Savitribai Phule Pune University

(Formerly University of Pune)

Centre for Distance and Online Education

(CDOE)

Master of Computer Application

(MCA) (2024 Pattern)

As per NEP 2020 Curriculum

Faculty of Interdisciplinary Studies

Master of Computer Application Curriculum (2024 Pattern)

2-year, 4 Semester Programme

1. Preamble of the syllabus:

This program is offered at the Department of Computer Science, Savitribai Phule Pune University, Pune. Master of Computer Applications (M.C.A.) program is of 96 credits.

---

2. Objectives:

The objective of the M.C.A. programme is to train the students to meet the challenges of the Software Industry and R&D Sector with computational techniques.

The structure of the program is as follows:

a) In semesters I, there will be 16 credits of mandatory major courses, 4 credits for a major elective and Research Methodology of 4 credits, making a total of 24 credits.

b) In semester II, there will be 18 credits of mandatory major courses and 2 major electives of 4 credits each, making a total of 26 credits.

c) The student can accumulate a total of 50 credits, in year I, out of which 34 credits towards mandatory major subjects, 12 credits for major electives and 4 credits are for Research Methodology.

d) If the student decides to exit, he/she will be awarded PG Diploma (after 3-year UG degree course), after completion of additional 4 credits for OJT.

e) In semester III, there will be 8 credits of mandatory major courses, 3 major electives of 4 credits each and a research project of 6 credits, making a total of 26 credits.

f) Semester IV will be full time Industrial training/Internship (12 credits) and 2 major electives of 4 credits each, which can be run online or can be completed on MOOC platforms. The total credits in the semester is 20.

g) To earn a 2-year PG degree, student will have to earn a total of 96 credits, out of which 42 credits for mandatory major subjects, 32 credits for major electives and 4 credits are for Research Methodology, 12 credits of OJT/FP and 6 credits of research project.

h) A student cannot register for the third semester, if he/she fails to complete 50% credits of the total credits expected to be 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.

i) A student will obtain non-zero credits only on obtaining a pass grade in a course.

j) In addition to CBCS guidelines for classroom delivery hours, 2 hours per subject across the course will be devoted towards outside classroom interactions.

k) 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.

l) 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:

- Written Test and/or Mid Term Test (not more than one for each course);
- Term Paper;
- Journal/Lecture/Library notes;
- Seminar presentation;
- Short Quizzes;
- Assignments;
- Extension Work;
- Research Project by individual students or group of students; or
- 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) A student will be rewarded by a grade/mark for Internship/Industrial Training upon the submission of certificate of completion, duly signed and sealed from the mentor from Industry/Research Organization/Academic Institution by rating between 1 to 6, where 1 being lowest and 6 being highest.

4. Completion of Degree Programme:

a) In order to earn two-year degree of Master of Computer Applications (M.C.A.) course a student has to obtain 96 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 taken into account for calculating GPA and overall grade. Only those courses in which the student has passed will be taken into account for calculating the GPA 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.

Semester	No of Courses	Credits
Semester I	06	24
Semester II	06	26
Semester III	06	26
Semester IV	06	20
	Total Credit	**96**

Course Structure

Year	Level	Semester	Subject Code	Subject Title	Major		RM	OJT/FP	RP
					Mandatory	Elective			
I	6	I	CA-501 MJ	Programming from First Principles	4				
			CA-502 MJ	Processor Architecture and Design	4				
			CA-503 MJ	Computational Mathematics	4				
			CA-504 MJ	Persistent Data Management	4				
			CA-510* MJ	Elective		4			
			CA-531 RM	Research Methodology			4		
		II	CA-551 MJ	Data Organization for Program Construction	4				
			CA-552 MJ	Software Sub-systems for Hardware Virtualization	4				
			CA-553 MJ	Computational Thinking	4				
			CA-554 MJ	Foundations of Data Analytics	4				
			CA-555 MJ	Foundations of Software Development	2				
			CA-560* MJ	Elective 1		4			
			CA-561* MJ	Elective 2		4			

Year	Level	Semester	Subject Code	Subject Title	Major		RM	OJT/FP	RP	
					Mandatory	Elective				
II	6.5	III	CA-601 MJ	Communication Protocols	4					
			CA-602 MJ	Software Component Engineering	4					
			CA-610* MJ	Elective 1		4				
			CA-611* MJ	Elective 2		4				
			CA-612* MJ	Elective 3		4				
			CA-631 RP	Research Project					6	
		IV	CA-681 OJT	Internship				12		
			CA-660* MJ	Online Elective 1 / MOOC-1		4				
			CA-661* MJ	Online Elective 2 / MOOC -2		4				

Semester I

CA-501 MJ Programming from First Principles

(4 Credits)

Objective: Two paradigms are used as vehicles to carry the ideas for this course: the functional and the imperative. The central issue here is to be able to use the computer as a high-level tool for problem solving. The paradigm conveyed may be simply expressed as: A modern non-strict functional language with a polymorphic type system is the medium for this part. Important ideas that are to be covered include:

- **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: The imperative paradigm is smoothly introduced as follows:

Worlds	**The Timeless worlds**	**World of Time**
Domain	Mathematics	Programming
Syntax	Expressions	Statements
Semantics Explicit	Values Data Structure	Objects 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 Imp 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 purpose.

References for further Reading:

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

CA-502 MJ Processor Architecture and Design

(4 Credits)

Objective: This course is 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.

- **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 trade-offs: 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**

References for further Reading:

1. Computer Organization and Design, Patterson and Hennessey
2. Computer Structures, Ward and Halstead
3. Digital Design: Principles and Practices, Wakerley
4. Modern Assembly language Programming with the ARM processor, Larry Pyeatt
5. Guide to Assembly Language Programming, S P Dandamudi, Springer
6. Art of Assembly, Randy Hyde
7. Modern Operating systems, Andrew Tanenbaum
8. Intel® 64 and IA-32 Architectures

CA-503 MJ Computational Mathematics

(4 Credits)

Objective: Build the mathematical foundation towards reasoning about the correctness issues in general and towards 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.

- **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.
- **Well-formed formulae:** Ordinary definition, refinement to types, necessity and limitation of computable type checking.
- **Graphs & Trees:** Definition and examples of graphs, Incidence and degree, Handshaking lemma, Isomorphism, Sub-graphs, Weighted Graphs, Eulerian Graphs, Definition and properties of trees, Pendent vertices, centre of a tree, Rooted and binary tree, spanning trees, minimum spanning tree algorithms, Matrix Representation of Graphs
- **Matrices:** Matrix notation, matrix algebra, matrix operations and their geometric significance, inverse, transpose.
- **Vector Spaces and subspaces**, linear independence, basis, dimension, linear transformations, orthogonal vectors and subspaces, projections, orthogonal bases,
- **Eigenvalues and eigenvectors:** Their significance, geometric interpretation, similarity transformation and eigenvalues
- **Computing and floating-point arithmetic**, truncation error, round-off error and it's propagation
- (Numerical) solution of differential equation(s)
- Numerical solution of linear equations using direct and iterative methods, computation of eigenvalues and eigenvectors

References for further Reading:

1. Logic for CS by Gallier
2. Discrete Math by Tremblay Manohar
3. Discrete Math by Stanat
4. Laws of Logical Calculi by Morgan

5. Computer modelling of mathematical reasoning by Bundy
6. Predicate Calculus and Program Semantics by Dijkstra
7. A Logical Approach to Discrete Math by Gries and Schneider
8. Practical Foundations of Mathematics by Paul Taylor
9. Logic in Computer Science: Modelling and Reasoning about Systems, by Michael Huth , Mark Ryan Cambridge University Press
10. Introduction to Graph Theory, Douglas West
11. Graph Theory, Robin Wilson
12. Graph Theory with Applications, Bondy, J. A. & U. S. R. Murty [1976], MacMillan
13. Graph, Networks and Algorithms, Swamy, M. N. S. & K. Tulsiraman [1981], John Willey
14. Unified introduction to Linear Algebra, Alan Tucker
15. Linear Algebra, Serge Lang
16. Elementary Linear Algebra, Howard Anton and Chris Rorres
17. Numerical Methods for Scientists and Engineers, Chapra, TMH
18. Elements of Numerical Analysis, Peter Henrici, John Wiley & Sons.
19. Numerical Linear Algebra, Leslie Fox, Oxford University Press.

CA-504 MJ Persistent 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.

- Objectives and architectures.
- **Data Models:** Conceptual model, ER model, object-oriented model, UML Logical data model, Relational, object oriented, object relational.
- **Physical data models:** Clustered, unclustered files, indices (sparse and dense), B+ tree, join indices, hash and inverted files, grid files, bulk loading, external sort, time complexities and file selection criteria.
- **Relational database design:** Schema design, Normalization theory, functional dependencies, higher normal forms, integrity rules, Relational operators.
- **Object oriented database design**: Objects, methods, query languages, implementations, Comparison with Relational systems, Object orientation in relational database systems, Object support in current relational database systems, complex object model, implementation techniques.
- **Mapping mechanism: C**onceptual to logical schema, Key issues related to for physical schema mapping.
- **DBMS concepts:** ACID Property, Concurrency control, Recovery mechanisms, case study Integrity, Views & Security, Integrity constraints, views management, data security.
- **Query processing, Query optimization:** Heuristic and rule-based optimizers, cost estimates, Transaction Management.
- **Case Study:** Case study for Understanding the transaction processing Concurrency and recovery protocols, query processing and optimization mechanisms through appropriate queries in SQL and PLSQL using one or more of Oracle, PostgreSQL, MySQL, some other Open Source Database Package.
- **Web based data model:** XML, DTD, query languages.
- **Advanced topics:** Other database systems, distributed, parallel and memory resident, temporal and spatial databases. Introduction to data warehousing, On-Line Analytical Processing, Data Mining. Bench marking related to DBMS packages, database administration. Introduction to Big Data. Recent advances in Database Management.

References for further Reading:

1. An introduction to database systems, C. J. Date
2. Database Management Systems, Raghu Ramakrishnan, Johannes Gehrke
3. Principles of Database Systems Vol. I & Vol II, J. D. Ullman
4. Relational Database Index Design and the Optimizers by Tapio Lahdenm, Michael Leach.
5. Database System Concepts, Silberschatz, Korth and Sudarshan, McGraw Hill

CA-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 for further Reading:

- 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

CA-551 MJ Data Organization for Program Construction (4 Credits)

Objective: Constructing of 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 it's 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 it's 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 for further Reading:

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

CA-552 MJ Software Sub-systems for Hardware Virtualization (4 Credits)

Objective: This course is the sequel to course CA-102 and hence it's goal is dovetailed in the same direction as CA-101.

- Simple computer systems made up of a single processor and single core memory spaces and their management strategies.
- Processes as programs with interpolation environments. Multiprocessing without and with IPC. Synchronization problems and their solutions for simple computer systems.
- Memory management: segmentation, swapping, virtual memory and paging. Bootstrapping issues. Protection mechanisms.
- Abstract I/O devices in Operating Systems. Notions of interrupt handlers and device drivers. Virtual and physical devices and their management.
- Introduction to Distributed Operating Systems. Architecture designs for computer systems with multiple processors, memories and communication networks. Clocking problem and Lamport's solution.
- Illustrative implementation of bootstrap code, file systems, memory management policies etc

References for further Reading:

1. S. Tanenbaum, Modern Operating Systems, Pearson Education
2. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating Systems Concepts, Wiley Nutt, Operating System, Pearson Education
3. S. Tanenbaum, Distributed Operating Systems, Prentice Hall
4. M. Singhal & N. Shivaratri, Advanced Concepts in Operating Systems, McGraw Hill Understanding the Linux Kernel, 2nd Edition By Daniel P. Bovet, Oreilly
5. The Design of Unix Operating System Maurice Bach, Pearson

CA-553 MJ Computational Thinking

(4 Credits)

Objective: A student is exposed to the notion that a correctly running program is not the be all and end all for an effective programmer and computer scientist. At the end of the course, a successful student should be able to design, analyse 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 random variable., expectation of a function of random variable, variance, notion of a probability distribution, (cumulative)distribution function, some standard discrete and continuous random variables.
- Jointly distributed random variables, Conditional probability and conditional expectation.
- Notion of efficiency, Big-Oh notation, it's use in expressing the efficiency of an algorithm, its 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, Biconnectivity, 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 for further Reading:

1. Introduction to Probability, John Freund
2. Introduction to probability theory and its Applications, William Feller
3. A first course in Probability, Sheldon Ross
4. Introduction to Algorithms, Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein,
5. Algorithms, Robert Sedgwick
6. The Design and Analysis of Computer Algorithms, A. V. Aho, J. E. Hopcroft, J. D. Ullman
7. Algorithm Design: Foundations, Analysis, and Internet Examples, Michael T. Goodrich, Roberto Tamassia
8. Algorithm Design, Kleinbergand Tardos
9. Combinatorial Algorithms (Theory and Practice) , F. M. Reingold, J. Nievrgelt and N. Deo

CA-554 MJ Foundations of Data Analytics

(4 Credits)

Objective: Build the mathematical foundation towards construction of programs that are supposed to handle huge amounts of data. There are various issues inherent in data that need to be handled effectively and efficiently before computers can be deployed to help us.

- Descriptive and Predictive Analytics, Measures of Central Tendency
- Probabilistic methods: probability, probability distributions, moments, Sampling Distribution and Hypothesis Testing
- Numerical methods: numerical linear algebra for data analysis
- Data cleanup: missing value analysis, interpolation and extrapolation, outlier analysis,
- Multidimensional data: handling higher dimensional data, techniques like Support Vector Machines for classification and regression of multidimensional data
- Teacher may use free and open-source software tools like scipy, numpy, scilab, and so on to help students apply the theoretical knowledge in challenging applications.

References for further Reading:

- Statistics and Data Analysis: From Elementary to Intermediate. Prentice Hall, 1999, by Tamhane and Dunlop. ISBN: 9780137444267.
- Statistical Learning Theory (1998), by Vladimir Vapnik
- The Nature of Statistical Learning Theory, 1995, by Vladimir Vapnik
- The Deep Learning textbook, MIT presss, by Yoshua Bengio et.el. [Freely available at: https://www.deeplearningbook.org]
- Foundations of the Theory of Probability: Second English Edition (Dover Books on Mathematics), by A. N. Kolmogorov
- https://www.scilab.org

CA-555 MJ Foundations of Software Development

(2 Credits)

Objective: Inculcate the discipline and build skills required to handle complexities of modern software development involving intricate interactions between large teams. To introduce the engineering issues of software development and appreciate the role of state of the art programming science we have at our disposal.

- Team Communication issues: communication issues involving various stakeholders, specifications, requirement analysis, design documentation, coding and final builds, feedback loops involving stakeholders
- Software Interfaces: structural typing to help build complex leak-proof interfaces, theorem provers to rescue, interface analysis using type theoretical concepts
- From requirements to blobs: issues in building complex software, reproducible builds, challenges in meaningful versioning
- Software complexity: cyclomatic complexity, the myth of LoC, coupling and cohesion, software metrics
- Software reliability: concepts of Mean Time Between Failure (MTBF), Mean Time To Failure (MTTF), Mean Time To Repair (MTTR), software maintainability
- Teacher may use free and open-source software tools like Haskell [ghc], scala, PVR, GNU-Linux/Unix, make, git, gitlab, and so on to help students apply the software engineering skills in challenging situations

References for further Reading:

- Software Engineering Foundations: A Software Science Perspective, by Yingxu Wang, ISBN 9780849319310
- The Mythical Man-Month: Essays on Software Engineering, 1995, by Fred Brooks ISBN 978-0-201-83595-3
- A Discipline of Programming, by Edsger W. Dijkstra
- Type Theory and Formal Proof: An Introduction, by Rob Nederpelt
- Types and Programming Languages, MIT Press, by Benjamin Pierce, ISBN 978-0-262-16209-8.