DEPARTMENT OF COMPUTER SCIENCE

SAVITRIBAI PHULE PUNE UNIVERSITY

Syllabus for
2 Years Master of Computer Applications (M.C.A.)
(Under CBCS guidelines)

To be implemented from Academic Year 2020 - 2021

Department of Computer Science,

Savitribai Phule Pune University,

Pune-411 007.

Department: Department of Computer Science, SPPU

Course: M.C.A. (Master in Computer Applications)

Duration: 2 Years

Total Number of Credits: 80

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 80 credits. The objective of the M.CA. 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 and II there will be FIVE courses per semester and each course will be of 4 credits.
   b) In semester III the students can choose courses to earn 20 credits. These credits can be earned by opting the courses offered in the department or from other departments on the campus as per CBCS guidelines.
   c) Semester IV will be full time Industrial training/Internship (18 credits) and a Seminar (2 credits).
   d) 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.
   e) A student will obtain non-zero credits only on obtaining a pass grade in a course.
   f) In addition to CBCS guidelines for classroom delivery hours, 2 hours per subject across the course will be devoted towards outside classroom interactions.

2. **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:
      a. Written Test and/or Mid Term Test (not more than one for each course);
      b. Term Paper;
      c. Journal/Lecture/Library notes;
      d. Seminar presentation;
      e. Short Quizzes;
      f. Assignments;
      g. Extension Work;
      h. Research Project by individual students or group of students; or

i. 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) Students will be awarded grade/marks upon completion of full time Internship/Industrial Training .Internship/Industrial Training will be completed upon the submission of certificate of completion, duly signed and sealed by the mentor with a rating 1 to 5, where 1 being least and 5 being highest.

f) Topic of the Seminar will be decided by the student in consultation with assigned teacher. This will be evaluated by the respective teacher based on the presentation/submission by the student.

3. **Completion of Degree Programme:**
   a) In order to pass the Master of Computer Applications (M.C.A.) course a student has to obtain 80 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.

4. **The overall course structure is given below:**

Course Structure

Semester I		
Subject Code	**Subject Title**	**#Credits**
CA-101	**Programming Fundamentals**	**04**
CA-102	**Computer Organization**	**04**
CA-103	**Mathematical Foundations**	**04**
CA-104	**Computational Mathematics**	**04**
CA-105	**Database Management Systems**	**04**
UGC Recommended Audit Courses	**Human Rights Education-1** **Human Rights Education-2** **Skill Development -1** **Skill Development -2**	**01** **01** **01** **01**
	Total Credits	**20**

Semester II		
Subject Code	**Subject Title**	**#Credits**
CA-201	**Data Structures**	**04**
CA-202	**Operating Systems**	**04**
CA-203	**Design and Analysis of Algorithms**	**04**
CA-204	**Computer Networks**	**04**
CA-205	**Software Engineering**	**04**
UGC Recommended Audit Courses	**Skill Development -3** **Skill Development -4** **Cyber Security-1**	**01** **01** **01**
	Total Credits	**20**
Semester III		
Subject Code	**Subject Title**	**#Credits**
CA-3××*	**Elective**	**04**
CA-3××	**Elective**	**04**
CA-3××	**Elective**	**04**
CA-3××	**Elective**	**04**
CA-3××	**Elective**	**04**
UGC Recommended Audit Courses	**Cyber Security-2** **Cyber Security-3** **Cyber Security-4**	**01** **01** **01**
	Total Credits	**20**
Semester IV		
Subject Code	**Subject Title**	**#Credits**
CA-401	**Full Time Industrial Training / Internship**	**18**
CA-402	**Seminar**	**02**
	Total Credits	**20**

- * - ×× can range from 01 to 99.

Semester I

CA-101 Programming Fundamentals (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:

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-102 Computer Organization (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.

- **Introduction to Digital Systems:** Introduction to Digital electronics, Digital and Analog Signals and Systems, Binary Digits, Logic Levels, and Digital Waveforms, Logic Systems-Positive and negative, Logic Operations, Combinational and Sequential Logic Functions, Programmable Logic.

- **Number Systems and Codes**: Introduction to Number Systems-Types-Decimal, Binary, Octal, Hexadecimal; Conversion from one number system to other; Binary arithmetic operations; Representation of Negative Numbers;1's complement and 2's complement, Complement arithmetic, BCD code.

- **Logic Gates:** Logical Operators, Logic Gates-Basic Gates, Other gates, Active high and Active low concepts, Universal Gates and realization of other gates using universal gates, Gate Performance Characteristics and Parameters.

- **Boolean Algebra:** Rules and laws of Boolean algebra, Demorgan's Theorems, Boolean Expressions and Truth Tables, Standard SOP and POS forms; Minterm and Maxterms, Canonical representation of Boolean expressions, Simplification of Boolean Expressions, Minimization Techniques for Boolean Expressions using Karnaugh Map and Quine McCluskey Tabular method.

- **Combinational Circuits-Part 1:** Introduction to combinational Circuits, Adders-Half-Adder and Full-Adder, Subtractors- Half and Full Subtractor; Parallel adder and Subtractor.

- **Combinational Circuits- Part 2:** Multiplexer, Demultiplexer, Encoder, Priority Encoder; Decoder, BCD to Seven segment Display Decoder/Driver, LCD Display, and Comparators.

- **Sequential Circuits:** Introduction to Sequential Circuits, Flip-Flops: Types of Flip Flops -RS, T, D, JK; Triggering of Flip Flops; Flip Flop conversions; Master-Slave JK.

- **Registers & Counters:** Synchronous/Asynchronous counter operation, Up/down synchronous counter, application of counter, Serial in/Serial out shift register, Serial in/Serial out shift register, Serial in/parallel out shift register, parallel in/ parallel out shift register, parallel in/Serial out shift register, Bi-directional register..

- **Computer Concepts:** Basic Computer System, concepts of hardware and software, Operating Systems, Microcontrollers and Embedded Systems., Digital Signal Processing, Digital Signal Processor (DSP).

- **Memory and Storage:** Semiconductor Memory Basics, Types-RAM, ROM, Programmable ROMs, Flash Memory, Memory Expansion, Special Types of Memories, Magnetic and Optical Storage.

References

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

CA-103 Mathematical Foundations (4 Credits)

Objective: Build the mathematical foundation 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.

- **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 cut sets, connectivity and separativity, max-flowmin-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:

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. Conceptual Mathematics by Lawvere
10. Logic in Computer Science: Modelling and Reasoning about Systems, by Michael Huth , Mark Ryan Cambridge Univeristy Press
11. Introduction to Graph Theory,Douglas West
12. Graph Theory,Robin Wilson
13. Graph Theory with Applications, Bondy, J. A. & U. S. R. Murty [1976], MacMillan
14. Graph, Networks and Algorithms, Swamy, M. N. S. & K. Tulsiraman [1981], John Willey

CA-104 Computational Mathematics (4 Credits)

Objective: Build the mathematical foundation 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 representation of a class of numbers on a finite machine and its attendant consequences.

- 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.

- Computing and floating point arithmetic, truncation error, round-off error and its 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

1. Unified introduction to Linear Algebra, Alan Tucker
2. Linear Algebra, Serge Lang
3. Elementary Linear Algebra, Howard Anton and Chris Rorres
4. Numerical Methods for Scientists and Engineers, Chapra, TMH
5. Elements of Numerical Analysis, Peter Henrici, John Wiley&Sons.
6. Numerical Linear Algebra, Leslie Fox, Oxford University Press.

CA-105 Database Management Systems (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 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 inverte 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:

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 Sudershan, McGraw Hill

Semester II

CA-201 Data Structures (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:

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

CA-202 Operating Systems (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

Reference:

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, Pearso

CA-203 Design and Analysis of Algorithms (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 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, 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.

- **Notion of an Algorithm :** Fundamentals of Algorithmic Problem Solving – Important Problem Types – Fundamentals of the Analysis of Algorithm Efficiency – Analysis Framework – Asymptotic Notations and its properties – Mathematical analysis for Recursive and Non-recursive algorithms.

- **Divide-and-Conquer and Greedy Method :** Traveling Salesman Problem – Knapsack Problem – Assignment problem. Divide and conquer methodology – Merge sort – Quick sort – Binary search – Multiplication of Large Integers – Strassen's Matrix Multiplication.

- **Dynamic Programming and Greedy Technique** : Computing a Binomial Coefficient – Warshall‟s and Floyd‟ algorithm – Optimal Binary Search Trees – Knapsack Problem and Memory functions. Greedy Methods with Examples Such as Optimal Reliability Allocation, Knapsack, Minimum Spanning Trees – Prim's and Kruskal's Algorithms, Single Source Shortest Paths – Dijkstra's and Bellman Ford Algorithms.

- **Iterative Improvement** : The Simplex Method-The Maximum-Flow Problem – Maximum Matching in Bipartite Graphs.

- **Coping with the Limitations of Algorithm Power :** Limitations of Algorithm Power-Lower-Bound Arguments-Decision Trees-P, NP and NP-Complete Problems–Coping with the Limitations – Backtracking – n-Queens problem – Hamiltonian Circuit Problem – Subset Sum Problem-Branch and Bound – Assignment problem – Knapsack Problem – Traveling Salesman Problem- Approximation Algorithms for NP – Hard Problems – Traveling Salesman problem – Knapsack problem.

References:

1. Introduction to Probability, John Freund
2. Introduction to probability theory and it's 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, Kleinberg and Tardos
9. Combinatorial Algorithms (Theory and Practice) , F. M. Reingold, J. Nivergelt and N. Deo

CA-204 Computer Networks (4 Credits)

Objective: 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 subsytem 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:

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

CA-205 Software Engineering (4 Credits)

Objective: This is the second course in the collection on software development. At the end of the course a successful student should be able to apply modern techniques of disciplined software development in large scale software development project and understand the theoretical underpinnings involved.

- Concepts of software management, The software crisis, principles of software engineering, programming in the small Vs programming in the large.
- Software methodologies/processes, The software life cycle, the waterfall model and variations, introduction to evolutionary and prototyping approaches.
- Software measurement.
- Object-oriented requirements analysis and modelling: Requirements analysis, requirements.
- Solicitation, analysis tools, requirements definition, requirements specification, static and dynamic specifications, requirements review. (just revisited).
- Software architecture.
- Software design, Design for reuse, design for change, design notations, design evaluation and validation 8. Implementation, Programming standards and procedures, modularity, data abstraction, static analysis, unit testing, integration testing, regression testing, tools for testing, fault tolerance.
- User considerations, Human factors, usability, internationalization, user interface, documentation, user manuals.
- Documentation, Documentation formats, tools.
- Project management, Relationship to life cycle, project planning, project control, project organization, risk management, cost models, configuration management, version control, quality assurance, metrics.
- Maintenance, the maintenance problem, the nature of maintenance, planning for maintenance.
- Configuration Management.
- Tools and environments for software engineering, role of programming paradigms, process maturity.

- Introduction to Capability Maturity Model People Capability Maturity Model Software Acquisition Capability Maturity Model Systems Engineering Capability Maturity Model.

- IEEE software engineering standards.

References:

1. Software Engineering, Ian Sommerville, Addison Wesley, (Note: This is also the preferred textbook for the IEEE Software Engineering Certificate Program.)
2. The Engineering of Software, Dick Hamlet, Joe Maybee, Addison Wesley,
3. Introduction to the Team Software Process, Watts S. Humphrey, Addison Wesley,
4. Software Engineering A Practitioner's Approach European Adaption, 5th Edn., Roger S. Pressman, adapted by Darrel Ince, McGraw Hill,
5. Software Engineering Theory and Practice, Shari Lawrence Pfleeger, Prentice Hall,
6. Practical Software measurement, Bob Huges, McGraw Hill,
7. Human Computer Interaction, Dix, Finlay, Abowd and Beale, Prentice Hall,
8. Software Project Management, Bob Huges & Mike Cotterell, McGraw Hill,

Semester III

- **5 Electives (Each 4 credits)**

Semester IV

CA-401 Industrial Training/ Internship (18 Credits)

- CA-401 - The students enrolled for MCA course have to undergo an Industrial Training/Internship for a minimum of 20 weeks duration in Software Industry and/or R&D Sector.

CA-402 Seminar (2 credits)

- Presentation will be given by students.