Department of Computer Science, SPPU		
Course: M.Sc. (Computer Science)		
Duration: 02 Years		
Total No. of credits for the Course:		
Semester	**Core Course Credits**	**Elective Course Credits**
1	**20**	**0**
2	**20**	**0**
3	**8/12/16**	**12/8/4**
4	**12/8/4**	**8/12/16**
Total Credits (80)	**60**	**20**
M.Sc. (Computer Science) Semester I		
Subject Code	**Subject Title**	**Number of Credits**
CS-101	**Programming – 1**	**5**
CS-102	**Systems- 1**	**4**
CS-103	**Mathematics of Computation– 1**	**4**
CS-104	**Mathematics of Computation- 2**	**4**
CS-105	**Database Management System**	**3**
UGC recommended	**Skill Development**	**2**
UGC recommended	**HRE**	**1**
Total Credits		**20+3**
M.Sc. (Computer Science)Semester II		
Core Courses		
Subject Code	**Subject Title**	**Number of Credits**
CS-201	**Programming – 2**	**5**
CS-202	**Programming – 3**	**4**
CS-203	**Systems-2**	**4**
CS-204	**Systems-3**	**4**
CS-205	**Theory of Computation**	**3**
UGC recommended	**Skill Development**	**2**
UGC recommended	**HRE**	**1**
Total Credits		**20+3**

Department of Computer Science, SPPU		
Course: M.Sc. (Computer Science)		
M.Sc. (Computer Science) Semester III		
Core Courses		
Subject Code	**Subject Title**	**Number of Credits**
CS-301	**Systems-4**	**4**
CS-302	**Formal Methods-1**	**4**
CS-303	**Elective-1**	**4**
****CS-MCP/CS-304**	**Degree Project-1/Elective-2**	**4**
****CS-MCP/CS-305**	**Degree Project-1/Elective-3**	**4**
UGC recommended	**Cyber Security - 1 and 2**	**2**
Total Credits		**20+2**
****The candidate is permitted to choose either 0/1/2 elective courses or can work on degree project for 0/4/8 credits. The total number of credits for degree project are 8 (Eight) and the same are distributed across the second year of degree program.**		
M.Sc. (Computer Science) Semester IV		
Core Courses		
Subject Code	**Subject Title**	**Number of Credits**
CS-401	**Formal Methods - 2**	**4**
****CS-MCP/CS-402**	**Degree Project-2/Elective-4**	**4**
****CS-MCP/CS-403**	**Degree Project-2/Elective-5**	**4**
CS-404	**Elective-6**	**4**
CS-405	**Elective-7**	**4**
UGC recommended	**Cyber Security - 3 and 4**	**2**
Total Credits		**20+2**

Course contents in detail

CS-301 Systems - 4

Objective: This is the fourth course in the collection on systems. At the end of the course a successful student should be able to appreciate the issues involved in developing a software subsystem in which the primary activity is fundamentally concurrent and not only data but computations can be divided across multiple possibly heterogeneous computing devices.

- Overview of concurrency control – Serializable scheduling, correctness of schedules, generation of efficient and correct schedules; atomicity, consistency, isolability and durability (ACID) properties; lock-based concurrency control, transactions as a concurrency control mechanism.
- Concurrent programing - concurrency as a fundamental aspect of programming high performance systems, concurrent programing is not just parallel programing, concurrent programing is not just distributed programing.
- Composable concurrency – impossibility of composability of lock based concurrent programs, introduction to software transactional memory (STM), STM based composition.
- Distributed computations – concurrent systems with lack of global clock, Message passing model, consensus problem, mutual exclusion in distributed systems, problem of atomic commit, 2-phase commit, 3-phase commit.

References:

- Principles of concurrent and distributed programming - Mordechai Ben-Ari
- Concurrent programming: principles and practice - Gregory R Andrews
- On Concurrent Programming - Fred B. Schneider
- Concurrent Programming: Algorithms, Principles, and Foundations - Michel Raynal
- Principles of Transactional Memory (Synthesis Lectures on Distributed Computing Theory) - Rachid Guerraoui, Michal Kapalka
- Control Flow and Data Flow: Concepts of Distributed Programming: International Summer School - E. W. Dijkstra et el
- A Discipline of Multiprogramming: Programming Theory for Distributed Applications (Monographs in Computer Science) - Jayadev Misra

Note: the topic “Overview of concurrency control”, should be covered in brief as an overview of topics learnt in the earlier year.

Suggested further reading:

- Functional Programming for Loosely-Coupled Multiprocessors (Research Monographs in Parallel and Distributed Computing) - Paul H. J. Kelly
- Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming - Simon Marlow
- Erlang Programming : a Concurrent Approach to Software Development - Simon Thompson
- Programming ERLANG: Software for a Concurrent World - Joe Armstrong
- Communicating sequential processes - C.A.R. Hoare

CS-302 Formal Methods - 1

Objective: This is the first course in the collection on formal methods. At the end of the course a successful student should be able to appreciate the basics of formalisms that play a crucial role in the verification and validation of software programs.

- Verification : verification of imperative programs as in Gries/Dijkstra.
- Specific techniques: Invariant assertive method, sub-goal induction method.
- Verification of pointer programs.
- Function Program verification: Induction on data-types, infinite data structure induction
- Homomorphic transformations, refinement Calculus Theory & application of List/Functional calculus
- Hoare Logic - correctness of computer programs, Hoare triple, Partial and total correctness, weakest precondition and strongest postcondition,
- Advanced topics– temporal logic, formal analysis of reactive systems using temporal logic.

References:

- Discipline of Programming, Dijkstra, Prentice Hall
- Method of Programming, Dijkstra & Feijen, Addison Wesley
- Logic of Programming and Calculi of Discrete Design: International Summer School - Richard S. Bird et el
- Specification Case Studies, Hayes, Prentice Hall
- Software Specification, Gehani & Mcgettrick, Addison Wesley
- Program Specifications & Transformations, Meertens, Prentice Hall
- Partial Evaluation and Mixed Computation, Ershov, Bjorner & Jones, North Holland.
- Programs from Specifications, Morgan, Prentice Hall
- Lectures of constructive functional programming, Bird, Lecture notes, PRG Oxford
- Introduction to the theory of lists, Bird, Lecture notes, PRG Oxford
- A calculus of functions for program derivation, Bird, Lecture notes, PRG Oxford

- Introduction to Formal Program verification, Mili, Van Nostrand Reinhold
- Logic in Computer Science: Modelling and Reasoning about Systems, by Michael Huth , Mark Ryan Cambridge Univeristy Press

Suggested further reading:

- First-Order Programming Theories (Monographs on Theoretical Computer Science) - Tamás Gergely, László Úry
- The Temporal Logic of Reactive and Concurrent Systems: Specification - Z. Manna and A. Pnueli
- Temporal Verification of Reactive Systems: Safety - Z. Manna and A. Pnueli
- Introduction to Mathematics of Satisfiability – V. W. Marek
- Value-Range Analysis of C Programs: Towards Proving the Absence of Buffer Overflow Vulnerabilities - A. Simon

CS-401 Formal Methods – 2

Objective: This is the second course in the collection on formal methods. At the end of the course a successful student should be able to apply formal techniques of program verification/validation to large scale software applications, and also appreciate the underlying theoretical basis.

- Software verification – Rice's theorem and the problem of undecidability of program verification; notion of non-trivial properties of partial functions, tractability issues in program verification.
- Introduction to program semantics - denotational semantics, axiomatic semantics, operational semantics.
- Static program analysis - Abstract interpretation, Symbolic execution, Data-flow analysis, Hoare logic, model checking.
- Abstract interpretation – sound approximations of semantics of programs, monotonic functions over ordered sets; abstraction function, concretization function, and the concept of valid abstraction;
- Symbolic execution – symbolic interpretation and constraint solving, problem of path explosion, problem of side effects, handling side effects with environment modeling and/or by creating virtual machines.
- Data-flow analysis – control flow graphs and fixpoints, the problem of reaching definition, blocks and computing entry and exit states of blocks; forward and backward analysis and convergence issues; liveness analysis as a subproblem in data-flow analysis.
- Model checking – state explosion problem, Symbolic model checking, binary decision diagrams.
- Alternative approaches to software verification - interactive theorem provers, automatic theorem provers, or satisfiability modulo theories (SMT) solvers.
- Concurrency and verification: Models of concurrency, The parallel random-access machine model, the actor model, bulk synchronous parallel (BSP) model, Petri nets model, Process calculi models (Communicating sequential processes (CSP) model)

Note: Teacher may introduce a suitable tool or two (e.g. by taking into consideration the teaching-learning activities happening in the Software Development 1 and Software Development 2 courses, or to complement the same) e.g. from the following non-exhaustive list: [Alloy analyzer, BLAST, FDR, Prism etc]

References:

- Semantics with Applications: An Appetizer - Flemming Nielson et el

- Path-Oriented Program Analysis - J. C. Huang
- Generation of Program Analysis Tools [PhD Thesis] (University of Amsterdam) - Frank Tip
- The Calculus of Computation: Decision Procedures with Applications to Verification - Z. Manna and A. Bradley

Suggested further reading:

- Principles of Program Analysis - Flemming Nielson et el
- Trustworthy compilers - Vladimir O. Safonov
- Introduction to Mathematics of Satisfiability – V. W. Marek
- Value-Range Analysis of C Programs: Towards Proving the Absence of Buffer Overflow Vulnerabilities - A. Simon
- Formal Methods: State of the Art and New Directions – Dines Bjørner

CS-MCP Degree project

Objective: To expose the students to the process of working on a post-graduate level research involving application of fundamental principles of computer science and programming to solve a non-trivial problem.

Structure:

Each student shall work on a large scale project preferably involving core programming/systems-programming and should defend the work before audience.