Formal Methods and Software Development: A Comprehensive Lecture Curriculum

Course Title: Formal Methods and Software Development

Course Code: CMP 415

Target Audience: 4th Year Computer Science Students

Credit Hours: 3 **Prerequisites:** Introduction to Programming, Data Structures and Algorithms, Discrete Mathematics, Introduction to Software Engineering

Course Description: This course introduces students to the principles and practices of formal methods and their application in improving the reliability, correctness, and safety of software systems. Students will learn to use mathematical techniques to specify, design, and verify software, moving beyond informal testing and debugging. The course emphasises a theoretical understanding coupled with practical application through case studies and hands-on exercises.

Course Objectives: Upon successful completion of this course, students will be able to:

1. **Understand the fundamental concepts** of formal methods, including specification, modelling, verification, and validation.
2. **Appreciate the benefits and limitations** of using formal methods in software development.
3. **Apply mathematical formalisms** like set theory, logic, and state machines to model software components and their behaviour.
4. **Write formal specifications** for software systems using various notations (e.g., Z, VDM, B-Method, temporal logic).
5. **Understand and apply formal verification techniques**, including model checking and theorem proving, at an introductory level.
6. **Recognise the role of formal methods** in critical software development domains (e.g., safety-critical systems, security).
7. **Evaluate the suitability of different formal methods** for specific software development problems.
8. **Gain practical experience** with at least one formal method tool.

Lecture Curriculum Outline

This curriculum is structured into modules, each covering a distinct aspect of formal methods and their application in software development. Each module is designed to be approximately one week of lectures (2-3 sessions).

Module 1: Introduction to Formal Methods and Software Engineering (Week 1)

- **Lecture 1.1: The Need for Rigour in Software Development.**
    - Challenges in traditional software development: complexity, bugs, and cost of errors.
    - Examples of software failures and their impact (e.g., Ariane 5, Therac-25, Mars Pathfinder).
    - The role of testing and its limitations (halting problem, combinatorial explosion, inability to prove absence of errors).
    - Introduction to formal methods: what they are, their goals (correctness, reliability, safety).
- **Lecture 1.2: Foundations of Formal Methods.**
    - Key concepts: Specification, modelling, verification, validation.
    - Types of formal methods (model-based vs. property-based).
    - Mathematical underpinnings: Brief review of set theory, relations, functions, and basic logic.
    - Overview of the formal methods lifecycle.

Module 2: Formal Specification Techniques (Weeks 2-4)

- **Lecture 2.1: Data Types and State-Based Modelling.**
    - Abstract Data Types (ADTs): defining data structures and their operations.
    - Introduction to set theory for modelling data.
    - Basic algebraic specifications (brief mention, focus on state-based).
- **Lecture 2.2: Introduction to Z Notation.**
    - The Z schema notation: declaration, predicate parts.
    - Basic Z types: sets, sequences, relations, functions.
    - Basic operations: initialisation, state transition.

    - Examples: Simple data structures (e.g., stack, queue).
- **Lecture 2.3: Advanced Z Notation and Modelling.**
    - Schema composition and refinement.
    - Generic schemas.
    - Modelling more complex systems with Z (e.g., library system, simple database).
- **Lecture 2.4: Introduction to VDM (Vienna Development Method).**
    - VDM-SL (VDM Specification Language).
    - Data type definitions (implicit and explicit).
    - Operation specifications (pre- and post-conditions).
    - Comparison with Z.
- **Lecture 2.5: Introduction to the B-Method.**
    - The B-Method: a refinement calculus.
    - Basic B constructs: abstract machines, states, operations, visible and internal states.
    - Refinement: moving from abstract to concrete specifications.
    - Advantages of B for tool support.

Module 3: Formal Verification Techniques (Weeks 5-7)

- **Lecture 3.1: Introduction to Formal Verification.**
    - Why verify? Proving correctness.
    - Verification vs. Testing.
    - Approaches to verification: Model Checking vs. Theorem Proving.
- **Lecture 3.2: Model Checking: Concepts and Principles.**
    - What is model checking?
    - State-space exploration.
    - Formalisms for representing systems and properties (e.g., finite state machines, Kripke structures).
    - Temporal Logic: Introduction (linear-time vs. branching-time – CTL and LTL).

- **Lecture 3.3: Model Checking: Properties and Algorithms.**
    - Common temporal operators: G (Globally), F (Finally), X (Next), U (Until).
    - Expressing safety and liveness properties.
    - Basic algorithms for model checking (e.g., Büchi automata, state-space traversal).
- **Lecture 3.4: Theorem Proving: Concepts and Principles.**
    - What is theorem proving?
    - Axiomatic systems and inference rules.
    - Proof assistants and interactive theorem provers.
    - The concept of formal proofs.
- **Lecture 3.5: Theorem Proving: Techniques and Challenges.**
    - Induction, recursion, and other proof techniques.
    - Automated vs. Interactive theorem proving.
    - Challenges: complexity of proofs, expertise required.

Module 4: Formal Methods in Practice and Case Studies (Weeks 8-9)

- **Lecture 4.1: Tools and Technologies for Formal Methods.**
    - Overview of popular tools: Z/EVES, VDMTools, Atelier B, UPPAAL (for model checking), Isabelle/HOL, Coq.
    - Demonstration of a chosen tool (e.g., UPPAAL for model checking of a simple system, or Atelier B for state-based modelling).
- **Lecture 4.2: Case Study 1: Safety-Critical Systems.**
    - Application of formal methods in aerospace, railway, or medical devices.
    - Example: Specification and verification of a simple control system (e.g., traffic light controller, simple elevator control).
- **Lecture 4.3: Case Study 2: Security and Cryptography.**
    - Formal analysis of security protocols.
    - Example: Specification and analysis of a simple authentication protocol.
- **Lecture 4.4: Integrating Formal Methods into the Software Development Lifecycle.**

- When and where to use formal methods.
- Cost-benefit analysis.
- Challenges in adoption and training.

Module 5: Advanced Topics and Future Directions (Week 10-11)

- **Lecture 5.1: Formal Methods for Object-Oriented Systems.**
  - Modelling classes and their interactions formally.
  - Extensions of formalisms for OO concepts (e.g., OCL).
- **Lecture 5.2: Probabilistic Model Checking.**
  - Dealing with uncertainty and randomness in systems.
  - Applications in performance analysis and fault tolerance.
- **Lecture 5.3: Refinement Calculus and Program Derivation.**
  - Deriving programs from formal specifications.
  - The role of refinement in ensuring correctness.
- **Lecture 5.4: The Future of Formal Methods.**
  - Trends in research and industry.
  - AI and formal methods.
  - Challenges and opportunities.

Module 6: Course Review and Project Work Preparation (Week 12)

- **Lecture 6.1: Comprehensive Review of Key Concepts.**
  - Recap of specification languages, verification techniques, and case studies.
  - Q&A session to address student queries.
- **Lecture 6.2: Introduction to Course Project and Assessment.**
  - Explanation of the final project requirements.
  - Guidance on choosing a topic and methodology.
  - Discussion of assessment criteria.

Assessment Strategy

- **Assignments (30%):**
    - Problem sets focusing on applying formal specification notations (Z, VDM, B).
    - Exercises on expressing properties in temporal logic.
    - Small exercises using a selected formal methods tool.
- **Midterm Examination (20%):**
    - Covers material from Modules 1-3, focusing on understanding of formalisms and basic techniques.
- **Course Project (30%):**
    - Students will work in groups (or individually) on a project that involves:
        - Selecting a small software system or component.
        - Formally specifying it using one of the introduced notations.
        - Verifying a critical property using an appropriate technique or tool (e.g., model checking a temporal logic property, attempting a proof).
        - Writing a report documenting their approach and findings.
- **Final Examination (20%):**
    - Comprehensive, covering all course material with a focus on synthesis, application, and critical evaluation of formal methods.

Recommended Textbook(s)

- **Primary:**
    - "Formal Methods: Foundations and Applications" by Paul Curzon and Janet E. Knight (or a similar introductory textbook)
- **Supplementary:**
    - "The Z Notation: A Reference Manual" by J. M. Spivey
    - "Software Engineering with Z" by K. L. Cameron and J. P. J. Z. Al-Khamis
    - "Model Checking" by Edmund M. Clarke, Orna Grumberg, and David E. Long (for advanced model checking)
    - Online documentation and tutorials for the chosen formal methods tools.

Teaching Methodology

- **Lectures:** Traditional lectures to introduce concepts, theories, and techniques.
- **Tutorials/Problem Sessions:** Dedicated sessions for working through examples, solving problems, and discussing assignment solutions.
- **Lab Sessions (Optional but recommended):** Hands-on sessions with formal methods tools to gain practical experience.
- **Guest Lectures (Optional):** Invite industry professionals or researchers to share their experiences with formal methods.
- **Student Presentations:** Students might present their project progress or findings.

Software and Tools

- **Specification Tools:**
    - Z/EVES (for Z notation)
    - VDMTools (for VDM)
    - Atelier B (a popular choice for B-Method)
- **Model Checking Tools:**
    - UPPAAL (for real-time systems, state-based models)
    - NuSMV (a popular explicit-state model checker)
- **Theorem Provers (for advanced exploration):**
    - Isabelle/HOL
    - Coq

(The specific tool(s) to be used will depend on availability, instructor expertise, and practical considerations for undergraduate students.)

Academic Integrity

Students are expected to uphold the highest standards of academic integrity. All submitted work must be their own, or properly attributed when collaboration is permitted. Plagiarism and cheating will not be tolerated and will be dealt with according to university policy.

This curriculum provides a comprehensive framework for teaching Formal Methods and Software Development at Anchor University Lagos. The emphasis on both theoretical understanding and practical application through tool usage and case studies will equip students with valuable skills for building robust and reliable software systems. The specific depth of coverage for each topic can be adjusted based on the time available and the students' prior knowledge.