DEPARTMENT OF COMPUTER SCIENCE
SAVITRIBAI PHULE PUNE UNIVERSITY

Syllabus for
2 Years Master of Computer Science (M.Sc.)
(Under NEP guidelines)

Wef Academic Year 2023 - 2024

Department of Computer Science,
Savitribai Phule Pune University,
Pune-411 007.

Department: Department of Computer Science, SPPU
Course: MSc. (Master of Computer Science)
Duration: 2 Years
Total Number of Credits: 88
The degree programme offered in the Department of Computer Science, Savitribai Phule Pune University, Pune, is a full-time course. The policies and procedures determined by the University will be followed for the conduct of examinations and declaration of the result of a candidate.

1. **Preamble of the syllabus:** This program is offered at the Department of Computer Science, Savitribai Phule Pune University, Pune. Master of Computer Science (M.Sc.) program is of 88 credits. The objective of the M.Sc. program is to train the students to meet the challenges of the Software Industry and R&D Sector with computational techniques.
2. **Program Structure of M.Sc. (Computer Science):** For M.Sc. (Computer Science) Degree, a student has to earn the minimum 88 credits from at least FOUR semesters. The structure of the program is as follows.
   a. In each of the four semesters I, II, III, and IV:
      i. the Department will offer at least 22 credits.
      ii. there will be three mandatory courses each of 4 credits, and one elective course. There will also be a mandatory course of 2 credits.
      iii. additionally, there will be compulsory courses in various semesters: Research Methodology (RM) - 4 credits, On Job Training (OJT) - 4 credits, Research Project (RP1) - 4 credits, Research Project (RP2)- 6 credits.
   b. A student cannot register for the third semester, if he/she fails to complete 50% credits of the total credits expected to be ordinarily completed within two semesters. In this case, a student can seek admission to first or second semester in order to complete the requisite number of credits and to be able to seek admission in the third semester.
   c. A student will obtain non-zero credits only on obtaining a pass grade in a course.
   d. For all courses, interaction and tutorial sessions will be scheduled every semester batch wise with a batch size of 12.
   e. The modus-operandi for the conduct and evaluation of a Research Project Course will be decided by the Departmental Committee from time to time as per the needs.
   f. The Departmental Committee in its meeting with the majority may introduce/design additional course(s) and include/exclude/modify the existing course(s) to accommodate the then developments from time to time.

3. **Evaluation Rules:**
   a. 50% of marks as semester-end examination of minimum 30 minutes to maximum 45 minutes per credit and
   b. 50% marks for internal (i.e. in-semester) assessment.
   c. Each credit will have an internal (continuous) assessment of 50% of marks and a teacher must select a variety of procedures for examination such as:
      i. Written Test and/or Mid Term Test (not more than one for each course);
      ii. Term Paper;
      iii. Journal/Lecture/Library notes;
      iv. Seminar presentation;
      v. Short Quizzes;
      vi. Assignments;
      vii. Extension Work;

viii. Research Project by individual students or group of students; or
ix. An Open Book Test (with the concerned teacher deciding what books are to be allowed for this purpose.)

d. To pass a course, the student has to obtain forty percent marks in the combined examination of in-semester assessment and semester-end assessment with a minimum of thirty percent in both these separately.
e. **Research Project I (CS-631 RP)** is in the semester 3 and 4 credits are reserved for it.
f. **Research Project II (CS-681 RP)** is in the semester 4 and 6 credits are reserved for it.
g. The student is expected to work on a project assigned by the guide.

4. **Completion of Degree Program:**
   a. In order to pass the Master of Computer Science (M.Sc.) course a student has to obtain 88 credit points and complete the audit courses floated by the University time to time.
   b. If a student fails in a course then the said course will not be considered for calculating CGPA and overall grade. Only those courses in which the student has passed will be considered for calculating the CGPA and overall grade.
   c. The policies and procedures determined by the University will be followed for the conduct of examinations and declaration of the result of the candidate.

Course Structure

M.Sc. (Computer Science)								
Duration: 02 Years								
Level	Semester	Subject Code	Subject Title	Core	Elective	RM	RP	FP
6	I	CS-501 MJ	Programming – I	4				
		CS-502 MJ	Systems – I	4				
		CS-503 MJ	Mathematics of Computation – I	4				
		CS-504 MJ	Mathematics of Computation – II (Part – 1)	2				
		CS-510* MJ	Elective1		4			
		CS-531 RM	Research Methodology			4		
		Total Credits		22				
	II	CS-551 MJ	Programming – II	4				
		CS-552 MJ	Mathematics to Data Management	4				
		CS-553 MJ	Foundations of Computing	4				
		CS-554 MJ	Mathematics of Computation – II (Part – 2)	2				
		CS-560* MJ	Electives		4			
		CS-581 OJT	FP/OJT					4
		Total Credits		22				
6.5	III	CS-601 MJ	Programming – III	4				
		CS-602 MJ	Systems – II	4				
		CS-603 MJ	Systems – III	4				
		CS-604 MJ	Formal Methods – I	2				
		CS-610* MJ	Elective		4			
		CS-631 RP	Research Project I				4	
		Total Credits		22				
	IV	CS-651 MJ	Formal Methods – II	4				
		CS-652 MJ	Software Comprehension	4				
		CS-653 MJ	Systems - IV	4				
		CS-660* MJ	Elective		4			
		CS-681 RP	Research Project II				6	
		Total Credits		22				
*- Electives in Ist Semester will be numbered from CS-510 onwards, in IInd semester, from CS-560 onwards, in IIIrd Semester from CS-610 onwards and in IVth semester from CS-660 onwards.								

Semester I

CS-501 MJ Programming – I (4 Credits)

Objective: This course is the first in a sequence of three courses on programming. A successful student would have learnt to write code for non-trivial problems by learning the appropriate abstractions and paradigm, namely functional. A subsequent step is the transformation of these into the imperative form.

- Standard Constructs: Function and type definition, block structure, Guarded equations, pattern matching, Special syntax for lists, comprehension.
- Standard Data Types: Fluency is to be achieved in the standard data types: numbers, Boolean, character, tuple, list, List programs in an algebraic vein, Lists in the context of general collections sets, bags, lists, and tuples.
- Calculus: A direct way for denoting functions.
- First-Class-ness: All values are uniformly treated and conceptualized.
- Higher Order Functions: Use of first class, higher order functions to capture large classes of computations in a simple way, an understanding of the benefits that accrue modularity, flexibility, brevity, elegance.
- Laziness: The use of infinite data structures to separate control from action.
- Type discipline
- Polymorphism: The use of generic types to model and capture large classes of data structures by factorizing common patterns.
- Inference: The types of expressions may be determined by simple examination of the program text, understanding such rules.
- User defined types: User defined types as a means to model, a means to extend the language, a means to understand the built in types in a uniform framework.
- Concrete types: Types are concrete. i.e. values that are read or written by the system correspond directly to the abstractions that they represent. More specifically, unlike abstract types which are defined in terms of admissible operations, concrete types are defined by directly specifying the set of possible values.
- Recursion: Recursive definitions as: a means of looping indefinitely; a structural counterpart to recursive data type definition; a means to understand induction in a more general framework than just for natural numbers;
- Operational Semantics: Functional programs execute by rewriting, Calculus as a rewriting system, Reduction, confluence, reasons for preferring normal order reduction.
- Type Classes: Values are to types as types are to classes. Only elementary ideas

The imperative paradigm is smoothly introduced as follows:

Worlds	**The Timeless Worlds**	**World of time**
Domain	**Mathematics**	**Programming**
Syntax	**Expressions**	**Statements**
Semantics	**Values**	**Objects**
Explicit	**Data Structure**	**Control structure**
Thinking with	**Input-Output relations**	**State change**
Abstractions	**Functions**	**Procedures**
Relation	**Denote programs**	**Implement functions**

In the following we spell out some of the points of how FP translates into Imperative P. The examples may be analogized from say how one would teach assembly language to someone who understands structured programming.

- Semantic relations: The central relation is that imperative programming's denotational semantics is FP, FP's operational semantics is imperative programming.

- Operational Thinking: In FP data dependency implicitly determines sequencing whereas in Imp P it is done explicitly. Advantages and disadvantages of operational thinking.
- Environment: In imperative programming there is a single implicit environment memory. In FP there are multiple environments; which could be explicit to the point of first class-ness (the value of variables bound in environments could be other environments). Use of environments to model data abstraction, various object frameworks, module systems.
- Recursion iteration equivalence: General principles
- Type Issues: Monomorphic, polymorphic and latent typing: translating one into another.

Language(s) to convey these two paradigms could be Gofer (or Haskell), Python, Scheme, etc. and can be expected to vary across time as better languages are developed for pedagogic purposes.

References:

- Introduction to Functional Programming, Bird and Wadler.
- Algebra of Programs, Bird
- Structure and Interpretation of Computer Programs, Abelson and Sussman
- Scheme and the Art of Programming, Friedman and Haynes
- Equations Models and Programs, Thomas Myers
- Algorithms + Data Structures = Programs, N Wirth
- Functional Programming, Reade
- Programming from First Principles, Bornat
- Discrete Math with a computer, Hall and Donnell
- Learning Python, Mark Lutz

CS-502 MJ Systems – I (4 Credits)

Objective This course is the first in a sequence which aims at getting an understanding as to what constitutes the various hardware subsystems of a (simple) modern computing device as well as software subsystems that are imperative in order to make effective (and efficient) use of a computing system.

Hardware subsystem

- From a calculator to a stored-program computer: Internal structure of a calculator that leads to this functionality. Machine language and programs writing a sequence of instructions to evaluate arithmetic expressions. Interpreting the computer's behavior when instructions are carried out: the fetch- decode-execute cycle as the basic or atomic unit of a computer's function. Control unit: that performs the fetch-decode-execute cycle.
- Basic Electronics: combinational functions and their implementation with gates and with ROM's; edge-triggered D-flip-flops and sequential circuits; Implementation of data-path and control, using the basic ideas developed so far.
- Parts of a computer: Processor (CPU), memory subsystem, and it's interfaces, peripheral subsystem. and it's interfaces. Parts of these interfaces integrated with the processor, and the remainder contained in the chip-set that supplements the processor. Two main parts of the processor apart from these interfaces: data-path and control (which supervises the data-path) .
- Introductory Machine: Modern computer design, dating back to the 1980's, marks a radical shift from the traditional variety. The new style has given rise to reduced instruction set computers (RISC), as opposed to the older complex instruction set computers (CISC). The ISA of one of the ARM family of processors will be examined. Assembly Language structure, syntax, macros, assembling and disassembling, clock cycle counting
- Pipelining: Improving the performance of a computer and increasing the usage of its subsystems by executing several instructions simultaneously. Analogy to assembly line manufacture of cars. Influence of instruction set design on ease of pipelining. Difficulties with pipelining: structural, data and branch hazards. Branch prediction
- Memory hierarchy: Performance tradeoffs: fast, small, expensive memories (static RAM); slower, larger, inexpensive memories (DRAM); very slow, very large and very cheap memories (magnetic and optical disks). Ideal memory: fast, inexpensive, unbounded size. Ways of creating illusions or approximations of ideal memory. On-chip and off-chip cache memories

Software subsystem

Abstraction of the physical computing device with better properties in order to use it more effectively. Multiplex multiple programs giving an illusion that each having view of their own processor, memory, etc.

- Memory abstraction: virtual memory, required hardware support and programming necessary to make it happen
- Exceptions, traps and interrupts, ISRs. Minimization and handling of non-determinism
- Software subsystem API (system call) and ABI
- Start up scripts, Basics of protected mode and device drivers

References

- Computer Organization and Design, Patterson and Hennessey
- Computer Structures, Ward and Halstead
- Digital Design: Principles and Practices, Wakerley
- Modern Assembly language Programming with the ARM processor, Larry Pyeatt
- Guide to Assembly Language Programming, S P Dandamudi, Springer
- Art of Assembly, Randy Hyde
- Modern Operating systems, Andrew Tanenbaum

CS-503 MJ Mathematics of Computation – I (4 Credits)

Objective: Build the mathematical foundation towards construction and formal reasoning about programs, inferring program behaviour, models of computation and various models which appear natural in the world of programing.

- Logic: Propositional Calculus: Alternative styles: Boolean Algebra, truth tables, equational, deduction, Formal systems, Syntax and semantics, Proof theory and Model theory, consistency and Completeness of different systems.
- Binding Constructs: Abstraction of lambda, for all, program function etc. Free and bound variables, substitution. Common laws.
- Set Theory: Definitions, proofs, notations, building models
- Well formed formulae: Ordinary definition, refinement to types, necessity and limitation of computable type checking.
- Relations: 3 alternative views of foundations of relations: as Cartesian products, as Boolean function(predicates), as power set functions 3 basic types - equivalences, orders, functions
- Graphs: Definition and examples of graphs, Incidence and degree, Handshaking lemma, Isomorphism, Sub-graphs, Weighted Graphs, Eulerian Graphs, Hamiltonian Graphs, Walks, Paths and Circuits, Connectedness algorithm, Chinese Postman problem, Traveling Salesman problem
- Trees: Definition and properties of trees, Pendent vertices, centre of a tree, Rooted and binary tree, spanning trees, minimum spanning tree algorithms, Fundamental circuits, cutsets and cut vertices, fundamental cutsets, connectivity and separativity, max-flow min-cut theorem
- Planar Graphs: Combinational and geometric duals, Kuratowski's graphs Detection of planarity, Thickness and crossings
- Matrix Representation of Graphs: Incidence, Adjacency Matrices and their properties
- Coloring: Chromatic Number, Chromatic Polynomial, the six and five color theorems, the four color theorem

References:

- Logic for CS by Gallier
- Discrete Math by Tremblay Manohar
- Discrete Math by Stanat
- Laws of Logical Calculi by Morgan
- Computer modelling of mathematical reasoning by Bundy
- Predicate Calculus and Program Semantics by Dijkstra
- A Logical Approach to Discrete Math by Gries and Schneider
- Practical Foundations of Mathematics by Paul Taylor
- Conceptual Mathematics by Lawvere
- Logic in Computer Science: Modelling and Reasoning about Systems, by Michael Huth , Mark Ryan Cambridge University Press
- Introduction to Graph Theory, Douglas West
- Graph Theory, Robin Wilson
- Graph Theory with Applications, Bondy, J. A. & U. S. R. Murty [1976], MacMillan
- Graph, Networks and Algorithms, Swamy, M. N. S. & K. Tulsiraman [1981], John Willey

CS-504 MJ Mathematics of Computation – II (Part – I) (2 Credits)

Objective: Build the mathematical foundation towards building machine learning, artificial intelligence, and any other computational models which require to handle data in structured/unstructured form, and/or as vectors/tensors.

- Matrix notation, matrix algebra, matrix operations and their geometric significance, inverse, transpose.
- Vector Spaces and subspaces, linear independence, basis, dimension, linear transformations, four fundamental subspaces
- Orthogonality: orthogonal vectors and subspaces, projections, orthogonal bases, Householder transform
- Eigenvalues and eigenvectors: Their significance, geometric interpretation, similarity transformation and eigenvalues
- Positive definite matrices: It's connection to optima and optimization, recognition of positive definite matrix

References

- Unified introduction to Linear Algebra, Alan Tucker
- Linear Algebra, Serge Lang
- Elementary Linear Algebra, Howard Anton and Chris Rorres

CS-531 RM Research Methodology (4 Credits)
Objective: This course aims at making students aware and familiar with the standard methods of research in the field of computer science.

- History of research, research methodology
- Literature search, selection of research topic (case study based), maintaining records (case study based).
- Ethical considerations, effective verbal and non-verbal communication,
- Data collection, data safety, data integrity
- Implementing research methodology in the field of computer science
- Statistical analysis: The module will consist of case studies of the research performed in various subjects using statistical methods, Error and noise analysis, curve fitting, regression analysis etc.
- Qualitative and Quantitative Research
- Writing research paper and/or project report, making a presentation (seminar/poster)

References:

- John Mandel: The Statistical Analysis of Experimental Data ISBN: 9780486646664
- Research Methodology: A Step-by-Step Guide for Beginners, Kumar, Pearson Education.
- Research Methodology Methods and Techniques, Kothari, C. R., Wiley Eastern Ltd.
- The Research Methods Knowledge Base, by William M. K. Trochim, James P. Donnelly
- Introducing Research Methodology: A Beginner's Guide to Doing a Research Project , by Uwe Flick
- A Guide to Research and Publication Ethics by Partha Pratim Ray, New Delhi Publishers
- RESEARCH & PUBLICATION ETHICS by Wakil kumar Yadav, NOTION PRESS
- Practical Research Methods, Dawson, C., UBSPD Pvt. Ltd.

Semester II

CS-551 MJ Programming – II (4 Credits)

Objective: This is the second course in the sequence on programming with the idea of constructing programs by effective use of data organization while building on ideas picked up in the first course in this sequence. The course is woven with the idea of the dual worlds: algebraic and algorithmic, which leads to a smooth and powerful way to develop programs.

- ADTs and Views: (Algebraic) Formulation as recursive data types, data structure invariants, principles of interface design and its reflection (algorithmic) storage representations, addressing semantics, maximizing abstraction using language features like macros, etc.
- Code: (Algebraic) Pattern matching based recursive definitions wherein exhaustive set of disjoint patterns correspond to total functions leading to runtime bug free programs, recursive code structures follow recursive data structures and its reflection (Algorithmic) refinement of recursive definitions into iterative algorithms, techniques for improving algorithms like sentinel, double pointers, etc.
- [Control as data, loops], [Co-routines vs. subroutines, functions], [General framework for error handling, escape procedures, stack-based software architecture]
- the case studies/examples for the above include but need not be limited to the following: Lists: Various types of representations. Applications: symbol tables, polynomials, OS task queues etc., Trees: Search, Balanced, Red Black, Expression, and Hash Tables Applications: Parsers and Parser generators, interpreters, syntax extenders, Disciplines: Stack, queue etc and uses Polymorphic structures: Implementations

References:

- Data Structures and Algorithms, Aho, Hopcroft and Ullman
- Data Structures, Kruse
- Structure and Interpretation of Computer Programs, Abelson Sussmann
- Functional Programming Henderson
- The Art of Programming Vol. 1. & Vol. 3, D. E. Knuth

CS-552 MJ Mathematics to Data Management (4 Credits)

Objective: Enable the student to appreciate the theoretical underpinnings of the relational model and hence enable them to excel in database design along with the basis which leads to the effective use of SQL.

- DBMS objectives and architectures: Data and models of data, concept of database and information.
- Relational model for data base design: Relational data model, Schema design, Normalization theory, functional dependencies, higher normal forms, integrity rules, Relational operators
- Query language: Relational calculus, relational algebra and SQL
- Database design issues: Query processing, indexing and performance of database system, B-tree index structure, transactions and transaction processing.
- Challenges in database design: conflicting design forces, trade-offs and resolution thereof.

References:

- An introduction to database systems, C. J. Date
- Database Management Systems, Raghu Ramakrishnan, Johannes Gehrke
- Principles of Database Systems Vol. I & Vol II, J. D. Ullman
- Relational Database Index Design and the Optimizers by Tapio Lahdenm, Michael Leach

CS-553 MJ Foundations of Computing (4 Credits)

Objective: Enable the student to get a fair understanding and appreciation of the power as well as the limits of computation.

- Low Power Formalisms Combinational Machines inadequacy
- FSM as acceptor, generator, regular expressions and equivalence
- PDA brief idea, relation between CFG's and programming languages (informal)
- Full Power Mechanisms: (i) Recursive functions, (ii) Turing machines cost models for the RAM, (iii)Post systems/Lambda Calculus/Markov algorithms, (iv) (any one) Use must be stressed along with mutual equivalences. Any of the (iii) should be done so as to give a theoretical backing to the practical notion of 'nonVon-Neumann' language.
- Recursive, Enumerable sets, generators and recognizers
- Universal Algorithm; Diagonal Results: Universal Turing Machine, Uncomputability; Goedel Theorem
- Complexity Basic ideas measuring time usage, time hierarchies Deterministic and Nondeterministic computations.
- Ability of a Mechanism to solve a problem. Formalization of a problem. Church Turing thesis
- Randomness in Computing: Randomness and Complexity, Pseudo-randomness, Cryptography.

References:

- Introduction to the Theory of Computation, Michael Sipser
- Introduction to Computer Theory, Daniel Cohen
- Computability and Complexity from a Programming Perspective, Neil Deaton Jones
- Introduction to Recursive Function theory, Nigel Cutland
- Foundations of Cryptography, Oded Goldreich

CS-554 MJ Mathematics of Computation – II (Part – II) (2 Credits)

Build the mathematical foundation towards building construction of programs whose output has accuracy issues or issues of uncertainty. These outputs are a common consequence of the models used in engineering, physical and biological sciences as well as inaccurate representation of a class of numbers on a finite machine and its attendant consequences.

- Computing and floating-point arithmetic, truncation error, round-off error and it's propagation
- (Numerical) solution of differential equation(s) (initial value problem and boundary value problem) leading to solution of difference equations towards error analysis of solution techniques.
- Numerical solution of linear equations using direct and iterative methods, computation of eigenvalues and eigenvectors
- Interpolation and approximation
- solution of non-linear equation(s), fixed point methods, order of convergence

References

- Numerical Methods for Scientists and Engineers, Chapra, TMH
- Elements of Numerical Analysis, Peter Henrici, John Wiley & Sons.
- Numerical Linear Algebra, Leslie Fox, Oxford University Press.

CS-581 OJT On Job Training (4 Credits)

To be completed, in the summer break after the end of semester examinations of Semester II/IV.

Semester III

CS-601 MJ Programming – III (4 Credits)

Objective: This is the third course of a sequence on programming where a student is exposed to the notion that a correctly running program is **not the end** for an effective programmer and computer scientist. At the end of the course, a successful student should be able to design, analyze and prove termination and correctness of (efficient) algorithms to previously unseen/unknown problem specifications using the general principles covered including being able to model/view a new problem as one of the previously solved problems. In addition, the student should be able to appreciate the value/power of randomness/uncertainty in achieving effective/efficient solutions.

- Probability as a model of mathematical uncertainty: Sample space, events, probabilities on events, conditional probability, independent events, Bayes' theorem
- Random variable, or function defined on a sample space: Expectation of a r.v., expectation of a function of a r.v., variance, notion of a probability distribution, (cumulative) distribution function, some standard discrete and continuous random variates.
- Jointly distributed random variates, Conditional probability and conditional expectation
- Notion of efficiency, Big-Oh notation, it's uses in expressing the efficiency of an algorithm, it's calculation for a given algorithm
- Divide and Conquer as an effective paradigm to decompose a given problem into problems of smaller size and then obtaining the solution to the original one as a composition of the solutions of the subproblems
- Sorting, Searching, Selection
- String processing: Knuth-Morris-Pratt Algorithm, Boyer-Moore Algorithm, pattern Matching.
- Graph Algorithms: DFS, BFS, Bi-connectivity, all pairs shortest paths, strongly connected components, network flow: Edge Saturation and Node Saturation Algorithms, Maximum Matching on Graphs
- Backtracking, Dynamic Programming, Branch & Bound, Greedy: Use of three paradigms for the solution of problems which involve optimization or exploration of a search space with some specific property of the desired solution
- Computation steps being decided by the toss of a coin, or randomized algorithms: An introduction with a few examples and analysis
- Introduction to the theory of NP-Completeness: Non-Deterministic Algorithms, Cook's Theorem, clique decision Problem, Node cover decision problem, chromatic number, directed Hamiltonian cycle, travelling salesman problem, scheduling problems.

References:

- Introduction to Probability, John Freund
- Introduction to probability theory and it's Applications, William Feller
- A first course in Probability, Sheldon Ross
- Introduction to Algorithms, Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein,
- Algorithms, Robert Sedgwick
- The Design and Analysis of Computer Algorithms, A. V. Aho, J. E. Hopcroft, J. D. Ullman
- Algorithm Design: Foundations, Analysis, and Internet Examples, Michael T. Goodrich, Roberto Tamassia
- Algorithm Design, Kleinberg and Tardos
- Combinatorial Algorithms (Theory and Practice), F. M. Reingold, J. Nivergelt and N. Deo

CS-602 MJ Systems – II (4 Credits)

Objective This course is the sequel to course CS-502 MJ and hence it's goal is dovetailed in the same direction as CS-501.

- program in execution --> process abstraction
- lighter version of a unit of computation --> thread,
- multiple such units --> concurrency, sharing and synchronization,
- software support and requirements to achieve these effectively
- Requirements and mechanisms to reach the stage \fI program in execution\fP
- The four dimensions of a programming activity as the basis for systems programming: concept, program generators (humans or other programs), sources and deliverables. For a variety of concepts, a set of program generators generate a set of (possibly overlapping) sources and produce a set of deliverables (executables, libraries, documentation).
- Interpretation as the fundamental activity in Software. Interpreters and interpretation. Program layout strategies on a Von Neumann machine (e.g. Pentium). Division of the final interpretation goal into subtasks and establishing interface export by producer tool and import by consumer tool. Compiler and Assembler translation phases
- Linkers and Loaders Linker as a layout specifying producer and loader as a layout specification consumer. Layout specification strategies: fixed and variable (relocatable and self-relocatable). Layout calculations. Dynamic linking and overlays. Executable format definitions. Object file format as the interface between the compiler and the linker. Few Object file formats like MSDOS, Windows and ELF. Object file manipulation utilities. Editors, version controllers. Version control on object and executable files (e.g. Version support for modules in the Linux kernel)
- Support tools: source browsers, documentation generators, make, GNU auto-conf, git, bug reporting systems, build systems. Debuggers for analysis. Package builders, installers, package managers for deployment
- Persistent storage: Storing and retrieving data (logically coherent from the viewpoint of the producer of the data) with an associated label/name --> file
- collections of files --> file system
- Organizing the collection of names/labels --> directory structure
- File layout
- Managing free space
- Process and file interaction
- Issues in robustness of filesystem
- Designing and organizing metadata for performance, adaptive to encompass newer technology like SSD

References

- Modern Operating Systems, Andrew Tanenbaum
- Design of the Unix Operating System, M, J, Bach
- Compilers: Principles, Techniques and Tools, Aho, Lam Sethi
- John Levine, Linkers and Loaders, John Levine
- System Software: An Introduction to Systems Programming, Leland L. Beck

CS-603 MJ Systems – III (4 credits)

Objective: This is the third 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 communication of data across widely differing geographical scales and with very little hold on the variety of software/hardware combinations of which this subsystem is a part. All this has to be achieved in the context that the entire underlying communication setup cannot be assumed to be reliable.

- Understand the nature of layering and it's naturalness in the design
- Examine the assumptions of the underlying communication setup (hardware/technology)
- Physical Layer, Transmission media, digital transmission, transmission & switching
- Data link layer and it's interface with the physical layer below and the network layer above, a look at Ethernet and IEEE 802.11
- Network layer: the service it is expected to provide and how it is achieved in the context of IP, SDN: an introduction
- Transport layer: the service it is expected to provide and how it is achieved in the context of TCP
- Application layer: Sitting at the top of the network stack, the service it is expected to provide and some sample protocols under the TCP/IP protocol suite
- The Session and presentation layers which have utility in certain contexts, primarily in the telecom sector.
- Network Security and how it should be integrated into the system
- the place and interaction of this sub-system in the larger context of the software/hardware system of which this a part and whose objective is to make effective and efficient use of the computing device
- Communication protocols at the small scale: communication at the processor level

Reference:

- Data and communications, W. Stallings
- Computer networks: A systems approach, Peterson and Davie
- Computer Networks, A. S. Tanenbaum
- UNIX Network Programming, Stevens
- TCP/IP Illustrated Volume 1, 2, 3, Richard Stevens

CS-604 MJ Formal Methods – I (2 Credits)

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 University 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-631 RP: Research Project I (4 Credits)

The student is expected to work for a semester (at an average of 12-15 hours per week) on a project assigned by the guide. Presentation will be given by the students

Semester IV

CS-651 MJ Formal Methods – II (4 Credits)

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 al
- 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 al
- 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-652 MJ Software Comprehension (4 Credits)

Motivation:

It is estimated that the maintenance (including minor modifications, feature enhancements, etc.) of existing systems consume 50% to 80% of the resources in the total software budget. It is further estimated that around 50% or more time is spent on the task of software comprehension (understanding the program/code). While traditional software engineering methods focuses on increasing the productivity and quality of systems under development or being planned this course will address the complementary problem.

Objectives:

The broad objective of the course is:

The ability to read and comprehend large pieces of software in an organized manner, even in the absence of adequate documentation.

The achievement of the above goal will imply the following milestones:

- **(Re)documentation:** Creation or revision of system documentation at the same level of abstraction
- **Design Rediscovery:** Construct the design using domain knowledge and other external information where possible along with information extracted from the code to create a model of the system at a higher level of abstraction (than the code).
- **Restructuring(/design):** Transformations in design at the same level of abstraction while maintaining same level of functionality and semantics.
- **Modification:** Modify design for change in functionality (addition/deletion).

Contents:

1. Basic programming elements, data types, control flow
2. Large projects: project organization, build process, configuration, revision control, coding standards and conventions.
3. Code reading tools: Using existing tools like an editor, regular expression matching using grep, file difference using diff, compiler, code browsers, etc. and writing your own if none of the existing do the desired task
4. Code analysis: Using (and writing) tools for graphing (control flow, data flow, dependence analysis), profiling and testing.

All the above should be covered in the context of some (ideally one or at most two) large software like Apache, Linux kernel, ArgoUML, libfftw, etc. In addition, it is expected that the software chosen be changed across different offerings of the course. With reference to the above contents, the coverage will reinforce and complement, where pertinent, material which has been addressed in other courses.

References:

- The Practice of Programming, Kernighan B, Pike R, Prentice-Hall India 2005
- Working Effectively with Legacy Code, Feathers M, Prentice Hall 2004
- Refactoring: Improving the Design of Existing Code, Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, Addison Wesley

CS-653 MJ Systems – IV (4 Credits)

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, isolation and durability (ACID) properties; lock-based concurrency control, transactions as a concurrency control mechanism.
- Concurrent programing - concurrency as a fundamental aspect of programing 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-681 RP: Research Project II (6 credits)

- The student is expected to work for a semester (at an average of 15 hours per week) on a project assigned by the guide. Presentation will be given by the students. Student is encouraged to publish his/her work to earn extra credits.

The departmental faculty will decide on the syllabus for the following subjects which may be offered as electives in first year and Second year as per the need and demands of the current industry/research institutes' requirements. Each elective will be of 4 credits.

Title
Foundations of Artificial Intelligence
Foundations of Machine Learning
Foundations of Human Software Interface
Foundations of Software Security
Foundations of Pervasive Computing
Foundations of Natural Language Processing
Foundations of Computational Text Processing
Foundations of Computational Finance
Foundations of Data Science
Foundations of Big Data
Introduction to Database systems

Title (to be offered in later semesters)
Artificial Intelligence and Machine Learning
Block-chain technology
Parallel Algorithms
Natural Language Processing
Computational Finance
Data Science
Quantum Computing
Advanced Typing
Typing for better programs
Advanced data structures
Advanced algorithms
File system implementation
Advanced Numerical Methods
Advanced Low Level Programming
Advanced Database Systems
Higher order logic
Software Engineering
Pragmatic Software Construction
Problem Solving
Modern Web Programming