CS4381: ADVANCED SOFTWARE DESIGN

Effective Term
Semester A 2025/26

Part I Course Overview

Course Title
Advanced Software Design

Subject Code
CS - Computer Science
Course Number
4381

Academic Unit
Computer Science (CS)

College/School
College of Computing (CC)

Course Duration
One Semester

Credit Units
3

Level
B1, B2, B3, B4 - Bachelor's Degree

Medium of Instruction
English

Medium of Assessment
English

Prerequisites
CS3342 Software Design

Precursors
MA2185 Discrete Mathematics

Equivalent Courses
Nil

Exclusive Courses
Nil

Part II Course Details

Abstract

This course aims to introduce the advanced techniques for the design of software applications. Students will develop their technical competence in modelling and designing sequential and concurrent software to satisfy software requirements of design solutions from multiple perspectives.

Course Intended Learning Outcomes (CILOs)

	CILOs	Weighting (if app.)	DEC-A1	DEC-A2	DEC-A3
1	Explore the challenges in developing dependable software.		x		
2	Create software models by using an array of semi-informal and formal tools and from multiple perspectives.				
3	Develop the ability to reason software models.			x	

A1: Attitude
Develop an attitude of discovery/innovation/creativity, as demonstrated by students possessing a strong sense of curiosity, asking questions actively, challenging assumptions or engaging in inquiry together with teachers.

A2: Ability
Develop the ability/skill needed to discover/innovate/create, as demonstrated by students possessing critical thinking skills to assess ideas, acquiring research skills, synthesizing knowledge across disciplines or applying academic knowledge to real-life problems.

A3: Accomplishments
Demonstrate accomplishment of discovery/innovation/creativity through producing /constructing creative works/new artefacts, effective solutions to real-life problems or new processes.

Learning and Teaching Activities (LTAs)

	LTAs	Brief Description	CILO No.	Hours/week (if applicable)
1	Lecture and tutorial	Students will engage in lectures that explain key concepts such as theories and formal models of software applications.	1, 2, 3	Lecture: 3 hours/week Tutorial: 8 hours/semester

2	Coursework	Students will model design scenarios by different kinds of semi-informal and formal languages to address the same or similar software requirements. The students are also required to generalize the design solutions so that the solutions can cope with wider classes of scenarios of the same or similar nature. Apply both informal and formal techniques to walk through the design solutions, or model a formal idea into an informal notation, and vice versa.	2, 3	
3	Project	Students will take on the role of software model developers to create a model using advanced design technique. Conduct a survey on case studies about software design to compare and contrast how different design solutions may solve the same or similar technical problems, as well as make critiques on how to make design decision based on their merits and limitations.	1, 2, 3	

Assessment Tasks / Activities (ATs)

	ATs	CILO No.	Weighting (%)	Remarks ("-" for nil entry)	Allow Use of GenAI?
1	Coursework	2, 3	25	-	Yes
2	Project	1, 2, 3	25	-	Yes

Continuous Assessment (%)

50

Examination (%)

50

Examination Duration (Hours)

2

Minimum Examination Passing Requirement (%)

30

Additional Information for ATs
For a student to pass the course, at least 30% of the maximum mark for the examination must be obtained.

Assessment Rubrics (AR)

Assessment Task
Coursework

Criterion
1.1 Ability to explain the methodology and procedure to create software model

Excellent (A+, A, A-)
High

Good (B+, B, B-)
Significant

Fair (C+, C, C-)
Moderate

Marginal (D)
Basic

Failure (F)
Not even reaching marginal levels

Assessment Task
Coursework

Criterion
1.2 Ability to reason the behaviour of software models

Excellent (A+, A, A-)
High

Good (B+, B, B-)
Significant

Fair (C+, C, C-)
Moderate

Marginal (D)
Basic

Failure (F)
Not even reaching marginal levels

Assessment Task
Project

Criterion

2.1 Ability to self-directed creation of a software model with behavioural analysis. Capacity for self-directed learning to compare and contrast software models.

Excellent (A+, A, A-)
High

Good (B+, B, B-)
Significant

Fair (C+, C, C-)
Moderate

Marginal (D)
Basic

Failure (F)
Not even reaching marginal levels

Assessment Task
Examination

Criterion
3.1 Ability to explain the methodology and procedure to create software model

Excellent (A+, A, A-)
High

Good (B+, B, B-)
Significant

Fair (C+, C, C-)
Moderate

Marginal (D)
Basic

Failure (F)
Not even reaching marginal levels

Assessment Task
Examination

Criterion
3.2 Ability to reason the behaviour of software models

Excellent (A+, A, A-)
High

Good (B+, B, B-)
Significant

Fair (C+, C, C-)

Moderate

Marginal (D)
Basic

Failure (F)
Not even reaching marginal levels

Assessment Task
Examination

Criterion
3.3 Ability to create software models with behavioural analysis.

Excellent (A+, A, A-)
High

Good (B+, B, B-)
Significant

Fair (C+, C, C-)
Moderate

Marginal (D)
Basic

Failure (F)
Not even reaching marginal levels

Part III Other Information

Keyword Syllabus
Software non-functional requirements, state machine diagram, message sequence chart, concurrency, process algebra, refinement, advanced design patterns, architectural patterns.

Syllabus

- Software non-functional specification
  Attribute-driven design, architecture design and analysis, non-functional requirements
- Semi-formal software modelling
  Advanced design patterns for concurrency and resources management, architectural patterns, quality attribute, design tactics
- Formal software modelling
  Process algebra, statecharts, pre-/post-condition, assertion
- Reasoning and development
  Usage scenarios, model refinement

Reading List

Compulsory Readings

	Title
1	Nil

Additional Readings

	Title
1	Schmidt, D., Stal, M., Rohnert, H., and Buschmann, F. (2004). Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. Wiley series in software design patterns. Wiley.
2	Kircher, M. and Jain, P. (2004). Pattern-Oriented Software Architecture, Volume 3, Patterns for Resource Management. Wiley series in software design patterns. Wiley.
3	Buschmann, F., Henney, K., and Schmidt, D. (2007). Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, Volume 4, A Pattern Language for Distributed Computing. Wiley series in software design patterns. Wiley.
4	Magee, J., and Kramer, J. (2006). Concurrency: State Model and Java Programs. Wiley.
5	Zeller, A. (2009). Why Programs Fail: A Guide to Systematic Debugging. Morgan Kaufmann; 2nd edition.
6	Pierce, B.C. (2002). Types and Programming Languages. MIT Press.
7	Harper, R. (2012). Practical Foundations for Programming Languages. Cambridge University Press.