
*Subject to change

BSc in Computer Science - 180 ECTS

Study plan and Module handbook

Table of Contents

18th of October
*Subject to change

Study plan of mandatory courses

1. semester - fall

Course ID	Course name	ECTS	Length	Workload
T-111-PROG	Programming	6	12 weeks	2+1
T-107-TOLH	Computer Architecture	6	12 weeks	2+1
T-117-STR1	Discrete Mathematics I	6	12 weeks	2+1
T-216-GHOH	Software requirements and Design	6	12 weeks	2+1
T-113-VLN1	Semester Project 1	6	3 weeks	L+E

2. semester - spring

Course ID	Course name	ECTS	Length	Workload
T-201-GSKI	Data Structures	6	12 weeks	2+1
T-419-STR2	Discrete Mathematics II	6	12 weeks	2+1
T-213-VEFF	Web-Programming	6	12 weeks	2+1
T-202-GAG1	Databases	6	12 weeks	2+1
T-220-VLN2	Semester Project 2	6	3 weeks	L+E

3. semester - fall

Course ID	Course name	ECTS	Length	Workload
T-317-CAST	Calculus and Statistics	6	12 weeks	2+1
T-301-REIR	Algorithms	6	12 weeks	2+1
T-303-HUGB	Software Engineering	6	12 weeks	2+1
-	Elective course	6	12 weeks	2+1
-	Elective course	6	3 weeks	L+E

4. semester - spring

Course ID	Course name	ECTS	Length	Workload
T-501-FMAL	Programming Languages	6	12 weeks	2+1
T-215-STY1	Operating Systems	6	12 weeks	2+1
-	Elective course	6	12 weeks	2+1
-	Elective course	6	12 weeks	2+1
X-204-STOF	Entrepreneurship and Starting New Ventures	6	3 weeks	L+E

5. semester - fall

Course ID	Course name	ECTS	Length	Workload
T-409-TSAM	Computer Networks	6	12 weeks	2+1
-	Elective course	6	12 weeks	2+1
-	Elective course	6	12 weeks	2+1
-	Elective course	6	12 weeks	2+1
-	Elective course	6	3 weeks	L+E

6. semester - spring

Course ID	Course name	ECTS	Length	Workload
-	Elective course	6	12 weeks	2+1
-	Elective course	6	12 weeks	2+1
-	Elective course	6	12 weeks	2+1
T-404-LOKA	Final Project	12	15 weeks	L+E

18th of October
*Subject to change

Study plan of elective courses

BSc in Computer Science	Credits	Length (weeks)		Term		Workload		
Elective Courses	**(ECTS)**	**12 w**	**3 w**	**F**	**S**	**L**	**E**	**L+E**
T-218-ALCO Algebra and Combinatorics	6	x			x	2	1	
T-219-REMO Real-time Models	6		x		x			x
T-622-ARTI Artificial Intelligence	6	x			x	2	1	
T-315-IUPP Introduction to experience design	6		x	x				x
T-316-UPPL The Information and Technology Society	6	x		x		2	1	
E-402-STFO Mathematical Programming	6		x	x				x
T-403-FORC Programming in C++	6	x			x	2	1	
I-406-IERP Introduction to ERP Systems (ERP)	6	x				2	1	
T-414-AFLV Effective programming and probem solving	6		x		x			x
T-417-TOOR Computer Security	6		x	x				x
T-419-CADP Concurrent and distributed programming	6	x				2	1	
T-427-WEPO Web-Programming II	6	x				2	1	
T-430-TOVH Developing Open-Sourced Web Solutions/Software	6	x		x		2	1	
T-431-HANE Practical Networks	6	x			x	2	1	
T-445-GRTH Graph Theory	6	x			x	2	1	
T-488-MAPP Mobile App Development	6		x	x				x
T-498-GAGR Data Analysis	6	x			x	2	1	
T-504-ITML Introduction to Machine Learning	6	x		x		2	1	
T-505-ROKF Logic in Computer Science	6	x			x	2	1	
T-511-TGRA Computer Graphic	6	x		x		2	1	
T-513-CRNU Cryptography and Number Theory	6	x		x		2	1	

T-514-VEFT Web Services	6	x		x		2	1	
T-515-NOTH User Centred Software Development	6	x			x	2	1	
T-519-STOR Theory of Computation	6	x		x		2	1	
T-535-CPSY Cyber-Physical Systems	6	x		x		2	1	
T-542-HGOP Introduction to Quality Management and Testing	6		x	x				x
T-603-THYD Compilers	6	x		x		2	1	
T-604-HGRE Design and analysis of algorithms	6	x			x	2	1	
T-622-UROP Undergraduate Research Opportunity	6	x		x		2	1	
T-624-CGDD Computer Game Design & Development	6		x	x				x
T-634-AGDD Advanced Game Design & Development	6	x			x	2	1	
T-631-SOE2 Software Engineering II - Testing	6	x			x	2	1	
T-636-SMAT Human Computer Interaction	6	x			x	2	1	
T-637-GEDE Game Engine Architecture	6	x			x	2	1	
I-707-VGBI Business Intelligence	6	x			x	2	1	

- 1 ECTS = 25-30 hours
- Each term is divided in to two periods, 12-week period and 3-week period
- F = Fall term
- S = Spring term
- L = Lectures, 1= 2 x 45 min
- E = Exercises, 1= 2 x 45 min
- L+E = Lectures and exercises combined, taught in 3-week period, approx. 8 hours a day, 5 days a week

Descriptions of mandatory courses

T-107-TOLH Computer Architecture

Credits: 6 ECTS
Year: 1st year BSc in Computer Science and Bsc in Discrete Mathematics, 2nd year in BSc in Software Engineering
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory
Prerequisites: None
Structure: 12. week course, on-site and HMV, 2 lectures and open problem-solving classes each week.
Lecturer: Gylfi Þór Guðmundsson

Description

In this course, students will learn the fundamental operations of a computer, with a special emphasis on issues related to programmers. They will learn how and why the CPU works, how it uses binary math for calculations, and how numbers and data is presented in binary. Students become familiar with reading x86_64 assembly code. They learn how programs are loaded into memory (register, cache, RAM etc.). Students learn how to use common commands in the command line.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be able to describe the architecture of a computer system in terms of the major building blocks, for instance, CPU, I/O, main memory and operating system.
- Be able to explain what programs are and how they run on the hardware
- Be able to explain programs written in an instruction set of a CPU (x86_64).
- Be able to describe in detail how data, including numbers, are represented, stored, and retrieved in computers.
- Have gained basic proficiency with the UNIX / Linux operating system

Skills

- Be able to write and explain basic x86_64 assembly code.
- Be able to disassemble, trace and perform rudimentary debugging of programs written in Intel x86_64 assembly.
- Be able to write and debug simple programs in the C programming language.
- Be able to use command line tools for basic tasks in Linux or other Unix-based operating systems.
- Be able to implement basic mathematical functions using only binary operators.

Course assessment

Assignments – 30 %
Bomb lab – 15 %
Data lab – 15 %
Final exam - 40 %

Reading Material

Computer Systems (2016): A Programmer´s Perspective: Randal E. Bryant and David R. O´Hallaron, Pearson, 3. ed. global ed.

18th of October
*Subject to change

T-111-PROG Programming

Credits: 6 ECTS
Year: 1st year
Semester: Fall and Spring semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory
Prerequisites: None
Structure: 12. week course, on-site and HMV, Flipped classroom, 2x4 class sessions a week
Lecturer: Hrafn Loftsson

Description

This is an introductory course in programming using Python. Fundamental programming constructs are covered, e.g. variables, types, control structures, and functions, as well as built-in data structures like strings, lists, and dictionaries. The concept of a class is introduced and how it supports encapsulation and information hiding in the context of object-oriented programming. Students learn to use both an Integrated Development Environment (IDE) and command prompt mechanisms for development and execution of programs.
The last three years, the course has been taught using the Flipped Classroom teaching method.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be able to analyse and explain the functionality of simple programs that use the following fundamental programming constructs: variables, types, expressions and assignments, simple I/O, conditional and iterative statements, collections and functions.
- Be able to analyse and explain the functionality of simple programs that use classes.
- Be able to explain the concepts of encapsulation, information hiding and abstract data type and how classes support these concepts.
- Be able to understand the difference between a declaration and an implementation.
- Be able to discuss the importance of an algorithm in problem-solving and how a problem can be solved with different algorithms. mentation.

Skills

- Be able to use a command shell and an integrated development environment (IDE) for developing and running a program.
- Be able to design, implement, test, debug and change a program that uses each of the following fundamental programming constructs: variables, types, expressions and assignments, simple I/O, conditional and iterative statements, collections and functions.
- Be able to choose appropriate conditional and iterative constructs for a given programming task.
- Be able to apply top-down design to break a program into smaller pieces.
- Be able to design, implement, test and debug a program that uses classes.
- Be able to design an algorithm to solve a simple problem.

Competence

- Be able to design and implement a program for a problem that is described in a general manner.

Course assessment

In class quizzes : 10%
In class programming assignments: 10%
Weakly home assignments (two in group): 20% (7 best out of 10)
Mid term exams: 20% (Two exams, no retake)
Final exam: 40-60%.

18th of October
*Subject to change

Reading Material

The Practice of Computing Using Python. Third Edition (Global Edition). William Punch & Richard Enbody. Pearson Education, 2017.

18th of October
*Subject to change

T-215-STY1 Operating Systems

Credits: 6 ECTS
Year: 2nd year
Semester: Spring semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory in all BSc programmes
Prerequisites: T-201-GSKI, Data Structures and T-107-TOLH, Computer Architecture
Structure: 12. week course
Lucture: Hans Reiser

Description

The course will cover many of the fundamentals of operating systems:
x86_64 assembly, virtual memory, processes, threads, process communications,deadlocks, scheduling, memory management, I/O, filesystems, access control and security. The crux of the course will be projects and hands-on assignments.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be able to explain the objectives and functions of modern operating systems.
- Be able to explain dynamic memory allocation in modern operating systems.
- Be able to describe the need for concurrency within the framework of an operating system.
- Be able to explain the memory hierarchy and cost-performance trade offs.
- Be able to describe the difference between processes and threads.
- Be able to discuss the need forhardware cache, as well as common algorithms and optimizations used to impleme nt it..
- Be able to explain signal handling within UNIX -based operating systems.

Skills

- Be able to explain conditions that lead to deadlock.
- Be able to compare and contrast the common algorithms used for both preemptive
- and non- preemptive scheduling oftasks in operating systems, such as priority, performance comparison,
- and fair-share schemes.
- Be able to explain the concept of virtual memory and how it is realized in hardware and software.
- Be able to summarize the principles of virtual memory as applied to caching, paging, and segmentation.
- Be able to compare and contrast paging and segmentation techniques.
- Be able to disassemble, trace and perform rudimentary debugging of programs written in Intel x86_64 assembly.
- Be able to boot an operating system using a simulator.
- Be able to write a simple kernel module for the Linux kernel.
- Be able to write a buffer-overflow exploit.
- Be able to write working C code that interacts with standard C libraries and the operating system kernel directly.
- Be able to write a multi-threaded multi-tenant service using semaphores and mutexes.
- Be able to write a primitive command shell for UNIX-based operating systems.

Competence

- Be able to disassemble, trace and perform rudimentary debugging of programs written in Intel x86_64 assembly.
- Be able to boot an operating system using a simulator.
- Be able to write a simple kernel module for the Linux kernel.
- Be able to write a buffer-overflow exploit.
- Be able to write working C code that interacts with standard C libraries and the operating system kernel directly.
- Be able to write a multi-threaded multi-tenant service using semaphores and mutexes.
- Be able to write a primitive command shell for UNIX-based operating systems.

Course assessment

Homework – 15%
Participation Activities – 15%
Programming Projects – 20%

18th of October
*Subject to change

Exams – 50% (25% each)

Reading Material

Operating Systems published by zyBooks/Wiley

T-216-Software requirements and Design

Credits: 6 ECTS
Year: 1st year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory in Computer Science and Computer Science – business minor.
Prerequisites: No prerequisites
Structure: 12-week course
Lecturer: Marta Kristín Lárusdóttir og Skúli Arnlaugsson

Description

Methods for the presentation and organization of requirements for software and its design are taught in the course. The user's needs are analyzed, the structure and design of systems are defined and collaboration with the user is practiced. Students practice the use of design methods and models in software development and user interface design. The course coveres different test methods and introduces definition and chracteristics of object-oriented modelling and design. The main emphasis of the course is on the practice of methods and models for claiming, analysis, design and testing in the early stages of software development.

Learning Objectives

On completion of the course, students should be able to:

Knowledge

- Be familiar with methods for analysing software requirements.
- Be familiar with methods for designing software.
- Know basic principles in the design of user interfaces, to be called standards and guidelines for the design of user interfaces.
- Know the main definitions and characteristics of object oriented modelling and design.
- Be familiar with different methods of information gathering.
- Know in what way the design of a computer system or application might succeed or fail because of the diversity of human beings.
- Be able to identify the main types of software testing and when these are used.
- Be able to describe the main concepts for user centred software development such as usability and user experience.

Skills

- Be able to state requirements, (both functional and non-functional) for a medium sized computer system.
- Be able to make paper prototypes and intermedium prototypes of a software system.
- Have developed skills of testing in the analysis and design phase.
- Be able to evaluate the individual parts of design software.
- Be able to model the system design using diagrams like state, sequence and class diagrams.
- Be able to write reports that are understandable for recipients.

Competence

- Be able to state the requirements for a software system in a comprehensive manner.
- Be able to design user interface software systems according to the needs of users.
- Be able to evaluate their designs and improve the design with iterations.
- Be able to set out an analysis and design of a comprehensive manner in the form of reports, prototypes and models.

Method of examination

40% of the final grade - Group assignments
10% of the final grade – Individual assignments
50% of the final grade – Written exam

18th of October
*Subject to change

To complete the course students have to: Get 4,75 or above in the written exam

Reading material

Interaction Design: Beyond Human-Computer Interaction (5th edition) by Sharp, H. ,Preece, J., Rogers, Y., Wiley, 2019
UML Distilled (3rd Edition) by Fowler, M., Addison-Wesley, 2004

18th of October
*Subject to change

T-220-VLN2 Semester project 2

Credits: 6 ECTS
Year: 1st year
Semester: Spring semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory in BSc in Computer Science, Bsc in Computer Science-Business minor
Prerequisites: T-216 Software requirements and Design, T-213-VEFF Web programming, T-202-GAG1 Databases
Structure: 3-week course, on-site
Lecturer: Arnar Leifsson

Description

In this course we will combine knowledge and skills from previous courses and create a fullfledged application you can be proud to show. In the course you will create an application from idea phase to a fully functioning application. You will embark on technologies such as Django, PostgreSQL and more.

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Student knows the requirement- and design phase of creating a complex web application.
- Student knows how to use Django to create complex web applications.
- Student knows what the MTV/MVC pattern is.
- Student knows how to communicate with a database using the Model API.
- Student knows how to migrate database changes using code-first in Django.
- Student knows how to communicate with Django views using JavaScript.
- Student knows how to use Git for better team collaboration.

Skills

- Student can work in group work when developing a complex web application.
- Student can setup a Django project.
- Student can create a complex web application using Django and Python.
- Student can pitch their assignment in an informative and concise manner. combinations.

Course Assessment

Programming Assignment: 55%
Reports: 30%
System Walkthrough: 10%
Peer Review: 5%

Reading material

Slides from lecturer.

18th of October
*Subject to change

T-301-REIR Algorithms

Credits: 6 ECTS
Year: .2nd year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory
Prerequisites: T-201-GSKI Data structure
Structure: 12. week course, on site
Lecturer: Magnús Halldórsson

Description

This course introduces the most important types of algorithms and data structures in use today. Emphasis is placed on algorithms for sorting, searching and graphs. The focus is on developing implementations, analyzing them or evaluating empirically, and assessing how useful they can be in actual situations.

Learning Objectives

Upon completion of the course, the student should:

Knowledge

- Be able to describe the efficiency of major algorithms and data structures for searching and sorting.
- Be able to describe the problem with exponential growth of brute-force solutions and its consequences.
- Be able to give examples of the use of graphs, trees, and symbol tables.

Skills

- Be able to formulate computational problems from general textual description.
- Be able to apply different search methods on trees and graphs.
- Be able to trace the execution of operations on classic data structures: heaps, binary search trees, red-black trees, union-find structures, and tries.
- Be able to implement and apply fundamental algorithms for graphs, such as depth-first and breadth-first search, transitive closure, topological sort, and algorithms for shortest paths and minimum spanning trees.
- Be able to assess the impact of different implementation of abstract data types on the time complexity of algorithms.
- Be able to use “big-O”, omega and theta notations to give the asymptotic upper, lower and tight limits on the time and space complexity of algorithms.
- Be able to apply the scientific method to infer the performance behavior of algorithms.
- Be able to implement generic data structures and apply them to different data.

Competence

- Be able to assess, algorithms, choose between possible solutions, justify the choice of method and implement in programs.
- Be able to solve algorithmic problems in a program by combining appropriate algorithms and data structures.
- Be able to write a comprehensive description of experimentation, outcomes, and their implications

Course Assessment

Small assignments: (7out of 8 best) 21%
Bigger assignments: 3 x 8% = 24%
Final exam 50%
Participation 5%

Reading material

Sedgewick and Wayne: Algorithms, 4th ed.

T-303-HUGB Software Engineering

Credits: 6 ECTS
Year: .2nd year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory in BSc in Computer Science, BSc in Discrete Mathematics and Computer Science, BSc in Computer Science-Business minor
Prerequisites T-111-PROG Programming and T-216-GHOH Software requirements ans design
Structure: 12. week course, On-site
Lecturer: Grischa Liebel

Description

T-303-HUGB will cover the essentials of the term Software Engineering (SE): Process models, Requirements Engineering, Software Modelling, Architecture, Design and Testing. This coverage of basic SE knowledge is complemented with several recent trends in SE. Knowledge in Requirements Engineering and Software Modelling is only provided in addition to the material covered in T-216-GHOH. The course is intended as an introduction course, thus covering basics in many topics, all of which could be deepened in the form of additional courses.

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Contrast software engineering techniques required for different types of software systems.
- Discuss ethical issues arising in the context of modern software engineering projects.
- Explain what software engineering is and why it is needed.
- Illustrate the term stakeholder in relation to different types of software systems.
- Summarise different techniques for performing requirements validation.
- Discuss how system modeling can be used in different ways to address the needs of modern software systems.
- Discuss the need for systematic processes in software engineering.
- Compare plan-driven and agile processes in relation to different types of software systems.
- Explain several common agile practices.
- Discuss the issues of applying agile processes in large-scale and regulated environments.
- Explain the different stages and scopes of testing.
- Discuss different testing coverage criteria.
- Discuss how architectural decisions can affect different system qualities.
- Illustrate key architectural patterns.
- Explain key design patterns of object-oriented design.
- Contrast security and safety in the context of software systems.
- Summarise design guidelines to achieve security in software systems.
- Illustrate the key ideas of model-based engineering.
- Summarise recent trends in software engineering.

Skills

- Classify different kinds of requirements needed in software engineering.
- Apply system modeling to provide an overview of a software system.
- Demonstrate understanding of different parts of the Scrum process.
- Conduct unit and system testing in a test-first matter.
- Make use of architectural styles/patterns to create a basic system architecture.

Competence

- Formulate functional and quality requirements using different techniques.

18th of October
*Subject to change

- Adapt a process to the specific needs of a software system.
- Examine the role of human factors in the development of software systems.

Course assessment

Theoretical part 50% and practical part 50%:
Quiz: 10%
Quiz: 10%
Quiz: 10%
Final exam: 20%
Project: 50%

Reading material

Software engineering by Ian Summerville (9th edition)

18th of October
*Subject to change

T-317-CAST Calculus and Statistics

Credits: 6 ECTS
Year: 1st year
Semester: Fall semester.
Level of course: N/A
Type of course: Elective
Prerequisites: None
Structure: 12. week course
Lecturer: Henning Ulfarsson and María Óskarsdóttir

Description

The main objective of the course is to show how to design and implement software enterprise solutions. To realize this, the focus is on object oriented architecture and module design. Many known design patterns are discussed and evaluated. The focus is on layered Internet systems and web APIs. The course covers how to build flexible system that are easy to adapt, maintain, and operate. Options facing architects and how to recognize important key design goals is covered. Topics like performance and scalability of enterprise solutions are also covered. The course uses the Java programming language along with several open source libraries, APIs, and open source tools.

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Know basic properties of functions of one variable, such as polynomial, rational, logarithmic, exponential and trigonometric functions.
- Be familiar with the basic concepts of calculus, such as continuity and differentiability.
- Know the derivatives of common functions of one variable, such as polynomial, rational, logarithmic, exponential and trigonometric functions.
- Understand integration and know integrals for common functions of one variable.
- Be familiar with integration by substitution and integration by parts.
- Know what a differential equation is and have seen some examples of differential equations.
- Be familiar with discrete probability and some basic methods for its calculation, in particular permutations and combinations.

Skills

- Be able to analyze various problems and apply the methods of the calculus of one variable to solve them.
- Be able to apply hypothesis testing to analyze sets of measured data.

Course Assessment

Theme project– 50%
Reading diary – 40%
Report – 10%

Reading material

Slides from lecturer.

T-404-LOKA Final Project

Credits: 12 ECTS
Year: 3rd year
Semester: Fall and spring semester
Level of course: 2. First cycle, intermediate
Type of course: Mandatory in all programmes
Prerequisites: T-216-GHOH, Software Requirements and Design, T-220-VLN2, Semester Project 2, T-303-HUGB, Software Engineering
Structure: 15. week course
Lecturer: Hallgrímur Arnalds

Description

The Final Project consists of software development in collaboration with a customer and users outside the university. The purpose of the final project is to give students experience of working independently on specification, design and implementation of software and to use accepted methods in the development cycle. Normally 2 to 4 students work together in a project group. While working on the project, students gain practical experience of analysis, design, programming and testing. The projects are evaluated by the project supervisor and two other internal examiners. The grade is based on evaluation at various stages of development and considers all aspects of the development work. The projects conclude with a public presentation. To be able to register for a final project, students need to have finished at least 78 ECTS credits.

Learning outcomes

Upon completion of the course, the student should:

- Present the work to different audiences with or without technical backgrounds.
- Have gained experience working on a mid-sized software project with a team.
- Use a version control system in software development.
- Organize the team, define a schedule, and work according to defined schedule in making a software system.
- Have gained practical training in project management.
- Design, analyze and implement software.
- Choose and justify the choice of an approved method of software development.
- Define and carry out the user, unit and system testing.
- Analyze user needs and implement the software necessary to fulfill the user needs.
- Explain the status of the project, what the project was created to perform, what is left, and give a project status based on schedule.

Course Assessment

Final Project, grade from supervisor and examiner.

Reading material

Slides from lecturers

18th of October
*Subject to change

T-409-TSAM Computer Networks

Credits: 6 ECTS
Year: 3rd year
Semester: HaustSemester Fall semester
Level of course: 2. First cycle, intermediate
Type of course: Mandatory
Prerequisites: T-201-GSKI Data structures
Structure: 12. week course, on-site
Lecturer: Jacqueline Clare Mallet og Stephan Schiffel

Description

The course will focus on teaching methods for analysing, designing and evaluating software systems anticipating the users aspects in software development. Students will gain skills in using particular methods for analysis, design and evaluation of user interfaces. Furthermore, other methods for analysing, designing and evaluating user interfaces will be described. Research on user centred software development methods will be described, when it is best to use each method and how practitioners have ranked the methods. The integration of user centred software development methods into Scrum will be discussed, and the integration of user experience into lean software development. Furthermore, experiences from industry will be a part of the course. The methods taught in this course supplement those taught in the course Software Requirements and Design.

Learning outcomes

Knowledge

- Be familiar with several methods for analysing the user needs for software systems.
- Be familiar with evaluation with and without the participation of users.
- Be familiar with guidelines for good user interface design.
- Be familiar with the integration of user-centered design methods in the Scrum software process.

Skills

- Be able to make a vision (ie, Visioning) for a software project and explain it.
- Be able to analyse the context of use for software systems.
- Be able to perform contextual inquiries and derive the results using an affinity diagram.
- Be able to design an interface that is based on the relationship schema and test it with users.
- Be able to perform formal user evaluations.

Competence

- Assessment Be able to choose which user-centered design methods are suitable in different cases.
- Know the advantages and disadvantages of user centred design methods.

Course assessment

Weekly quizzes: 10%
Home assignments: 15% (3x5%)
Programming assignments: 25% (5%+10%+10%)
Final exam: 50%
Bonus points for questions and answers on Piazza: 5%
Bonus point for „excellent code quality“: 5%

Reading material

Computer Networks Andrew Tannenbaum Pearson (New International Edition/5th)

T-419-STR2 Discrete Mathematics II

Credits: 6 ECTS
Year: 2nd year
Semester: Spring
Level of course: 1. cycle, advanced
Type of course: Mandatory in Computer Science and Computer Science – minor in business
Prerequisites: Discrete Mathematics I
Structure: Twelve- week course
Lecturer: Antonios Achilleos

Description

This course is the follow-up of Discrete Mathematics I. It covers the basics of three topics in discrete mathematics that are of fundamental importance in the theory and practice of computer science: grammars and finite automata as models of languages and computation, respectively; linear algebra and its applications in computer science, with emphasis on solving systems of linear equations using Gaussian elimination. matrix algebra and matrix transformations, operations on vectors (scalar multiplication, dot product and cross product) and equations for lines and planes; discrete probability, with focus on assigning (conditional) probabilities, Bayes´ Theorem and expectation.

Learning Objectives

On completion of the course, students should be able to:

Knowledge

- Understand recursive definitions and their associated proof techniques.
- Understand finite automata, grammars, regular expressions and their relationships.
- Understand the concepts of cardinality, countably infinite sets and uncountable ones.
- Be aware that the halting problem is algorithmically unsolvable.
- Understand the basic notions in linear algebra related to matrices, vectors and linear transformations.
- Know how to use linear algebra in computer graphics.
- Understand probability and conditional probability.
- Understand Bayes' rule.Understand the concept of expectation.
- Be familiar with the binomial distribution.
- Be familiar with the definitions of trigonometric functions, including addition formulas.

Skills

- Be able to formulate inductive definitions of discrete structures, such as strings and trees, and construct proofs by structural induction over those structures.
- Be able to argue whether an infinite set is countable or uncountable.
- Be able to design grammars generating some simple languages and finite automata accepting them.
- Be able to write regular expressions denoting some simple regular languages.
- Be able to solve systems of linear equations using Gaussian elimination.
- Be able to use matrix algebra and geometric transformations in computer graphics.
- Be able to use operations on vectors (scalar multiplication and dot product) and to write equations for lines and planes.
- Be able to assign probabilities to events over finite probability spaces.
- Be able to calculate conditional probabilities.
- Be able to apply Bayes´ Theorem to estimate probabilities based on partial evidence.
- Be able to calculate the expected value of a discrete random variable.
- Be able to use trigonometric functions for rotations in computer graphics

Competences

- Have the knowledge to use automata, grammars and regular expressions in computer science applications, such as compiler, design, software engineering and testing.

- Have the knowledge to search for applications of linear algebra in computer science.
- Know when and how to use the tools of discrete probability, conditional probability and Bayes' rule.

Course assessment

3 projects 15%

5 smalles projects. 15%

Mid term exam 30%

Final exm 40%

Reading material

Discrete mathematics and Its Applications, (aðalbók)

Höfundur: Kenneth H. Rosen

Útgefandi: McGraw-Hill

Útgáfa: Seventh Edition (Global Edition)

ÚtgáfuYear: 2013

T-501-FMAL Programming languages

Credits: 6 ECTS
Year: 2nd year
Semester: Spring
Level of course: 1. cycle, introduction
Type of course: Mandatory
Prerequisites: T-201-GSKI, Data Structures, T-419-STR2, Discrete Mathematics II
Structure: Twelve-week course
Lecturer: Tarmo Uustalo

Description

The evolution of programming languages is an important factor in computer science. The course describes this evolution from the first programming languages to the more recent languages. Different types of programming languages are discussed and their characteristics compared. Programming languages syntax is introduced as well as Backus-Naur Form (BNF). Main characteristics of imperative languages are examined, particularly regarding to scope rules and procedure activations. The main characteristics of object-oriented languages will be covered. The constructs of functional programming languages are examined with emphasis on Lambda calculus. Logic programming is introduced. Students are introduced to the design and syntax of various languages and experiment with several programming projects using some of these languages

Learning Outcomes

Upon completion of the course, students should be able to:

Knowledge, skills and competence.

- Be able to describe formal methods used for describing programming languages.
- Know the role of the individual components of a compiler.
- Be able to describe the difference between static and dynamic scope rules.
- Be able to describe the run-time stack and the role and implementation of activation records.
- Be able to define control abstraction and data abstraction.
- Know the main characteristics of object-oriented languages.
- Know the main characteristics of functional languages.
- Know the main characteristics of logic languages.
- Know the history and trends of language developments.
- Be able to use and define a context-free grammar for a simple programming language.
- Be able to program a simple compiler.
- Be able to program in a functional programming language.
- Be able to program in a logic language.

Course assessment

Four assignments	70%
Final exam	30%

Reading Material

Peter Sestoft. *Programming Language Concepts*, 2nd ed. Springer, 2017

18th of October
*Subject to change

T-113-VLN1 Semester Project 1

Credits: 6 ECTS
Year: 1st year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory
Prerequisites: To have been enrolled in the course(s) T-111-PROG Programming and T-216-GHOH
Structure: 3. week course, on-site, the course is divided up into week long subtasks. Week 1- design and planning of the software. Week 2-implementation (coding). Week 3-finishing up code and preparation for delivery (hand-in).
In this course the students are expected to do a full 8 hours of work for each working day over the three weeks that this course spans.
Lecturer: Gylfi Þór Guðmundsson

Description

The course is based on the knowledge and experience the students have previously acquired during the programming course. Students will gain a greater understanding of the use of classes and object-oriented programming by creating layered software projects. Troubleshooting and debugging will be covered. Students are introduced to the SQL programming language and response driven programming with a graphical user interface. Students will be introduced to a version control system that will be used throughout the course.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

Be able to identify the main types of UI programs and write simple such programs.
Know the advantages of a layered architecture.
Recognize the benefits of using a version control system.
Be able to discuss copyright, intellectual property, data protection and security.

Skills

Be able to write simple algorithms.
Be able to give simple commands in the console.
Know additional skills in input data validation and application debugging.

Competence

Be able to use classes and object- oriented programming when constructing a simple software project.
Be able to set up small databases, retrieve data from them, and write data using SQL.

Course assessment

Group member evaluation (self and peer assessment): 10%
Final report/product: 90%

Reading material:

slides from lecturer

T-117-STR1 Discrete Mathematics 1

Credits: 6 ECTS
Year: 1st year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory in BSc in computer science, Bsc in Computer Science-Business minor, Diploma in Computer Science
Prerequisites: None
Structure: 12. week course,2 lectures and 2 problem solving classes each week.
Lecturer: Steinunn Gróa Sigurðardóttir

Description

The main material in this course consists of various aspects of mathematics that are basic to an understanding of the fundamentals of Computer Science. Various topics are discussed and their relevance to practical issues in Computer Science demonstrated. Topics covered include: logic and set theory, functions, relations, matrices, mathematical induction, counting techniques and graph theory.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be familiar with various topics in discrete mathematics that are important for an understanding of the fundamentals of computer science.
- Know basic concepts in propositional logic and predicate logic.
- Have been introduced to logic and formal reasoning.
- Know basic set operations.
- Know basic properties of functions, in particular logarithmic and exponential functions, the floor function and ceiling function.
- Have learnt introductory matrix algebra.
- Know basic counting techniques.
- Have learnt basic concepts of recurrence relations.
- Be familiar with basic material on relations.
- Know basic concepts in graph theory, for instance Euler and Hamilton paths, shortest path and graph coloring.

Skills

- Be able to construct truth tables, use basic logical equivalences in propositional logic and use quantifiers.
- Be able to construct direct and indirect proofs.
- Be able to construct proofs by mathematical induction and strong induction. Also, be able to construct inductive definitions.
- Be able to prove formulas in set theory using basic set identities.
- Be able to solve simple problems involving logarithmic functions, exponential functions, the floor function and the ceiling function.
- Be able to use basic matrix operations, e.g. multiplication, for matrices with numbers as well as boolean matrices.
- Be able to solve simple counting problems for finite sets, e.g. using permutations and combinations.
- Be able to construct recurrence relations.
- Be able to use recurrence relations as a model to solve various problems.
- Be able to analyze basic properties of relations, in particular equivalence relations.
- Be able to solve problems in graph theory, e.g. involving Euler and Hamilton paths and counting the number of different paths of a certain length.
- Be able to use Dijkstra´s algorithm to find the shortest path in a graph.
- Be able to find the chromatic number of various graphs.
- Be able to use graph theory to solve certain practical problems.

18th of October
*Subject to change

Competence

- Be able to use logic to analyze statements in the English language.
- Be able to apply graph theory models in various situations outside the scope of the course.
- Be able to use the material in the course to understand formal reasoning in later courses. courses.

Course assessment

Participation– 5%
Assignments – 10%
Group assignments - 15%
Mid-term exam - 20%
Final exam - 50%

Reading material

Rosen: Discrete Mathematics and Its Applications, 8. edition.

18th of October
*Subject to change

T-202-GAG1 Databases

Credits: 6 ECTS
Year: 1st year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory
Prerequisites: T-111-PROG Programming
Structure: 12-week course
Lecturer: Anna Sigríður Islind

Description

The course is a hands-on introduction to information management in general and relational database management in particular, covering the following topics: the role and function of database management systems; the relational database model, including relational concepts and relational query languages; data modeling using the ER model and its conversion into a relational database schema; all major aspects of the SQL language, covered in detail, including DDL, DML, complex queries, views, procedures, triggers and transactions; transaction and administration concepts; and, finally, a brief discussion of alternative data models and approaches, such as unstructured databases, information retrieval and "big data".

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be able to discuss structured and unstructured databases in social and organizational context, including privacy and ethical issues, access and preservation.
- Be able to describe concepts and measures related to reliability, scalability, efficiency and effectiveness.
- Be able to describe major components and functions of database management systems.
- Be able to describe and compare common data models.
- Be able to describe fundamental principles of the relational model.
- Be able to describe fundamental transaction concepts.
- Be able to describe basic database administration functions.
- Be able to discuss concepts and techniques for unstructured data and information retrieval.
- Be able to discuss major approaches to storing and processing large volumes of data.

Skills

- Be able to write SQL commands to create a complete relational database.
- Be able to write SQL commands to insert, delete, and modify data.
- Be able to write simple and complex SQL queries to retrieve data, including joins, aggregates, sub-queries, nested sub-queries, and division.
- Be able to write simple database views, stored procedures, triggers and transactions.
- Be able to write queries in relational algebra and tuple relational calculus.

Competence

- Be able to model data requirements and constraints using the ER-model.
- Be able to convert an ER-model into a corresponding relational schema.
- Be able to normalize a relational schema.
- Be able to select and create the appropriate indices for simple database queries and constraints.

Course assessment

In-class quizzes: 10%
Group assignments: 50%
Final exam – 40%

18th of October
*Subject to change

Reading Material

Ramakrishnan & Gehrke. Database Management Systems

18th of October
*Subject to change

T-213-VEFF Web programming

Credits: 6 ECTS
Year: 1st year
Semester: Spring semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory in BSc in Computer Science, Computer Science – minor in Business and BSc in Software Engineering
Prerequisites: T-111-PROG Programming
Structure: 12-week structure
Lecturer: Grischa Liebel

Description

This course covers the basics of web development and of server-side web applications. Additionally, it covers web security, and how to test and debug web applications.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Define and contrast client-side and server-side web applications.
- Summarise the content of HTTP requests and responses.
- List different HTTP verbs and explain their purpose.
- Explain the features of the different HTTP verbs.
- Define and explain key language concepts of HTML, CSS, and JavaScript.
- Define accessibility for web applications and give examples for accessible/not accessible code.
- Predict the behaviour and look of a web application based on its source code.
- Predict the behaviour of asynchronous JavaScript codel.
- Discuss web application testing and contrast different testing techniques.
- Summarise the different principles of RESTful APIs.
- Discuss the correctness of HTTP response status codes for different REST endpoints.
- List and explain the most important web security threats according to the OWASP TOP 10.

Skills

- Develop basic client-side web applications using HTML, CSS, and JavaScript.
- Make use of AJAX to enrich web applications with asynchronous behaviour.
- Debug and test basic client-side web applications.
- Analyse web application source code for errors.
- Choose the correct HTTP request method for different REST endpoints.
- Build a RESTful backend application using Node.js and Express.js.
- Analyse web application source code for errors.
- Analyse an existing RESTful API and point out shortcomings.
- Deploy a server-side JavaScript application to an online cloud provider.
- Test and debug server-side JavaScript code.
- Develop tests for common web security threats.
- Inspect web application source code for potential security threats.

Competence

- Propose improvements to web application source code.
- Improve existing web application source code.
- Assess existing code for errors and security vulnerabilities.
- Compare different testing techniques for web applications.
- Design a RESTful API according to given requirements.

- Convert a backend API so that it conforms to the REST style.
- Debate the importance of testing and debugging for web application development.

Course assessment

Assignments: 50%
Final exam: 50%

Reading Material

Semmy Purewal, Learning Web App Development: Build Quickly with Proven JavaScript Techniques, O'Reilly Media

18th of October
*Subject to change

X-204-STOF Entrepreneurship and Starting New ventures

Credits: 6 ECTS
Year: 1.st year
Semester: Spring semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory in all BSc programmes
Prerequisites: No prerequisites
Structure: 3. week course, Lectures, teamwork, 2-3 weeks at the end of term are assigned as teamwork for this course.
Lecturer: TBA

Description

The course aims at developing business ideas into business opportunities and a comprehensive business plan for the new company, which is divided into four main areas: (i) business opportunities and sources - the business idea. (ii) Preparation of a business plan - a reality test. (iii) Making a business plan. (iv) Introduction to business ideas for investors.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Students can identify basic terms such as Business Model, Business Plan, Grants, Seed funding, Business Angels, VC Investments, Entrepreneurship Center, Incubators, Accelerators and Startups
- Students understand a typical growth path for new startups.
- Students can identify the services available for new startups in Iceland in different stages up until VC investments.
- Students understand where to find information about the services available for startups on the web

Skills

- Students can pitch a short elevator pitch for an audience
- Students can pitch a business idea to an investor
- Students can draw an example of the growth path of new businesses
- Students can understand media reporting regarding Startups
- Students can talk to possible customers and acquire information regarding their needs and rather the business idea needs to pivot in order to meet those needs

Competence

- Students are able to understand the effects of political decision making on the Entrepreneurial Eco System
- Students realise what data gives information about the growth of the Entrepreneurial Eco System
- Students can interpret media news regarding Startup Businesses
- Students know what main documents are needed in founding a business in Iceland and key factors to think about before doing so.

Course assessment

Peer Assessment	20%
Sprint Daily	15%
Sprint Retrospective	5%
Cap table, Vesting and options plan	10%
Business Plan	15%
Final Presentation - Verbal	20%
Final Presentation - Pitch Deck	10%
Elevator Pitches 2	5%

Reading Material: Slides from lecturers

Description of elective Courses

T-218-ALCO Algebra and Combinatorics

Credits: 6 ECTS
Year: 2nd year
Semester: Spring term, tought on odd years
Level of course: 1. First cycle, introductory
Type of course: Mandatory in BSc in Discrete Mathematics and Computer Science
Prerequisites: Discrete Mathematics for engineering or Discrete mathematics II
Structure: 12-week course
Lecturer: Christian Bean

Description

This course covers groups, rings and related themes in abstract algebra. We will also cover basic combinatorics: Counting methods, partitions, generating functions, permutations and patterns.

Learning objectives

On completion of the course, students should be able to:

Knowledge

- Know when a set with an operation is a group and when a subset of a group is a subgroup.
- Know the most common properties of groups, such as commutative and cyclic groups, normal subgroups etc.
- Know when a set with two operations is a ring and when a subset of a ring is a subring.
- Know common sets that appear in combinatorics, such as special sets of lattice paths, permutations, strings, etc.
- Know standard methods of counting.
- Know classical permutation patterns.

Skills

- Be able to use common theorems from group theory to solve/prove statements in group theory and combinatoric s.
- Be able to use bijective maps and standard methods of counting to find the size of sets of combinatorial objects.
- Be able to use generating functions to count.

Competence

- Know when mathematical concepts in other courses relate to groups and rings and can use group theoretic prope rties outside of group theory, such as in linear algebra and number theory.
- Be able to work with generalizations of groups, such as semi-groups, groupoids and others.

Course assessment

Final exam 50%
Final project: 20%
Weekly tutorials: 30%

Course workload

36 hours lecture
16 hours exercise sessions
3 hours final exam
25 hours exam preparation
68 hours homework
20 hours lecture preparation

Reading Material

Abstract Algebra: Theory and Applications, by Tom Judson.
Combinatorics: The art of counting, by Peter Cameron.

18th of October
*Subject to change

T-219-REMO Real-time Models

Credits: 6 ECTS
Year: .1st year
Semester: Spring semester
Level of course: N/A
Type of course: Mandatory in BSc in Discrete Mathematics and Computer Science
Prerequisites: T-111-PROG, Programming
Structure: 3. week course
Lecturer: Luca Aceto

Description

Computing systems are everywhere in modern society; they are becoming increasingly sophisticated and they control key aspect s of our lives. In fact, computation is even more widely present in our world than most people realize! Think, for instance, of embedded computing devices,
such as those that control ABS systems in cars, the temperature of our houses or the functioning of mobile phones. This populati on of 'effectively invisible' computers around us is embedded in the fabric of our homes, shops, vehicles,
farms and some even in our bodies. They help us command, control, communicate, do business, travel and entertain ourselves, a nd these
'invisible' computers largely outnumber the desktop or laptop computers we see each day. In light of the increasing complexity o f such computing devices, and of the fact that they control important, when not altogether safety critical, operations, it is important to adopt high standards
of quality in their development and validation. However, when dealing with software controlled devices, we still accept routinely that
such systems crash and must be rebooted. In fact, we would be surprised if we did not have to send error reports to software ma nufacturers! Come to think of it, software-controlled devices are just about the only products we engineer for
which we accept this level of brittleness. You do not enter your car each day expecting it to stop and ready to send an error report to the car manufacturer, do you? Do software-controlled systems have to be more unreliable than cars, say?
A key scientific challenge in computer science is to design and develop computing systems that do what they were designed to d o, and do so reliably. In order to meet the challenge of building dependable systems, computer scientists are increasingly using m odel-
based approaches to their design and validation. This means that, before actually constructing a system, one follows the time-honoured engineering approach of making a model of its design and of subjecting the model to a thorough analysis, whose ultim ate aim is to certify that the design embodied by the model meets its intended specification.The aim of this course is to introduce the basic ideas underlying the model of timed automata, a graphical formalism for the description of real-
time computing systems due to Rajeev Alur and David Dill. During the course, you will use the model to describe algorithms, gam es, scheduling problems and other fun scenarios with relevance to computer science, and to analyze the behaviour of the system s you have modelled using the automatic verification tool Uppaal. Uppaal is
an integrated tool environment for the description, validation and verification of real-
time systems modelled as networks of communicating timed automata, extended with data types. Summing up, this is a course in which you will be introduced to a little neat theory with real impact on the practice of the development of computing systems in a world that increasingly depends on the quality of software-controlled devices. Can you do without this knowledge?

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Be able to identify the concepts of parallel and interactive systems.
- Be familiar with systems that are subject to time.
- Be familiar with the modeling tool Uppaal and the logic associated with it.

Skills

- Be able to do Uppaal models of simple systems that do not depend on time.
- Be able to describe simple properties such as systems with associated logic and demonstrate that they hold.
- Be able to do Uppaal models of simple systems that are subject to time.

Competence

- Be able to understand, analyze and create models of real systems Uppaal and explain that the model is correct with the help of the tools.

18th of October
*Subject to change

- Be able to understand and use such models as the basis for proper implementation.
- Be able to use their own knowledge and use other similar tools when appropriate.

Course assessment

Projects – 70%
Final exam - 30%

Course workload

150-180 hours per semester

Reading material

Slides from lecturer

T-622-ARTI Artificial Intelligence

Credits: 6 ECTS
Year: .2nd year
Semester: Spring semester
Level of course: 3. First cycle, advanced
Type of course: Elective course
Prerequisites: T-301-REIR Algorithms
Structure: 12. week course, on site
Lecturer: Dr. Stephan Schiffel and Dr. Adín Ramíres Rivera

Description

Artificial intelligence (AI) is devoted to the computational study of intelligent behaviour, including areas such as problem-solving, knowledge representation, reasoning, planning and scheduling, machine learning, perception and communication. This course gives an overview of the aforementioned AI subfields from a computer science perspective and introduces fundamental solution techniques for addressing them. On the completion of the course, the students should have a good overview of the field of artificial intelligence (AI) and a thorough understanding of the fundamental solution methods used to attack a wide variety of AI-related problems. In addition, the student should have gained experience building a small special-purpose AI system.

Learning Objectives

Upon completion of the course, the student should:

Knowledge

- Be able to name methods for modeling and reasoning with imperfect information, such as Bayesian networks.
- Be able to describe problems and possible solutions for acting in continuous, partially observable and dynamic env ironments.
- Be able to describe different types of machine learning methods.

Skills

- Be able to classify autonomous agents and environments that agents operate in.
- Be able to compare and implement different search methods and optimizations for problem solving in single-agent and adversarial environments.
- Be able to use logic for knowledge representation and problem solving.

Competence

- Be able analyze a problem, select a well-suited AI method and create an agent to solve that problem.

Course assessment

Assignments , quizes and labs 20%
2 X 20% projects
Final exam 40%

Course workload

54 hours in class (lectures, lab classes),
3 hour exam,
20 hour exam preperation,
25 hour homework assignments,
50 hour programming assignments.

Reading material

Artificial Intelligence: A Modern Approach, Stuart Russell and Peter Norvig, 3rd edition - 2010

18th of October
*Subject to change

T-315-IUPP Introduction to experience design

Credits: 6 ECTS
Year: .2nd year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Elective
Prerequisites: None
Structure: 3. week course, On-site, Lectures and problem-solving lessons
Lecturer: Margrét Dóra Ragnarsdóttir

Description

The course will be comprised of three interrelated parts.
Part one covers user research, the second part covers experience design and the third one covers interaction design.
The students will build on one product idea throughout the course and that way each part builds on the next Part 1: User Research This part of the course examines what can be learned from those that
are supposed to be use the product/service that we are designing and how we can use that information to create
a product/service that is useful. We will cover various techniques for
user research such as interviews, observations, ethnography, usability testing, web statistics, focus groups, surveys and others. W e will discuss the strengths and weaknesses of each technique and what information they can glean, when they are appropriate t o use and how to incorporate the results into the product development. Students will build skills in setting up and executing user research and recognize which technique to use when. Part 2: Experience design When designing a product/service it is
not enough just to consider the digital aspect of the service (the app/website). In order to give great service you have to understa nd all the touch points with the customers throughout their journey. This is what we call experience design. In this part
of the course we'll discuss how to define the beginning and end of
a customer journey and how we define how the organization interfaces with its customers depending on where they are in the jo urney, whether it is in person, over the phone, or through
a digital interface. Students will build skills in understanding and defining an experience through a journey map. Part
3: Interaction design The last part
of the course covers interaction design. We will discuss how to create wireframes, what tools to use, how to iterate and test all t he way from
a rough sketch on paper until you have a fully designed interface in the appropriate branding. Topics include wireframes, tools (including Sketch, Invision, some Adobe products), grids for layout, call to action, copy, color and fonts. Students will build skills in sketching a digital user interface and iterating on it until it is ready for development.

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Explain the basic concepts and methodology of user research.
- Explain the basic concepts and methodology of experience design.
- Explain the basic concepts and methodology of interaction design

Skills

- Apply experience design methods in product development.
- Evaluate which user research technique will give them the information they need.
- Setting up and executing user research effectively
- Communicate the user research and design effectively to the team that will execute the development
- Create a journey map
- Create wireframes

Competence

- Know the benefits and drawbacks of various methods of experience design
- Execute and incorporate user research into product development.

Course assessment

Attendance: 15%

18th of October
*Subject to change

Participation in class: 15%
Assignment and Report (group): 50%
Presentation of assignment (group). 20%

Course workload

36h of lectures
24h exercise classes
90h projects and reading
Total 150 h

Reading material

Slides from lecturer

18th of October
*Subject to change

T-316-UPPL The Information and Technology Society

Credits: 6 ECTS
Year: 2nd year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Elective
Prerequisites: None
Structure: 12. week course
Lecturer: Ásrún Matthíasdóttir

Description

This course exams the social, legal and ethical topics related to information and communication in modern society.
The main themes of the course will be: Privacy and security, Intellectual wealth, Computer crime and other legal issues, Comput ers and risk, Ethical base, instructions and warranty, Effects of computerization on the workplace, work practices, teamwork and professional culture, E-commerce and E-government, Society,
internet culture and the impact on health and education Emphasis will be placed on training students to write reports with their projects.

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Be able to describe the advantages and disadvantages of the information society
- Be aware of the social, ethical and philosophical impact of computerization.
- Be able to understand the impact of information and communication technology in homes, schools, workplaces, recreation, health and education.
- Be able to identify key moral questions relating to computers and who is responsible when working with computers.
- Know the legal environment, information technology and legal issues such as privacy and security, intellectual wealth and computer crime.

Skills

- Be able to write reports and articles on topics related to computers.

Competence

- Be able to follow the development of the information society and be able to evaluate it critically.
- Be able to articulate a vision of the desired effect of computerization.

Course Assessment

Theme project– 50%
Reading diary – 40%
Report – 10%

Course workload

150-180 hours per semester

Reading material

Books, research papers, online presentations

18th of October
*Subject to change

T-403-FORC Programming in C++

Credits: 6 ECTS
Year: 2nd year
Semester: Spring semester
Level of course: 1. First cycle, introductory
Type of course: Elective
Prerequisites: Calculus I, Discrete Mathematics for engineering and Algortithms or Calculus and statistics, Discrete Mathematics I and Algorithms.
Structure: 3-week course, on-site
Lecturer: KYeari Halldórsson

Description

The course is intended for students to increase their knowledge and training in general programming and to learn the programming language C++ in preparation for courses and projects that demand use of C++ or related programming languages. Students will learn differences between compiled and scripted programming languages, the difference between loosely and strongly typed programming languages and different methods of memory allocation and argument passing. Students will finish several assignments where they will use pointers and dynamic memory allocation, multithreading, object orientation, inheritance and polymorphism in their C++ programs. Students will also learn to use the C++ standard template library for data storage and manipulation in their programs. Furthermore, they will practice some specific programming methods such as function pointers or bit-shifting and design patterns such as the singleton design pattern or other similar methods.

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Students understand the main differences between compiled languages and scripting languages.
- Students understand the main differences between loosely and strongly typed programming languages.
- Students know and understand different argument passing methods.
- Students know and understand different methods of memory allocation in programs.

Skills

- Students can write code in C++ and compile and run the programs using standard C++ compilers.
- Students can utilize both call-by-value and call-by-reference correctly in C++ programs.
- Students can write programs in C++ using pointers and dynamic memory allocation.
- Students can write multithreaded programs in C++.
- Students can write programs that use the standard.

Competence

- Students can write, compile and run programs, written in the C++ language, utilizing C++ specific methods and various general programming methods to solve diverse computational problems.

Course Assessment

Programming assignments (5x10%): 50%
Participation: 5%
Quizzes: 5%
Final exam: 40% (Students have to pass final exam with 48 points or more for other grades to count)

Course workload

36 hours lectures and quizzes,
10-20 hours practice exercises,
60-100 hours programming assignments,
20 hours exam preparation,
3 hours exam.

Reading material

Slides from lecturer.

I-406-IERP Introduction to ERP Systems (ERP)

Credits: 6 ECTS
Year: 3rd year
Semester: Fall semester
Level of course: 1. First cycle, introductory
Type of course: Mandatory in BSc in Computer Science-Business minor
Prerequisites: T-111-PROG, Programming
Structure: 12. week course, on-site
Lecturer: Sigríður Jónsdóttir and **Þórdís Magnús**dóttir

Description

The largest investment companies make in information technology is Enterprise Resource Planning, ERP. ERP is another way to describe the interaction between the processes and technologies in operation.
The processes and technologies that
are included are planning, procurement and product management, human resources, finances, inventory and sales. In recent yea rs there has been a lot of change in this sector with systems becoming more versatile and powerful than before, and extends into the activity of companies. According to a recent survey by Gartner´s, companies investment in information technology business s ystems will continue to increase in the coming years.
The main obstacle for growth in this sector relates to the lack of personnel with expertise in this area, which is a combination of sustained knowledge of information technology and major business processes.
The largest players in this market are SAO,
Oracle and Microsoft, and the working environment is global. This course seeks to create
a good base of knowledge for anyone who wants to further study the function and development of
ERP systems. In the course different ways of implementation and the impact ERP systems have on business operaions will be disc ussed. The teaching in this course will be based on Microsoft Dynamics NAV. By the end of the course students will be able to understand the function and main features of
ERP systems. During the course we will also "look under the hood" and give students the opportunity to work in the technology e nvironmentother programming and mathematical courses, for testing conjectures, drawing graphs, etc.

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Be able to describe the functions and use of ERP systems and their development over time
- Be able to define key components of ERP systems and describe their context.

Skills

- Be able to work with the main system components in Microsoft Dynamic NAV
- Competence Be able to program simple functionality with the ERP system.

Competence

- Be able to design processes within the ERP system

Course assessment

Group assignment: 20%
Assignments: 35%
Final project/essay: 15%
Small assignments and quizzes: 25%
Group project in GIT: 5%

Course workload

36h of lectures
24h exercise classes
90h projects and reading
Total 150 hours

Reading material

Material from lecturers, Dynamics 365 Business Central (BC)Windows Client.

T-414-AFLV Effective programming and probem solving

Credits: 6 ECTS
Year: 2nd/3rd year
Semester: Spring
Level of course: 1. cycle, advanced
Type of course: Elective
Prerequisites: T-111-PROG Programming, T-110-VERK, Problem Solving, T-301-REIR, Algorithms
Structure: Three-week course
Lecturer: Arnar Bjarni Arnarsson

Description

Computer scientists often have to deal with challenging tasks requiring both fast algorithmically linear solutions and efficie nt coding. This is one of the reasons why programming puzzles and oral exams are used so often in job interviews when appl ying to the strongest companies or graduate schools.
The goal of this course is to make students better in solving algorithm tasks and acquire skills in a fun competition environm ent. The training exercises include challenges from international competitions, such as the ICPC and ToCoder.
Other main tasks is to make decisions under strict time limits. Training will also be done in cooperation and dialogue, and by utilizing scarce resources (e.g. , one computer for each team with a limited time).
The course is intended to be informative, but at the same time fun. The material that
will be covered includes much of the material in the algorithm's courses (e.g., data structures, dynamic programming, netw ork search, and share-and-rule), but the emphasis will be different: how we perceive what solution method
is applicable, the choice of design decisions when project is brought into the framework of solution method, and how this is implemented in the code. Students will tackle with applying and refining the core methods of transferring demonstration s olutions into a programming solution.

Learning Objectives

On completion of the course, students should be able to:

Knowledge

- Be able to describe algorithms, data structures and projects in a clear manner.

Skills

- Be able to develop the correct implementation of a well-defined algorithm or data structure.
- Be able to compare the difficulty of different tasks.
- Be able to report on the effectiveness of different solution approach for the given task to determine which methods are effective enough for the given conditions.
- Be able to apply various types of algorithms, such as greedy methods, dynamic programming, share-and-rule and Heuristic to solve given tasks.
- Be able to communicate and work in a group setting to solve problems under time pressure.

Competence

- Be able to develop solutions to projects that have not been seen before.

Course assessment

Problem sets:70%
Problem sessions:10%
Final exam: 20%

Course workload

36 hours lecture,
80-120 hours exercises and programming assignments,
20 hours exam preparation,
3 hours exam.

Reading material

Slides from lecturer.Page Break

18th of October
*Subject to change

T-417-TOOR Computer Security

Credits: 6 ECTS
Year: 2nd/3rd year
Semester: Fall
Level of course: 1. cycle, advanced
Type of course: Elective
Prerequisites: T-215-STY1, Operating systems, T-409-TSAM, Computer networks
Structure: Three-week course
Lecturer: Níels Ingi Jónasson

Description

This course covers the section of information security that covers software and hardware and their use. We will dive into common vulnerabilities in software and web services, how attackers exploit them, and how it is possible to defend systems against such attacks. We will also cover network security, and many other attacks used by hackers today. The goal of this course is for students to gain a deep understanding of the core fundamentals of cyber and information security and understand the mindset of the hacker well enough to prevent attacks.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Can explain the ideas and implementations of common programs used to exploit vulnerabilities in well protected software systems to obtain some privileges

Skills

- Can explain the most common vulnerabilities of today, both within software and networking.
- Can identify vulnerabilities within real world software

Competence

- Can write programs to exploit vulnerabilities within vulnerable systems.
- Can explain in detail methods to prevent common vulnerabilities and their exploitation within software and networking

Course assessment

Assignments and final project.

Course workload

48 hours lecture,
24 hours lab classes,
24 hours lecture preparation,
72 hours self-study and practical assignments,
12 hours exam preparation and exam.

Reading material

Slides from lecturer.

Page Break

18th of October
*Subject to change

T-419-CADP Concurrent and distributed programming

Credits: 6 ECTS
Year: 3rd year
Semester: Spring semester
Level of course: 1. cycle, intermediate
Type of course: Mandatory in BSc in Software Engineering
Prerequisites: T-215-STY1, Operating Systems, T-301-REIR, Algorithms
Structure: 12. week course, on-site
Lecturer: Marcel Kyas

Description

Multi-Core machines, networks of interconnected computers and heterogeneous computing environments have become ubiquitious. Writing programs that utilize these systems 's resources to its fullest involves writing multi-threaded and distributed programs. In this course, participants learn to write such programs in C using the pthreads API and in the Go programming language. The Go programming language is a concurrency-oriented programming language developed by Google for concurrent and distributed applications. They learn to avoid unintended nondeterministic effects and deadlocks and they learn to structure concurrent and distrbibuted programs. The basics of threads, processes, semaphores and mutexes will be repeated. Then, patterns are described to structure common algorithms for concurrent execution and understand the basic architectures. Programming with monitors and with transactional memory will be considered. Distributed message passing systems and middleware will be discussed. Participants learn to structure distributed applications and understand their architecture.
They will also consider coordination methods that describe how the activities of the processes in a distributed system achieve a common goal. At the end, participants are able to demonstrate a concurrent application, understand the way it is constructed and be able to justify the functional and nonfunctional properties of the application.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be able to describe the need for concurrency for programs.
- Be able to explain the challenges of concurrency (non-determinism, combinatorial explosion, problems of testing).
- Be able to explain the difference between processes and threads.Be able to explain conditions that lead to deadlo ck.
- Be able to explain strategies for deadlock avoidance.
- Be able to enumerate memory models, describe their purpose and how they work, and write programs that are data-race-free consistent.
- Be able to explain the problems of shared variable concurrency and how they are addressed in a distributed model.
- Be able to describe the need for coordination and some coordination methods.
- Be able to describe models for fault-tolerance and resilience.
- Be able to explain the actor model and apply it in applications.

Skills

- Be able to write a multi-threaded program using semaphores, mutexes, and read-write locks.
- Be able to write concurrent programs using monitors.
- Be able to write concurrent programs using transactional memory.
- Be able to structure distributed applications with a client-server, 3-tier, n-tier, peer-to-peer, and space based architecture.
- Be able to write distributed applications with message passing.

Competence

- Assessment Be able to design a concurrent and fault-tolerant application.
- Be able to critique the essence of a concurrent programming solution and its expression with language elements.

Course assessment

Quizzes: 12%
Group assignments:36%

18th of October
*Subject to change

Programming assignments:18%
Final exam: 34%

Course workload

30 hours lecture
30 hours lab classes
45 hours self-study
30 hours assignment
30 hours programming project
12 hours exam preparation and exam

Reading Materialmaterial

Principles of Concurrent and Distributed Programming, 2/E
M. Ben-Ari, Addison-Wesley, Second Edition, 2005
Go in Practice, Matt Butcher and Matt Farina, Manning Publications,2016

18th of October
*Subject to change

T-427-WEPO Web-Programming II

Lecturer: Arnar Leifsson
Year: 2nd or 3rd year
Semester: Spring semester
Level of course: 1. cycle, intermediate
Type of course: Elective
Prerequisites: T-213-VEFF, Web-Programming
Structure: 12. week course, on-site
Lecturer: Arnar Leifsson

Description

Web programming II is an advanced course on developing for the web. It embarks upon subjects such as JavaScript, CSS3, React and more. In an ever-changing world of web development there is a constant need for individuals which have mastered the skills of web development. This course will provide you with the materia l you will need to start the journey of mastering those skills.

Learning outcomes

Upon completion of the course, the student should:

Knowledge

- Student knows the difference between SPA applications and server-side rendering technologies such as MVC.
- Student knows the HTML5 standard, and APIs provided.
- Student knows various client-side libraries and their differences.
- Student knows JavaScript on a deep level and difference between JavaScript and other object-oriented programming languages.
- Student knows when Redux is useful and when it is not.

Skills

- Student can write complex JavaScript code.
- Student can use CSS3 to create complex styles.
- Student can use CSS Grid and Flexbox to create complex layouts.
- Student can write a React application from start to end.
- Student can use REST APIs to retrieve data in their React application.
- Student can use Redux for state management in their React application.
- Student can integrate Socket.io in their React application.

Competence

- Assessment Know what to consider when building a website in order for it to be accessible on multiple devices.
- Know what the advantages and disadvantages are for websites that have custom made client applications/apps.

Course assessment

Small assignments: 15%
Large assignments: 45%
Final assignment: 40%
Extra assignments: 4%

Course workload

36 hours lectures – approximately 3 hours per week,
8 hours individual small assignments (4 in total),
30 hours large assignments (3 in total),
6 hours code demonstration lectures (6 total),
15-20 hours final assignment – individual.

Reading Material material

David Flanagan, JavaScript: The Definitive Guide, 2011, O'Reilly
ÚtgáfuYear: 2011 og,
Fullstack React, Anthony Accamazzo, og fl., Fullstack.io

18th of October
*Subject to change

T-430-TOVH Developing Open-Sourced Web Solutions/Software

Credits: 6 ECTS
Year: 2nd and 3rd year
Semester: Fall semester
Level of course: 1. cycle, intermediate
Type of course: Elective
Prerequisites: T-201-GSKI, Data Structures, T-213-VEFF, Web-Programming and T-303-HUGB, Software Engineering
Structure: 3. week course
Lecturer: Hilmar Kári Hallbjörnsson

Description

In the course Developing Open-Source Web Software we're going to dip our toes into the open source community and see how it works.
We will discover why choosing open source web software can be a better choice than starting from ground up. Why belonging to a community of tens of thousand, hundreds of thousand or even a million skilled professionals in the same profession as yourself will strengthen you as a developer. And also why it's so great that you don't need to write yet another authentication function!
We also take a look at how errors are reported and how they are handled. We will investigate previously reported errors and try to fix them, therefore putting our weight in to improve the software. Finally we will take a look at how various parts of the software are documented and what we can do to make it better.
We're going to take a look at the Drupal CMS (Content Management System) and research it to the bone. Both in how the system works itself and also how the Drupal community is built up and how each and every member in the community has a voice.

Learning outcomes

Upon completion of the course, students should be able to:

Knowledge

- Have knowledge of the basic definition of open source software.
- Have knowledge of the programming standards that are put forth in each and every software project.
- Have knowledge of different types of software licenses.

Skills

- Report bugs in open source software.
- Participate in discussion on bug solutions and extensions.
- Improve software that was written by others.
- Adapt to the standards and work procedures that was chosen by the project.

Competence

- Estimate and propose changes on open source software.
- Write his/hers own solution. Either a bug fix or an extension to a functionality of an open source software.
- Estimate and describe pros and cons of an open source software.
- Accept code from others, analyze it and improve.

Course assessment

Assignments and oral exam

Course workload

TBA

Reading Material material

Drupal 9 Module Development - Daniel Sipos

18th of October
*Subject to change

T-431-HANE Practical Networks

Credits: 6 ECTS
Year: 2nd/3rd year
Semester: Fall
Level of course: 1. cycle, introduction
Type of course: Elective
Prerequisites: T-107-TOLH, Computer Architecture, T-215-STY1 Operationg systems
Structure: Three-week course
Lecturer: Not defined

Description

The importance of networks is much more than most people realize. If everything is okay no one knows of their existence, but in the event of failures and problems in networks this can affect one's work and play that is involved online. Knowledge of how the network works and is structured is missing, even for those who use it the most, like programmers and system administrators. The evolution of technology means that the importance of networks is increasing, we now see communications being moved to the network and internet. The network is thus becoming more part of our security and coordination. The foundation of all communications is networks and is therefore essential to have an understanding and thorough knowledge of the functionality and possibilities. This course seeks to create a solid foundation that will be useful for anyone intending to establish themselves in information technology. The course is part lecture but mostly it is project based, which utilize the knowledge gained from the lecture. The objective is to teach design and implementation of networks, how requirements of performance and accessibility influence implementation of networks. We go over what is necessary to design and implement a network. This is broken into three parts: 1. Wired communication: Network equipment (Routers, switches), X area networks and protocols 2. Wireless communication: UMTS, 802.11, communications, antennas, wireless security 3. Security: L2/L3 Security, communications, VPN, encryption/decryption, firewalls and IPS/IDS. At the end of the course students have created a coherent network which include all previously mentioned parts.

Learning Objectives

Upon completion of the course, students should be able to:

Knowledge:

- Be able to describe the importance of networks and good installation for their business operations.
- Be able to describe the structure of networks and the equipment that the network consists of.
- Be able to describe what the trend has been in network systems and how they are likely to develop in the future

Skills

- Be able to design and set up a simple network, both wired and wireless.
- Be able to define and apply basic safety methods for networks

Competence

- Be able to identify the needs for performance and security of networks.
- Be able to report common defects and faults in networks and improved them.

Course assessment

TBA

Course workload

TBA

Reading material

Slides from lecturer

18th of October
*Subject to change

T-445-GRTH Graph Theory

Credits: 6 ECTS
Year: 2nd or 3rd year
Semester: Spring semester
Level of course: N/A
Type of course: Mandatory in Discrete Mathematics and Computer Science
Prerequisites: T-101-STA1 Calculus, T-301-REIR Algorithms
Structure: 12. week course
Lecturer: María Óskarsdóttir

Description

The course provides a foundation in statistical inference and computational thinking using a hands-on approach with Python using real-world data-mining applications. It also discusses social and privacy issues surrounding data analysis.

Learning Objectives

Upon completion of the course, students should be able to:

- Be familiar with discrete and continuous probability and probability distributions.
- Be familiar with confidence intervals and hypothesis testing.
- Be familiar with correlation and regression in multiple dimensions.
- Be familiar with fundamentals of data mining, including preprocessing, visualizing and, modelling of data. Be familiar with fundamental predictive analytical modelling techniques.
- Be able to calculate discrete probability using techniques such as permutations and combinations.
- Be able to calculate expected value and standard deviation for discrete probability distributions
- Be able to compute probabilities for discrete and continuous variables, using for example the binomial, normal and the t-distributions.
- Be able to compute confidence intervals and test hypotheses. Be able to compute the correlation coefficient.
- Be able to do regression in multiple dimensions using data mining tools.
- Be able to do classification on nominal and numerical data using data mining tools.
- Be able to apply hypothesis testing to analyze sets of measured data.
- Be able to build and visualize predictive models from noisy real-life data sets using modern data analysis tools and libraries.

Course assessment

Assignments (15%)
Midterm (15%)
Topic presentations (20%)
Topic quizzes (10%)
Final project (40%)

Course workload

30 hours lectures
16 hours exercise sessions
2 hours midterm exam
16 hours midterm exam preparation
24 hours work on assignments
25 hours work on topic presentation
5 hours work on topic quizzes
50 hours work on final project

Reading material

Python for data analysis, Wes McKinney
Learning predictive analytics with Python, Ashish Kumar

18th of October
*Subject to change

T-488-MAPP Mobile App Development

Credits: 6 ECTS
Year: 3rd year
Semester: Fall semester
Level of course: N/A
Type of course: Elective
Prerequisites: T-201-GSKI, Data Structures, T-213-VEFF, Web-Programming
Structure: 3. week course
Lecturer: Arnar Leifsson

Description

This course introduces app software development for mobile devices. The concepts studied are applied in a practical group project taking an application through a complete development cycle.

Learning outcomes

Upon completion of the course, students should be able to:

Knowledge

- Know the fundamentals of app development, including an app's life-cycle.
- Know best app design and implementation practices.
- Know how to program graphical user interfaces and touch screen interactions.
- Know different ways for apps to retrieve, store and share data.
- Know how to program responsive apps using asynchronous flow.

Skills

- Be able to use a selected app software development environment effectively.
- Be able to make interactive apps that handle all aspects of the life-cycle, run gracefully on different sized devices, e.g. smartphones and tablets, and that effectively retrieve, store an d share data..
- Be able to work in groups on developing non-trivial apps.

Competence

- Be able to develop robust and responsive non-trivial interactive apps for different sized devices that behave in accordance with relevant standards and guidelines.

Course assessment

First week - 30%
Second week – 30%
Third week – 30%
Video demonstration 10%

Course workload

28 hours lectures
75 hours assignments

Reading Material

Slides from lecturerPage Break

18th of October
*Subject to change

T-498-GAGR Data Analysis

Credits: 6 ECTS
Year: 3rd year
Semester: Fall semester
Level of course: N/A
Type of course: Elective
Prerequisites: T-201-GSKI, Data Structures, T-213-VEFF, Web-Programming
Structure: 3. week course
Lecturer: Arnar Leifsson

Description

This course introduces app software development for mobile devices. The concepts studied are applied in a practical group project taking an application through a complete development cycle.

Learning outcomes

Upon completion of the course, students should be able to:

Knowledge

- Know the fundamentals of app development, including an app's life-cycle.
- Know best app design and implementation practices.
- Know how to program graphical user interfaces and touch screen interactions.
- Know different ways for apps to retrieve, store and share data.
- Know how to program responsive apps using asynchronous flow.

Skills

- Be able to use a selected app software development environment effectively.
- Be able to make interactive apps that handle all aspects of the life-cycle, run gracefully on different sized devices, e.g. smartphones and tablets, and that effectively retrieve, store an d share data..
- Be able to work in groups on developing non-trivial apps.

Competence

- Be able to develop robust and responsive non-trivial interactive apps for different sized devices that behave in accordance with relevant standards and guidelines.

Course assessment

First week - 30%
Second week – 30%
Third week – 30%
Video demonstration 10%

Course workload

28 hours lectures
75 hours assignments

Reading Material

Slides from lecturer

18th of October
*Subject to change

T-504-ITML Introduction to Machine Learning

Credits: 6 ECTS
Year: 3rd year
Semester: Fall
Level of course: 1. cycle, introduction
Type of course: Selection
Prerequisites: T-301-REIR, Algorithms, T-317-CAST, Calculus and Statistics, T-419-STRA2, Discrete Mathematics II
Structure: Twelve-week course
Lecturer: Dr. Stephan Schiffel

Description:

This course presents an overview of the field of machine learning, which deals with finding patterns and rules in large datasets. Such rules can then be used to predict outcomes of future events, for example with the aim of improving decision making in a wide range of business and manufacturing disciplines. In this course we will study machine learning techniques for classification, clustering, and association analysis as well as other selected techniques. In addition to introducing the underlying theory the methods will be used to solve practical problems.

Learning outcomes:

After completion of the course the student will hold a knowledge, skills and competence of:

Knowledge:

- Know how data mining is carried out.
- Recognize different types of training data and how to deal with common problems that arise, such as incomplete data.
- Be familiar with key algorithms and models used for classification, including decision trees, set of rules, Naïve Bayes, neural networks and support vector machines.
- Know the basic algorithms used with clustering, including K-means.
- Know the basic algorithms used to find relationships in data (e.g., association analysis).
- Be familiar with basic ideas behind evolutionary and reinforcement learning.

Skills:

- Be able to use software tools and programming libraries for data mining to categorize and cluster data.
- Be able to set up problems and apply data mining techniques to solve them.

Competences:

- Be able to determine the mechanical data mining strategies best suited to the solution of various practical problems, and be ready ro use data mining tools and libraries to their solution

Course assessment:

Homework assignments and in-class quizzes 25%
Two projects 30%
Final exam 45%

Course workload:

54 hours in class (lectures, lab classes),
3 hours exam,
20 hours exam preparation,
5 hours quizzes,
20 hours homework assignments,
50 hours programming assignments.

Reading Material:

Lecture notes provided by teacher.

18th of October
*Subject to change

T-505-ROKF Logic in Computer Science

Credits: 6 ECTS
Year: 2nd year
Semester: Spring semester
Level of course: N/A
Type of course: Capstone in BSc in Discrete Mathematics and Computer Science
Prerequisites: T-117-STR1, Discrete Mathematics I or T-419-STR2, Discrete Mathematics II
Structure: 12. week course
Lecturer: Anna Ingólfsdóttir

Description

Logic has been called "the calculus of computer science". The argument is that logic plays a fundamental role in computer science, similar to that played by calculus in the physical sciences and traditional engineering disciplines. Indeed, logic plays an important role in areas of Computer Science as disparate as architecture (logic gates), software engineering (specification and verification), programming languages (semantics, logic programming), databases (relational algebra and SQL), artificial intelligence (automatic theorem proving, multi-agent systems, knowledge and belief), algorithms (complexity and expressiveness), and theory of computation (general notions of computability). See, for instance, the slides available at http://www.ru.is/faculty/luca/SLIDES/logic-and-cs.pdf for more information. This course provides the student with a thorough introduction to computational logic, covering the topics of syntax, semantics, decision procedures and formal systems for various logics that play a crucial role in applications in computer science, namely propositional and first-order logic, and modal and temporal logics. The material is taught from a computer science perspective, with an emphasis on the use of logic as a specification language and general-purpose problem-solving tool in computer science. As part and parcel of the course, we shall introduce various logic-based software tools and the algorithms and data structures underlying them; examples include BDD-based tools, SAT solvers and model checkers. The goal is to prepare the students for using logic as a formal tool in computer science.

Learning outcomes

After completion of the course the student will hold a knowledge, skills and competence of:

Knowledge

- To use suitable logical languages (such propositional and first-order logic as well as modal and temporal logics)
- To model computer science related problems
- To solve problems using techniques from logic and tools embodying those techniques
- To apply logic to formalize reasoning in their own fields of interest within computer science.

Course assessment

Home assignments – 50%
Oral exam – 50%

Course workload

36h of lectures
24h exercise classes
90 - 120h projects and reading
Total 150 – 180 hours

Reading material

Slides from lecturer

18th of October
*Subject to change

T-511-TGRA Computer Graphic

Credits: 6 ECTS
Year: 3rd year
Semester: Fall
Level of course: 1. cycle, introduction
Type of course: Elective
Prerequisites: T-301-REIR, Algorithms
Structure: Twelve-week course
Lecturer: KYeari Halldórsson

Description:

Computer graphics is an increasing part of the projects of today´s programmer. The first part of this course covers the use of the OpenGL library, vector tools for graphics, transformations of objects and polygonal meshes. The second part deals in more detail with three-dimensional drawing with emphasis on perspective, depth, light and colour. Finally, several issues regarding the implementation of a renderer are presented, in addition to curve and surface design. During the course students build several programs related to the course material.

Learning outcomes:

After completion of the course the student will hold a knowledge, skills and competence of:

Knowledge:

- Be familiar with the algorithms and calculations used when three-dimensional images are drawn on screen in real time (pipeline graphics), including, model transformations, perspective transformations, lighting, shading, clipping and rasterization.
- Be familiar with methods in OpenGL that implement these algorithms and calculations and how they are used in graphics applications such as computer games (OpenGL pipeline).
- Know how the flow in a graphical real-time application (i.e. computer game) is implemented, with respect to input, movement and drawing.

Skills:

- Be able to use the OpenGL standard to draw a three-dimensional image on a screen.
- Be able to implement a drawing loop which draws a motion picture, frame by frame, in real time.
- Be able to implement a programming loop that receives input and output, moves things, makes decisions and draws each frame with respect to camera angles and objects in a three-dimensional space.

Competences:

- Be able to implement three-dimensional video games and real time animations with the OpenGL standard.

Course assessment:

Hand in assignment 10%
Programming assignments 50%
Final exam 40%

Course workload:

18 hours lectures
24 hours practice and exercises
10 hours homework
50-80 hours programming assignments
20 hours exam preparation
3 hours exam

Reading Material:

Lecture notes provided by teacher.

18th of October
*Subject to change

T-513-CRNU Cryptography and Number Theory

Credits: 6 ECTS
Year: 3rd year
Semester: Fall semester
Level of course: N/A
Type of course: Mandatory in Discreta Mathematis and Computer Science
Prerequisites: T-101-STA1, Calculus I, T-103-STST, Discrete Mathematics for Engineering, T-301-REIR, Algorithms, T-317-CAST, Calculus and Statistics, T-419-STR2, Discrete Mathematics II
Structure: 12. week course
Lecturer: Christian Bean

Description

This course covers the basics of cryptography and number theory, starting with classical ciphers and the tools from number theory necessary for doing cryptography. Symmetric and asymmetric ciphers will be covered. Some topics from groups, rings and fields will be introduced and used, especially when looking at elliptic curve cryptography. There will be some programming exercises in addition to standard mathematical homework. The programmlanguage Sage will be used.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Know the purpose of cryptography and its uses throughout history.
- Know the basics of number theory, especially relating to cryptography.
- Know the Sage programming language, especially how to implement algorithms from number theory and cryptography.
- Know the most common algorithms used in cryptography, e.g. the RSA public key system.
- Know the basics of information theory.
- Know the basics of finite fields and how they are used in cryptography.
- Know the basics of elliptic curves and how they are used in cryptography.
- Know how cryptography is applied, e.g. in multi-party computation, zero knowledge proofs, digital cash and voting systems.

Skills

- Know how to use simple cryptographic methods to encrypt short texts by hand.
- Be able to write code in Sage to use powerful cryptographic methods to encrypt text.
- Be able to solve number theoretic problems, with pencil and paper, as well as with Sage.
- Be able to implement common algorithms in cryptography, e.g. Euclids algorithm for the greatest common divisor and Diffie-Hellman key exchange.

Competence

- Recognize where to apply the methods of cryptography and which methods are breakable.
- Be able to apply their knowledge of number theory to solve problems in other mathematical courses, especially where algebra is needed.
- Be able to use Sage as a tool in other programming and mathematical courses, for testing conjectures, drawing graphs, etc.

Course assessment

Problem sessions – 30%
Programming assignments - 20%
Final exam - 50%

Course workload

36 hours lecturer,
16 hours exercise sessions,
3 hours final exam,
25 hours final exam,
68 hours homework,
20 hours lecture preparation.

Reading Material: Hoffstein, Pipher and Silverman's Introduction to Mathematical Cryptography 2nd edition.

18th of October
*Subject to change

514-VEFT Web Services

Credits: 6 ECTS
Year: 2nd year / 3rd year
Semester: Fall
Level of course: N/A
Type of course: Elective
Prerequisites: T-202-GAG1, Databases and T-213-VEFF, Web-Programming
Structure: 12. week course
Lecturer: Arnar Leifsson

Description

This course focuses on the subjects associated with web services. It will embark on subjects such as: Web services (WS) in general, HTTP, RESTful WS, RPC WS, .NET Core Web API, NodeJS, ExpressJS, Authentication, Authorization, Microservices, GraphQL and more.
The course will provide you with both knowledge and skills to start diving in to the world of web services.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Student knows how the HTTP protocol works.
- Student knows the difference between RPC and REST.
- Student knows how to setup REST APIs using .NET.
- Student knows how to communicate with a database using an Object Relation Mapper (ORM).
- Student knows how to setup REST APIs using NodeJS.
- Student knows how to communicate with a NoSQL database.
- Student knows the difference between a NoSQL and SQL database.
- Student knows how to setup a GraphQL layer using JavaScript.
- Student knows what microservices are and are not.
- Student knows when microservices are useful and are not.
- Student knows the benefits of using containers for deployment.

Skills

- Student can setup a HTTP message using tools such as Telnet and CURL.
- Student can setup REST APIs with .NET.
- Student can setup REST APIs with NodeJS.
- Student can setup a GraphQL layer with JavaScript.
- Student can setup multiple microservices that communicate with each other using effective strategies such as event collaboration.
- Student can deploy a containerized web service using Docker.
- Student can deploy a whole microservice structure using Docker.

Reading material

Contextal design, 2nd edition by Karen Holtzblatt, Hugh Beyer.

Course assessment

Class assignments – 12 %
Large assignments – 30 %
Small assignments – 25%
Final assignment – 33%

Course workload

36 hours lecturer – approximately 3 hours per week
8 hours individual small assignments (4 in total)
30 hours large assignments (3 in total)
6 hours code demonstration lectures (6 total)
15 – 20 hours final assignment - individual

18th of October
*Subject to change

T-515-NOTH User Centred Software Development

Credits: 6 ECTS
Year: 2nd year / 3rd year
Semester: Fall
Level of course: 1. First cycle, advanced
Type of course: Elective
Prerequisites: T-216-GHOH Software Analysis and Design or T-133-UIAD User Interface Analysis and Design
Structure: 12. week course
Lecturer: Marta Kristín LYearusdóttir

Description

The objective of the course is to teach methods for analyzing, designing and evaluating software systems anticipating the users aspects in software development. Students gain skills in using particular methods for analyzing, design and evaluation user interfaces. Furthermore, other methods for analyzing, designing and evaluating user interfaces are described. Research on user centered software development methods is described, concerning for example when it is best to use each method and how practitioners have ranked the methods.
The focus in the course is on analyzing the digital work environment of users. Methods are used to analyse, design and evaluate software systems taking the digital work environment into consideration.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be familiar with several methods for analysing the users´ needs for software systems.
- Be familiar with evaluation with and without the participation of users.
- Be familiar with guidelines for good user interface design.

Skills

- Be able to describe the vision for a software project and explain it.
- Be able to analyse the context of use for software systems.
- Be able to perform contextual iquiries and interpret the results.
- Be able to design an interface that is based on the users needs and test it with users.
- Be able to perform formal user evaluations on the designs.

Competence

- Be able to choose which user-centered design methods are suitable in different cases.
- Know the advantages and disadvantages of user centred design methods.

Reading material

Contextal design, 2nd edition by Karen Holtzblatt, Hugh Beyer.

Course assessment

Individual assignments – 40 %
Group assignments – 50 %
Attendance/in-class assignments – 10%

Course workload

48 hours – mixture of lectures and problem solving sessions with the lecturer coaching the students,
36 – 48 hours – homework on assignments.

T-519-STOR Theory of Computation

Credits: 6 ECTS
Year: 3rd year or 1st year in Masters program
Semester: Fall
Level of course: 3. First cyvle, advanced
Type of course: Mandatory in BSc in Discrete Mathemetics and Computer Science
Prerequisites: T-103-STST Discrete Mathematics for Engineering Students and T-301-REIR Algorithms or T-419-STR2 Discrete Mathematics II and T-301-REIR, Algorithms
Structure: 12. week course
Lecturer: Antonios Achilleos

Description

The main topic of this course is the theoretical basis of computer science. Various types of finite automata are introduced and connected to the formal definition of a programming language. Turing machines are introduced as a theoretical model for computation. Computability is discussed and the classification of solvable and unsolvable problems. Finally, there is a discussion of complexity classes and the classification of algorithmically hard and easy problems.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- A number of recurring themes, and a set of general principles that have broad application to the field of computer science
- The social, legal, ethical, and cultural issues inherent in the discipline of computing
- That software systems are used in many different domains. This requires both computing skills and domain knowledge.
- Software development fundamentals, including programming, data structures, algorithms and complexity.
- System fundamentals, including architectures and organization, operating systems, networking and communication, parallel and distributed computation and security.
- Mathematics, including discrete structures, statistics, calculus and optimization
- Software engineering principles, including a thorough understanding of software analysis and design, evaluation and testing and software quality and correctness.
- Software engineering processes, including management of complex software development projects.
- Application fundamentals, including information management and intelligent applications.
- Multiple programming language, paradigms, and technologies

Skills

- Know how to apply the knowledge they have gained to solve real problems
- Realise that there are multiple solutions to a given problem and these solutions will have a real impact on people´s lives
- Communicate their solution to others, including why and how a solution solves the problem and what assumptions were made
- Successfully apply the knowledge they have gained through project experience.
- Encompass an appreciation for the structure of computer systems and the processs involved in their constructions and analysis
- Understand individual and collective responsibilities and individual limitations as well as the limitations of technical tools.
- Understand the range of opportunities and limitations of computing.

Competence

- Understand the multiple levels of detail and abstraction
- Recognise the context in which a computer system may function, including its interactions with people and the physical world.
- Able to communicate with, and learn from experts from different domains throughout their careers.
- Possess a solid foundation that allows and encourages them to maintain relevant skills a sthe field evolves.
- To be able to manage their own career development, including managing time, priorities, and progress
- Have developed interpersonal communication skills as part of their project experience

18th of October
*Subject to change

- Work effectively both individually and as a member of teams
- Make effective presentations to a wide range of audience about technical problems and their solutions
- Encompass an appreciation of the interplay between theory and practice.

Course assessment

Assignments – 30 %
Quizzes – 10 %
Midterm Exam – 20 %
Final exam – 40 %

Course workload

36h of lectures
24h exercise classes
90 - 120h projects and reading
Total 150 – 180 hours

Reading material

Introduction to Theory of Computation by Michael Sipser, third edition.

T-535-CPSY Cyber-Physical Systems

Credits: 6 ECTS
Year: 1st year
Semester: Fall
Level of course: Not defined
Type of course: Mandatory in Software engineering
Prerequisites: T-215-STY1, Operating Systems
Structure: 12. week course.
Lecturer: Marcel Kyas

Description

Cyber-physical systems introduces students to the design and analysis of computational systems that interact with physical processes. Applications of such systems include medical devices and systems, consumer electronics, toys and games, assisted living, traffic control and safety, automotive systems, process control, energy management and conservation, environmental control, aircraft control systems, communications systems, instrumentation, critical infrastructure control (electric power, water resources, and communications systems for example), robotics and distributed robotics (telepresence, telemedicine), defense systems, manufacturing, and smart structures. A major theme of this course is on the interplay of practical design with models of systems, including both software components and physical dynamics. A major emphasis will be on building high confidence systems with real-time and concurrent behaviours.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Describe a realtime or hybrid system as a system characterized by a known set of configurations with transitions from one unique configuration (state) to another (state).
- Describe the distinction between systems whose output is only a function of their input (Combinational) and those with memory/history (Sequential).
- Derive time-series behavior of a state machine from its state machine representation.
- List capabilities and limitation, like their uncertainties, of robot systems, including their sensors, and the crucial sensor processing that informs those systems, and in general terms how analog signals can be reasonably represented by discrete samples and articulate strategies for mitigating these uncertainties.
- Identify physical attacks and countermeasures, attacks on non-PC hardware platforms and discuss the concept and importance of trusted path.
- Describe what makes a system a real-time system, explain the presence of and describe the characteristics of latency in real-time systems, and summarize special concerns that real-time systems present, including risk, and how these concerns are addressed.
- Explain the relevance of the terms fault tolerance, reliability, and availability, outline the range of methods for implementing fault tolerance, and explain how a system can continue functioning after a fault occurs.

Skills

- Program a robot to accomplish simple tasks using deliberative, reactive, and/or hybrid control architectures.
- Integrate sensors, actuators, and software into a robot designed to undertake some task.

Competence

- Design and implement an industrial application on a given platform (e.g., using Raspberry Pi).

Course assessment

Assignments – 40 %
Project – 30 %
Final exam - 50 %

Course workload

48 hours lecture,
24 hours lab classes,
36 hours self-study,

18th of October
*Subject to change

30 hours assignment,
30 hours programming project,
12 hours exam preparation and exam.

Reading material

Embedded Systems Design by Peter Marwedel, Springer, 2021.

18th of October
*Subject to change

T-542-HGOP Introduction to Quality Management and Testing

Credits: 6 ECTS
Year: 1st year
Semester: Fall
Level of course: Not defined
Type of course: Elective
Prerequisites: T-220-VLN2, Semester Project 2 and T-303-HUGB, Software Engineering
Structure: 3. week course.
Lecturer: Guðlaugur Stefán Egilsson and Hannes Pétursson

Description

The course will cover methods to ensure the quality of software, both the application code, user interface, delivery process and more. The technologies that were introduced in Software Engineering will be discussed in more detail. We will discuss the various types of tests and automation connections, such as unit testing, automated acceptance testing and automated software delivery.
Property tests will also be discussed briefly.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be able to identify which items have the most impact on software quality.
- Be able to identify the main types of tests.
- Know which items need to be present to deliver the software repeatedly and reliably.

Skills

- Be able to write unit tests that cover the majority of the code system (code coverage). .
- Be able to define and perform other types of tests, such as integration testing, load/performance testing, and manual testing such as usability testing and exploratory testing.
- Be able to set up a "delivery pipe" for web applications.

Competence

- Be able to determine how much work is required to test a system, and determine what kind of testing to focus on.

Course assessment

TBA

Course workload

36h of lectures,
45h exercise classes,
60h projects and reading
12h for exam prep. and
3 hours for the exam.

Reading material

Slides from lecturer.

18th of October
*Subject to change

T-603-THYD Compilers

Credits: 6 ECTS
Year: 3rd year
Semester: Fall
Level of course: Not defined
Type of course: Elective
Prerequisites: T-501-FMAL Programming Languages
Structure: 12. week course.
Lecturer: Yngvi Björnsson

Description

The course defines the function and structure of a compiler. Lexical and syntax analysis is discussed in detail, including use of regular expressions, finite automata, and top-down and bottom-up parsing approaches. Semantic analysis and (intermediate) code generation is also covered in some detail. The course also introduces tools for automatically generating lexers and parsers from formal specifications, both their use and underlying algorithms. Hands-on construction of a compiler/interpreter is a large component of the course.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Understand the structure and design of compilers.
- Understand the role and function of lexical-, syntax- and semantic-analysis, as well as (intermediate) code generation.
- Have the necessary theoretical foundation for constructing a simple compiler.

Skills

- Be able to use regular expressions and finite machines to perform lexical analysis.
- Be able to use formal grammar for describing programming languages and understand how to implement top-down and bottom-up parsing methods.
- Be able to generate (intermediate) code from an abstract-syntax tree, e.g. for virtual machines.
- Be able to use prevalent software tools that automatically generate lexers and parsers from formal specifications.

Competence

- Be able to design and build a simple compiler.

Reading material

Introduction to Compiler Design, Torben Ægidius Mogensen
Slides from lecturer.

Course assessment

Homework (written) – 5%
Labs – 10%
Project (programming a compiler) – 45%
Exams (total) 40%

Course workload

30 hours lectures
5 hours exams
30 hours lecture preparation
15 hours exam preparation
10 hours written homework
15 hours labs
60 hours project (programming an interpreter/compiler)

T-604-HGRE Design and analysis of algorithms

Credits: 6 ECTS
Year: 1st year
Semester: Spring
Level of course: Not defined
Type of course: Mandatory in BSc in Discrete Mathematics and Computer Science
Prerequisites: T-301-REIR, Algorithms and T-419-STR2, Discrete Mathematics II
Structure: 12. week course.
Lecturer: Magnús M. Halldórsson

Description

The course presents leading techniques for developing efficient algorithms (with applications in all areas of computer science and beyond). Developing solutions goes hand in hand with reasoning about their correctness and efficiency. Thus a major objective of the course is to develop skills in reasoning and expressing them verbally and in writing. The course is an excellent preparation for graduate studies.class.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Can describe the workings of the main types of algorithmic strategies, including dynamic programming, greedy alg orithms, network flow, randomized algorithms.
- Have obtained insight into various advanced fields of algorithm theory.

Skills

- Be able to classify algorithm according to complexity, in order to choose between possibilities during design.
- Be able to apply the main algorithmic strategies to the solution of practical problems.
- Be able to analyze the time complexity of algorithms.
- Be able to recognize intractable problems and be able to find workarounds.
- Be able to argue formally the correctness and efficiency of algorithms.

Competence

- Be able to design efficient solution methods to general realistic problems.

Course assessment

Written homework problems – 50%
Exam – 50%

Course workload

36 hours lecture,
18 hours problem sessions,
3 hours exam,
13 hours exam preparation,
72 hours assignments,
24 hours lecture preparation.

Reading material

Kleinberg, Tardos: Algorithm Design. Addison-Wesley.

18th of October
*Subject to change

T-622-UROP Undergraduate Research Opportunity

Credits: 6 ECTS or 12 ECTS
Year: 3rd year
Semester: Fall semester
Level of course: 3. First cycle, advanced
Type of course: Elective
Prerequisites: None
Structure: On-site
Lecturer: N/A

Description

Students receive training in research by working on research projects within the department in close collaboration with teachers. Activities can take various forms, all with the objective of increasing the skills and competences of students in the field of computer science or related fields. Projects can be independent research or development projects, or a part of a larger project.

Learning outcomes

Að námskeiði loknu er gert ráð fyrir að nemandinn:

Knowledge

- Be able to describe a research project and the area it belongs to.
- Be able to explain research and in particular research in computer science.

Skills

- Be able to define and follow a project schedule.
- Be able to follow the necessary steps to complete the goals set out.

Competence

- Be able to present and defend his/her findings in research to an audience.

Assessment

Reading Material

Reading material depends on project

18th of October
*Subject to change

T-624-CGDD Computer Game Design & Development

Credits: 6 ECTS
Year: 1st year
Semester: Fall
Level of course: Not defined
Type of course: Elective
Prerequisites: T-301-REIR, Algorithms
Structure: 3. week course.
Lecturer: Steingerður Lóa Gunnarsdóttir

Description

This course covers the theory and practice of designing and developing computer games, from generating initial concepts to creating a fully playable game.
Computer games are interactive environments that serve a specific goal: some enable player fun, some convey rich emotions, and some change the way that people think about the world. The emphasis of this course will be on team-based collaboration, with each team working to design and develop a game from the start to finish. In support of this process, each team will progress through a structured sequence of challenges during lab time, as guided by the concepts that are discussed and practiced during class.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be able to describe the formal elements of games and the relationships between them.
- Be able to explain some common game AI techniques.
- Be able to describe common forms and structures of narrative in games.
- Be able to discuss insights gained from games industry practitioners.
- Be able to describe some current directions in computer game research.

Skills

- Be able to employ focused strategies to generate ideas for computer games.
- Be able to apply some practical paradigms for game design & development.
- Be able to communicate game ideas clearly and concisely.

Competence

- Be able to navigate intellectual property concerns in game development.
- Be able to design and conduct a play-test to evaluate a game.
- Be able to design and develop a game demo in a limited amount of time.

Reading material

Slides from lecturer.

Course assessment

Group work methods, progress and final demo.

Course workload

Group work and presentations throughout course

18th of October
*Subject to change

T-634-AGDD Advanced Game Design & Development

Credits: 6 ECTS
Year: 2nd and 3rd year
Semester: Spring
Level of course: Not defined
Type of course: Elective
Prerequisites: T-624-CGDD, Computer Game Design & Development
Structure: 12. week course
Lecturer: Steingerður Lóa Gunnarsdóttir

Description

This course expands RU's prior offerings in game design & development with more advanced topics in game design as well as del ving into useful aspects of interaction and experience design. Through lectures, lab exercises, and project work, students will learn and gain experience with a variety of game design topics. Working together in teams, students will design, develop, and critically analyze several smaller ga mes, each focused on applying the concepts that are discussed in class. Each of these exercises will differ in terms of either the team's composition, the game's scope, or the constraints that the instructors pro vide to guide the creation process. oning.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Discuss game design, interaction design, player experience.
- Explain different methods for game design.
- Understand the roles and responsibilities required in a game's production.

Skills

- Critically analyze given game designs and interaction designs.
- Conduct design sessions involving players.
- Develop focused game prototypes.
- Identify uses for game design outside of the games industry.

Competence

- Assess team health and their effect on it.
- Design game mechanics to achieve an intended experience.
- Analyze and evaluate game prototypes.
- Develop a game informed by past prototypes and research.

Reading material

Slides from lecturer.

Course assessment

Projects, analysis reports and participation.

Course workload

39 hours classes,
19 hours lab,
110 hours estimated for project work.

18th of October
*Subject to change

T-631-SOE2 Software Engineering II - Testing

Credits: 6 ECTS
Year: 3rd year BSc in Software Engineering
Semester: Spring
Level of course: Not defined
Type of course: Mandatory
Prerequisites: T-303-HUGB, Software Engineering
Structure: 12. week course
Lecturer: TBA

Description

Various studies show that over than 50% of efforts and costs of software development are devoted to activities related to testing. This includes: test design, execution, and evaluation. This course is an introductory course in software testing. In which, students will learn quantitative, technical, and practical methods and techniques that software engineers use to test their software throughout the software lifecycle.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Understand what is software testing and why we need it.
- Understand the concepts and theory related to software testing.
- Learn the different types of formal coverage criteria.
- Differentiate between different techniques that can be used for software testing and when to apply each of them.
- Understand how software developers can integrate a testing framework into code development in order to incrementally develop and test code.

Skills

- Identify the test requirements.
- Define a model of the software, then find ways to cover it.
- Derive the test plan and evaluate the test suite coverage.
- Learn to use automated testing tools in order to measure code coverage.

Competence

- Design tests based on structures: graph, logic, and input space.
- Define coverage criterion, define the test requirements for each coverage criterion, and derive the test cases that satisfy a coverage criterion.
- Apply the coverage criteria and software testing techniques to uncover defects in a large software system.
- Use open-source testing tools such (e.g., JUnit and NUnit) to test a software system.

Course assessment

Assignments – 20 %
Project – 20 %
Labs – 10 %
Final exam - 50 %

Reading Material

No textbook required. Lecture slides or notes will be provided
Optional Textbook: Introduction to Software Testing 2nd Edition, Cambridge University Press, 2016 l ed.

18th of October
*Subject to change

T-636-SMAT Human Computer Interaction

Credits: 6 ECTS
Year: 2nd/3rd year
Semester: Spring
Level of course: 1.cycle, introduction
Type of course: Selection
Prerequisites: Software requirements and Design or Interactive Design
Structure: 12. week course (not teached yearly)
Lecturer: Marta Kristín Lárusdóttir

Description

The learning material suits students that want to learn about various fields of human-computer interaction, especially focusing on non-mouse interaction. Students study the characteristics of non-mouse interaction types, for example: speech, gestures, tangible interaction and brain computer interaction. Students choose one interaction type and design and evalute a prototype for a software system using that interaction type. The Google Design Sprint process is used while designing and evaluating the prototype. Additionally, students will learn about research in the field and the future trends.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Be familiar with the characteristics of various ways of communication (ex. interaction types) for software.
- Be familiar with the characteristics of various interaction types of software, such as virtual reality, wearable computing, ambient, ubiquitous and mobile computing.
- Be familiar with the research on non-mouse interaction.
- Be familiar with concepts and principles regarding human-computer interaction.

Skills

- Be familiar with the advantages and limitations of various ways of communication (ex. Interactions types).
- Be able to argue for when it is good to apply a particular interaction type.
- Be able to describe the vision for a software project and explain it.

Competence

- Be able to design an innovative computer system interface with user participation.
- Be able to evaluate design examples with users.

Course assessment

48% of the final grade - Individual assignments
22% of the final grade - Group assignments
30% of the final grade – Written exam
To complete the course students have to:
Get 4.75 or above in the written exam

Course workload

48 hours – mixture of lectures and problem solving sessions
36-48 hours – homework on assignments

Reading Material

Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days by Jake Knapp, John Zeratsky, and Braden Kowitz, Simon and Schuster, 2016.

18th of October
*Subject to change

T-637-GEDE Game Engine Architecture

Credits: 6 ECTS
Year: 2nd or 3rd year
Semester: spring
Type of course: Elective course
Necessary Prerequisites: T-211-LINA Linear Algebra, T-511-TGRA, Computer Graphics.
Organization of course: twelve-week course
Teacher: Hannes Högni Vilhjámsson.
Language of teaching: English

Description:

The course covers the theory and practice of game engine software development, bringing together topics that range from large-scale software architectures and modern game programming paradigms to the design and implementation of subsystems for memory management, interface devices, resource management, rendering, collision, physics and animation. Through practical lab exercises and group projects, the students get technical hands-on experience in C++ game development, including the use and development of supporting tool pipelines. The course includes visiting talks and Q&A from industry veterans.

Learning outcomes:

After completion of the course the student will hold a knowledge, skills and competence of:

Knowledge:

- Be able to explain game engines and their role in game development.
- Be able to sketch the typical components of a run-time game architecture.
- Be able to explain programming paradigms and data structures that are commonly used in game development
- Be able to understand what goes on in the rendering pipeline.
- Be able to explain engine sub-systems that deal with start-up/shut-down, memory management, engine configuration, file system, game resources, game loop, rendering loop and interface devices

Skills:

- Be able to explain game engines and their role in game development.
- Be able to use and extend a C++ graphics engine to develop tech demos.
- Be able to use industry standard C++ development and version control tools.
- Be able to apply 3D math, covering points, vectors and matrices, for solving game world problems. Be able to import resources from Digital Content Creation tools.
- Be able to read input from game interface devices.
- Be able to program a basic vertex and fragment shader. Be able to use a particle system to create visual effects.
- Be able to use a physics library for realistic object behavior.

Competence:

- Be able to analyze and compare existing game engines with respect to game development goals and system requirements.
- Be able to research, design, implement and present a tech demo of a low-level engine feature.
- Be able to design new game engines or engine sub-systems, based on established practices and an insight into various architectural decisions (pros and cons).

Assessment:

Participation 5%
Labs 8%
Problem sets 12%
Engine Presentation 10%
Final Project 35%
Final Written Exam 30%
Total 100%

Workload:

36 hours attending lectures
20 hours lecture preparation and study
14 hours lab work

18th of October
*Subject to change

16 hours problem set work
40 hour project work
24 hours final exam preparation.

Reading Material:

Game Engine Architecture by Jason Gregory, CRC Press third ed. (2018).

I-707-VGBI Business Intelligence

Credits: 6 ECTS
Year: 3. Year
Semester: Spring semester
Level of course: 2. Fyrst cycle, intermediate
Type of course: Mandatory in BSc in Computer Science – minor in business
Prerequisites: T-111-PROG, Programming
Structure: 12. week course
Lecturer: Hinrik Jósafat Atlason

Description

The business environment is constantly changing, and it is becoming more and more complex. Organizations, both private and public, are under pressures that force them to respond quickly to changing conditions and to be innovative in the way they operate. Such activities require organizations to be agile and to make frequent and quick strategic, tactical, and operational decisions, some of which are very complex. Making such decisions may require considerable amounts of relevant data, information, and knowledge. Processing these, in the framework of the needed decisions, must be done quickly, frequently in real time, and usually requires some computerized support. This course is about using business analytics as computerized support for managerial decision making. It concentrates on the theoretical and conceptual foundations of decision support, as well as on the commercial tools and techniques that are available. It presents the fundamentals of the techniques and the way these systems are constructed and used.

Learning outcomes

On completion of the course, students should be able to:

Knowledge

- Knows the concept Business Intelligence and can discuss it from theoretical, technical and practical perspectives.
- Understands the structure of Business Intelligence solutions, the importance of Business Intelligence in the business world and the purpose of the different tools used in Business Intelligence.

Skills

- Can design a data model, digital dashboards and make decisions regarding the presentation of data..
- Can work with Microsoft Azure cloud services to create and manipulate data.
- Can work with Microsoft Azure Data Mining Studio to create data mining models..
- Can work with Microsoft PowerBI to create dashboards.

Competence

- Can define a Business Intelligence Competency Center and form a corporate Business Intelligence strategy.

Course assessment

Group projects - 75%
Status exams – 25%

Course workload

36h of lectures, 24h exercise classes, 90 - 120h projects and reading

Reading Material

Business Intelligence: A Managerial Approach, Global Edition