Syllabus for M.C.A. / M.Sc. /M.Tech. w.e.f. 2008-09

First three semesters of the M.C.A., M.Sc. and M.Tech courses are same in content and prerequisite/co-requisite requirements.

Semester 1 (5 Credits Each Course)

- CS-101 Introduction to Programming
- CS-102 Computer Organization
- CS-103 Mathematical Foundations
- CS-104 Concrete Maths & Graph Theory
- CS-105 Database Management System

Semester 2 (5 Credits Each Course)

- CS-201 Numerical Methods
- CS-202 Data Structures & Algorithms
- CS-203 Low-level Programming
- CS-204 Operating Systems
- CS-205 Science of Programming

Semester 3 (5 Credits Each Course)

- CS-301 Design & Analysis of Algorithms
- CS-302 Theoretical Computer Science
- CS-303 Computer Networks
- CS-304 Systems Programming
- CS-305 Distributed computing

Semester 4 (MCA Only) (5 Credits Each Course)

- CS-401 Computer Graphics
- CS-402 Modelling and Simulation
- CS-403 Operations Research
- CS-404 Software Engineering - I
- CS-405 Elective *

Semester 5 (MCA Only) (25 Credits)

- CSMCP: Full-time Industrial Training

Semester 6 (MCA Only) (5 Credits Each Course)

- CS-601 Programming Paradigms
- CS-602 Software Engineering – II
- CS-603 Applications of Software Engineering and Programming Paradigms
- CS-604 Elective *
- CS-605 Elective *
Semester 4 (M.Sc. Only)

- CS-411  Software Engineering (5 Credits)
- CS-601  Programming Paradigms (5 Credits)
- CS-405  Elective * (5 Credits)
- CS-MSP Degree Project (10 Credits)

Semester 4 (M.Tech Only)

- CS-411  Software Engineering (5 Credits)
- CS-601  Programming Paradigms (5 Credits)
- CS-405  Elective * (5 Credits)
- CS-MTP Degree Project (10 Credits)

Elective Courses (offered in the last few years)

- Genetic Algorithms
- Management Information Systems
- Object Oriented Modelling and Design
- Motivation and Emotion
- Windows Programming
- Compiler Construction
- Advanced Algorithms
- Network Security
- System Administration
- COM - Component Object Modelling
- Advanced Networks
- Program Analysis
- Distributed Systems
- Machine Learning
- Programming in Real World
- Information Security
- Grid Computing
- Enterprise Application Integration
- Information Audit and Security
- Data Mining
- Procedural Texture Generation and Shading
CS-101 - Introduction to Programming

- **Contents:**
  Two paradigms are used as vehicles to carry the ideas and execute practical for this course the functional and the imperative.

**The Functional Paradigm:**
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. The currently used language is the internationally standardized language, Haskell.

Important ideas that are to be covered include:

1. **Standard Constructs**
   Function and type definition, block structure.
   Guarded equations, pattern matching.
   Special syntax for lists, comprehension.

2. **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. (MF)

3. **Calculus**
   A direct way for denoting functions.

4. **First-Class-ness**
   All values are uniformly treated and conceptualized.

5. **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.

6. **Laziness** The use of infinite data structures to separate control from action.

7. **Type discipline**

8. **Polymorphism:**
   The use of generic types to model and capture large classes of data structures by factorizing common patterns.

9. **Inference:**
   The types of expressions may be determined by simple examination of the program text.
   Understanding such rules.

10. **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.

11. **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.

12. **Recursion**
    Recursive definitions as a means of looping indefinitely
    a structural counterpart to recursive data type definitions
a means to understand induction in a more general framework than just for natural numbers

13. **Operational Semantics**
   Functional programs execute by rewriting.
   calculus as a rewriting system
   Reduction, confluence, reasons for preferring normal order reduction.

14. **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:

<table>
<thead>
<tr>
<th>Worlds</th>
<th>The Timeless World</th>
<th>World of Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Mathematics</td>
<td>Programming</td>
</tr>
<tr>
<td>Syntax</td>
<td>Expressions</td>
<td>Statements</td>
</tr>
<tr>
<td>Semantics</td>
<td>Values</td>
<td>Objects</td>
</tr>
<tr>
<td>Explicit</td>
<td>Data Structures</td>
<td>Control Structure</td>
</tr>
<tr>
<td>Think with</td>
<td>Input Output relations</td>
<td>State Change</td>
</tr>
<tr>
<td>Abstractions</td>
<td>Functions</td>
<td>Procedures</td>
</tr>
<tr>
<td>Relation</td>
<td>Denote programs</td>
<td>Implement functions</td>
</tr>
</tbody>
</table>

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.

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

16. **Operational Thinking**
   IN FP data dependency implicitly determines sequencing whereas in Imp P it is done explicitly. Advantages and disadvantages of operational thinking.

17. **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.

18. **Semi Explicit Continuation**
   Explicit in the sense that goto labels can be dealt with first-classly (as in assembly), but not explicit in the sense of capturing the entire future of a computation dynamic execution of a code block may be 'concave'.

19. **Recursion iteration equivalence**
   General principles as well as scheme semantics of tail recursion.

20. **Type Issues**
   Monomorphic, polymorphic and latent typing: translating one into another.

21. **Guile**
   A variety of vehicles have been used for the imperative paradigm, e.g. Pascal, C, Java, Tcl. The current choice is Scheme in the guile dialect because it gives a full support for the functional and the imperative paradigm. In fact Guile has been chosen over C
because the single data structure in guile expressions is universal (aka XML) and thus
imperative and functional thinking do not quarrel with data structure issues.

Orthogonal kinds of abstractions, which are usually considered 'advanced', such as
functional, higher order functional, object-oriented, stream based, data driven,
language extensions via eval, via macros, via C can be easily demonstrated. In fact,
once guile has been learnt, it is much faster to pick up C in the subsequent semester.

**Note:** In addition to being a system programming and general purpose language Guile
is also a scripting, extension and database programming language because it is the
flagship language for FSF (The free software foundation).

• **References:**
  
  Introduction to Functional Programming, Bird and Wadler, Prentice Hall
  Algebra of Programs, Bird, Prentice Hall
  Structure and Interpretation of Computer Programs, Abelson and Sussman, MIT Press
  Scheme and the Art of Programming, Friedmann and Haynes, MIT Press
  Equations Models and Programs., Thomas Myers, Prentice Hall
  Algorithms + Data Structures = Programs, N Wirth
  Functional Programming, Reade
  Programming from First Principles, Bornat, Prentice Hall
  Discrete Math with a computer, Hall and Donnell, Springer Verlag
CS-102 Computer Organization

- **Contents**:
  1. **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.
  2. **Parts of a computer**:
     Processor (CPU), memory subsystem, peripheral subsystem. The memory interface: memory subsystem minus the actual memory. Ditto with the peripheral interface. 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) An important aim of the CO course is to understand the internals of these parts, and the interactions between them.
  3. **Instruction set formats**:
     Three-address and one-address instructions and the corresponding data-path architectures, namely, general-purpose register architecture (the classic RISC) and accumulator architecture. Zero-address instructions and the stack architecture. Two-address instructions, e.g., in the Pentium.
  4. **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 MIPS R2000, arguably the classic RISC machine,
  5. **Basic Electronics**:
     Just those concepts needed to understand CO: 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.
  6. **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, redundant arrays of independent disks (RAID).
  7. **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.
  8. **Peripherals**:
     Interconnecting peripherals with memory and processor.

**References:**
Computer Organization and Design, Patterson and Hennessey
Computer Structures, Ward and Halstead
Digital Design: Principles and Practices, Wakerley
1. **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.

2. Self-reference, paradoxes, Gödel's theorem Alternative Logics e.g. modal, dynamic, intuitionistic, situational Applications: Prolog, Program Verification

3. **Binding Constructs:** Abstraction of lambda, for all, program function etc. Free and bound variables, substitution. Common laws.

4. **Set Theory:** Definitions, proofs, notations, building models Applications: Z, Abrial's machines

5. **Well formed formulae:** Ordinary definition, refinement to types, necessity and limitation of computable type checking.

6. **Category Theory:** Problems with Set theory constructive, conceptual and type and their categorical solution Applications: functional programming equivalents of categorical results

7. **Relations:** 3 alternative views of foundations of relations: as Cartesian products, as Boolean functions (predicates), as power set functions 3 basic types - equivalences, orders, functions - properties and applications in databases

8. **Calculus (Closely integrated with IP)** Explicit and Implicit definitions. The 3 ingredients of function definition: naming, abstraction/quantification, property/predicate. Mathematically - separates the 3 Computationally - delays by transforming computation into recopies Philosophically - enriches the programmer's world by moving programs from syntax to first-class semantics

9. **Algebraic Structures:** Development: Logic, Set Theory, Cartesian Products, Relations, Functions, Groupoids, Groups, Many sorted Algebras, Lattice Theory Applications to cryptography, denotational semantics, cryptography

**References:**
Logic for CS by Gallier
Discrete Math by Tremblay Manohar
Discrete Math by Stanat
Laws of Logical Calculi by Morgan
Category Theory tutorial by Hoare
Category Theory by Burstall and Rydeheard
Computer modeling of mathematical reasoning by Bundy
Shape of mathematical reasoning by Gasteren
Predicate Calculus and Program Semantics by Dijkstra
Algebra of Programming by Richard Bird
A Gentle Introduction to Category Theory the calculational approach by Fokkinga http://wwwhome.cs.utwente.nl/~fokkinga/#mmf92b
A Logical Approach to Discrete Math by Gries and Schneider
Practical Foundations of Mathematics by Paul Taylor
Conceptual Mathematics by Lawvere
Practical Foundations of Mathematics by Taylor
Internal Documents of R.P.Mody on notation, style, combination
CS-104 Concrete Math and Graph Theory

- **Contents**
  - **Graph Theory**
    1. **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, Shortest Path Algorithm, Fleury's Algorithm
       - Chinese Postman problem, Traveling Salesman problem
    2. **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
    3. **Planar Graphs**
       - Combinational and geometric duals
       - Kuratowski's graphs
       - Detection of planarity, Thickness and crossings
    4. **Matrix Representation of Graphs**
    5. **Coloring**
      - Chromatic Number, Chromatic Polynomial, the six and five color theorems, the four color theorem
    6. **Directed Graphs**
    7. **Enumeration of Graphs**
      - Counting of labeled and unlabeled trees, Polya's theorem, Graph enumeration with Polya's theorem
  - **Concrete Mathematics**
    8. **Sums**
      - Sums and recurrences, Manipulation of sums, Multiple Sums, General methods of summation
    9. **Integer Functions**
      - Floors and ceilings, Floor/Ceiling applications, Floor/Ceiling recurrences, Floor/Ceiling sum
    10. **Binomial Coefficients**
       - Basic Identities, Applications, Generating functions for binomial coefficients
    11. **Generating Functions**
       - Basic maneuvers, Solving recurrences, Convolutions, Exponential generating functions
    12. **Asymptotics**
       - O notation, O manipulation, Bootstrapping, Trading tails
  - **References**
    - Graph Theory with Applications, Bondy, J. A. & U. S. R. Murty [1976], MacMillan
    - Graph Theory with Applications to Engineering and Computer Science, Deo, Narsing [1974], Prentice Hall
    - Concrete Mathematics, A Foundation for Computer Science, Graham, R. M., D. E., Knuth & O. Patashnik [1989], Addison Wesley
    - Graph, Networks and Algorithms, Swamy, M. N. S. & K. Tulsiram [1981], John Wiley
CS-105 Database Management System

Contents:
1. DBMS objectives and architectures
2. Data Models
   - Conceptual model, ER model, object oriented model, UML Logical data model,
     Relational, object oriented, object relational
3. 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.
4. Relational database design
   - Schema design, Normalization theory, functional dependencies, higher normal forms,
     integrity rules, Relational operators
5. 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
6. Mapping mechanism
   - Conceptual to logical schema, Key issues related to for physical schema mapping
7. DBMS concepts
   - ACID Property, Concurrency control, Recovery mechanisms, case study Integrity,
     Views & Security, Integrity constraints, views management, data security
8. Query processing, Query optimization -
   - heuristic and rule based optimizers, cost estimates, Transaction Management
9. Case Study
   - ORACLE/POSTGRES DBMS package: understanding the transaction processing
     Concurrency and recovery protocols, query processing and optimization mechanisms
     through appropriate queries in SQL and PLSQL.
10. Web based data model -
    - XML, DTD, query languages
11. 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

References:
Database System Concepts, Silberschatz, Korth and Sudershan, McGraw Hill
Relational Database Index Design and the Optimizers by Tapio Lahdenm¨aki Michael
Leach, John Wiley
PostgreSQL, Sams Publications
Science Press, 1998
CS-201 Numerical Methods

- **Contents:**
  1. Introduction to Complex Variable theory
  2. Matrix Algebra
  4. Solutions of Systems of Nonlinear Equations
  8. Propagation of roundoff error
  9. Interpolation and approximation
     Interpolating Polynomials, Existence, Error and Convergence of Interpolating, Polynomial Construction of Interpolating Polynomials from ordinates and by using differences.

- **Notes:**
  The course will start by teaching Complex Variable Theory and asking the students to read the Matrix Algebra by themselves. This will be followed by a test of these topics. The remaining topics will now be covered more or less in the same order as listed in the syllabus.

- **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-202 Data and File Structures

- **Prerequisite:** (Student should have undergone the prerequisite course) CS101 (Introduction to Programming)
- **Course Overview**

<table>
<thead>
<tr>
<th>Algebraic view</th>
<th>Algorithmic view</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td></td>
</tr>
<tr>
<td>Data Structures, Mathematical Definitions, Laws, Manipulations,MF relations</td>
<td>Storage Structures, Engineering Considerations related to CO, LLP</td>
</tr>
<tr>
<td><strong>Code</strong></td>
<td></td>
</tr>
<tr>
<td>Recursive and closed form program specification. May be implementable in a high level language like gofer or may not be implementable directly. The intrinsic value of specification apart from programs.</td>
<td>Explicit control through built in control structures like sequencing, if, while Engineering efficient implementation of correct specifications</td>
</tr>
</tbody>
</table>

- **Contents**
The course is organized according to the philosophy in the table below. The case studies/examples include but need not be limited to
  1. **Lists**: Various types of representations. Applications: symbol tables, polynomials, OS task queues etc
  2. **Trees**: Search, Balanced, Red Black, Expression, and Hash Tables Applications: Parsers and Parser generators, interpreters, syntax extenders
  3. **Disciplines**: Stack, queue etc and uses
  4. **Sorting and Searching**: Specification and multiple refinements to alternative algorithms
  5. **Polymorphic structures**: Implementations (links with PP course)
  6. **Complexity**: Space time complexity corresponds to element reduction counts. Solving simple recurrences.

- **Course Organization**

<table>
<thead>
<tr>
<th>Correctness</th>
<th>Algebraic world</th>
<th>Algorithmic world</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bird Laws, Category Theory</td>
<td>Refinement, Predicates</td>
</tr>
<tr>
<td>Transformation</td>
<td>Via Morgan Refinement</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADTs and Views</th>
<th>Algebraic world</th>
<th>Algorithmic world</th>
</tr>
</thead>
<tbody>
<tr>
<td>o Formulation as recursive data types</td>
<td>o C storage</td>
<td></td>
</tr>
<tr>
<td>o Data structure invariants</td>
<td>o Representation Invariants</td>
<td></td>
</tr>
<tr>
<td>o Principles of interface design</td>
<td>o Addressing Semantics</td>
<td></td>
</tr>
<tr>
<td>o Algebraic Laws</td>
<td>o Use of struct, union and other assorted C stuff</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o Maximizing abstraction</td>
<td></td>
</tr>
<tr>
<td>Mapping</td>
<td>Via transforms and coupling invariants</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Code</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Pattern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matching based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>recursive definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Exhaustive set</td>
<td></td>
<td></td>
</tr>
<tr>
<td>of disjoint patterns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>correspond to total functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Correspond to runtime bug free programs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Recursive Code structures follow from recursive data structures</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Refinement of recursive definitions into iterative algorithms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Techniques (Bentley) for improving algorithms e.g. sentinel, double pointers, loop condition reduction, strength reduction etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Continuations</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Control as Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Co routines vs. subroutines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o General framework for escape procedures, error handling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Loops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Functions @</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Stack based software architecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Error Policy Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Patterns</td>
<td>Predicate Transformer Semantics for control</td>
<td></td>
</tr>
<tr>
<td>o Laws</td>
<td></td>
<td></td>
</tr>
<tr>
<td>o Deliberate Partiality</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Modules</strong></td>
<td>Category Theory</td>
<td></td>
</tr>
<tr>
<td>Files, make</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **References:**
  Data Structures and Algorithms, Aho, Hopcroft and Ullman, Addison Wesley Inc.
  Data Structures, Kruse, Prentice Hall
  Programming from Specifications, Carroll Morgan, Prentice Hall
  Algebra of Programs, Bird, Prentice Hall
  Programming Perls, Writing Efficient Programs, John Bentley, Prentice Hall
  Structure and Interpretation of Computer Programs, Abelson Sussmann, MIT Press
  Functional Programming Henderson, Prentice Hall
CS-203 Low-Level Programming

- Contents
  1. C Language Basics
  2. Assembly Language structure, syntax, macros
  3. Use of linker, librarian, object editor(s), debugger
  5. 8086 architecture going up to P4. Survey of Intel architecture history
  6. Inline Assembly, Floating point operations
  8. I/O Classification: Memory mapped vs. IP mapped. Polled, Interrupt, DMA
  9. Interrupts: h/w and s/w. ISRs. Assembly and C. Minimization and handling of nondeterminism Separation of binding times: Hard-coding of chip, board, OS, system s/w, user levels
  10. OS use: system call interface
  11. OS implementation: Start up scripts, Basics of protected mode and device drivers
  12. Chip Level Programming

References
Professional Assembly Language, Richard Blum, Wrox
Guide to Assembly Language Programming, S P Dandamudi, Springer
Linux Device Drivers, 3rd Edition By Rubini, Orielly
Art of Assembly, Randy Hyde
Intel Manuals
OS, chip manuals
Compiler and System S/w manuals
C Programming, Kernighan and Ritchie
CS-204 Operating Systems

• **Contents:**
  1. Simple computer systems made up of a single processor and single core memory spaces and their management strategies.
  2. Processes as programs with interpolation environments. Multiprocessing without and with IPC. Synchronization problems and their solutions for simple computer systems.
  6. Illustrative implementation of
     - bootstrap code,
     - file systems,
     - memory management policies etc.

• **References**
  A. S. Tanenbaum, Modern Operating Systems, Pearson Education
  Galvin, Operating Systems Concepts, Wiley
  Nutt, Operating System, Pearson Education
  A. S. Tanenbaum, Distributed Operating Systems, Prentice Hall
  Understanding the Linux Kernel, 2nd Edition By Daniel P. Bovet, Oreilly
  The Design of Unix Operating System Maurice Bach, Pearson
CS-205 Science Of Programming

- Contents:
  1. Verification: verification of imperative programs as in Gries/Dijkstra.
  2. Specific techniques: Invariant assertive method, sub-goal induction method.
  3. Verification of pointer programs.
  6. Clear as an example of a model axiomatic/categoric language.
  7. Transformation/Refinement
  8. Homomorphic transformations, refinement Calculus Theory & application of List/Functional
  9. Calculus
  10. Theory Logics of Programs
  11. Hoare Logics, Dynamic Logic
  12. Temporal Logic Application to OOP

References:
Functional Programming, Henson, Blackwell scientific
Science of Programming, Gries, Narosa
Discipline of Programming, Dijkstra, Prentice Hall
Method of Programming, Dijkstra & Feijen, Addison Wesley
Specification Case Studies, Hayes, Prentice Hall
Software Specification, Gehani & Mcgettrick, Addison Wesley
Program Specifications & Transformations, Meertens, Prentice Hall
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
CS-301 Design and Analysis of Algorithms

- **Contents:**
  1. String processing
  3. Graph and geometric Algorithms
  4. DFS, BFS, Biconnectivity, all pairs shortest paths, strongly connected components, network flow
  5. Ford-Fulkerson Algorithm, MPN Algorithm, Karzanov Algorithm, Maximum Matching in bipartite graphs
  6. Geometric Algorithms
  7. Backtracking, Dynamic Programming, Branch & Bound, Greedy
  8. Use of three paradigms for the solution of problems like Knapsack problem, Traveling Salesman etc.
  9. Lower Bound Theory
  10. Sorting, Searching, Selection

- **References:**
  Algorithm Design: Foundations, Analysis, and Internet Examples Michael T. Goodrich, Wiley
  Computer Algorithms, Sara Baase, Addison Wesley, 1988
  Combinational Algorithms, T. C. Hu, Addison Wesley, 1982
CS - 302 Theoretical Computer Science

- **Prerequisite:** (Student should have undergone the prerequisite course) CS-103 (Mathematical Foundation)
- **Contents**
  1. Low Power Formalisms Combinational Machines inadequacy
  2. FSM as acceptor, generator, regular expressions and equivalence
  3. PDA brief idea, relation between CFG's and programming languages (informal)
  4. 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 'non-Von-Neumann' language.
  5. **Self References**: Use mention distinctions, 'escape methods' for self referencing quines, self references in the expression domain, the formulation of the 'halting problem' and decidability in C and Scheme
  6. **Recursive Data**: Recursive, Enumerable sets, generators and recognizers formulated as recursive types in Haskell, 'S' expressions in Scheme.
  7. Complexity Basic ideas measuring time usage, time hierarchies
  8. Deterministic and Nondeterministic computations.
  10. Universality
  11. Equations in language spaces
     Operational approach
     Denotational approach

**References:**
Introduction to the theory of computation, Sipser, Thompson Learning
Introduction to Computer Theory, Cohen, Wiley
Computabilities and complexity from a programming perspective, Niel Jones, MIT Press
The Quine page, Gary P. Thompson, at http://www.myx.net/~gthompson/quine.htm
Computation and Automata, Salomaa, CUP
Switching and finite Automata Theory, Kohavi, ZVI, Tata McGrawHill
Finite and Infinite Machines, Minsky, Prentice Hall
Post Systems, Krishnamurthi E. V.
Godel, Escher, Bach, Hoffstader, Vintage Books
Introduction to Recursive Function theory, Cutland, CUP
Handbook of TCS Vol A,B, Jan Van Leeuven ed, Elsevier
CS-303 Computer Networks

- **Contents**:
  1. Network architecture, ISO-OSI Reference model
  2. Network Topology:
  3. Topology design problem, connectivity analysis, delay analysis, backbone design, and local access network design.
  4. Physical Layer, Transmission media, digital transmission, transmission & switching,
  5. Integrated Services Digital Network.
  6. Data Link Layer: Design issues, protocols, CRC
  7. Network Layer: Design issues, routing algorithm, congestion control, Packet switched networks,
  8. X.25 standards, ATM networks
  9. Transport Layer: TCP, UDP, Design issues
  10. Session Layer: Design issues, client server model, remote procedure calls
  11. Local Area Networks, IEEE 802 standards for LAN (Ethernet, token ring, optical fiber, wireless)
  12. Application layer environment
  13. Application layer architecture, building applications with sockets, DNS, HTTP, SMTP, LDAP, NFS, NIS, SNMP, WAP Mobile computing
  14. Internet, extranet, Virtual Private Network (includes tunneling, internet work routing and fragmentation)
  15. Internet Security: Firewalls, SSL, Popular encryption protocols

- **References**:
  Computer networks: A systems approach, 2nd Edn., Peterson and Davie, Morgan Kaufman
  Computer Networks, 4th Edn., A. S. Tanenbaum, Pearson Education
  UNIX Network Programming Volume 1 Stevens, Adison Wesley 2003
CS-304 Systems Programming

**Contents:**

1. 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).

2. 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

3. **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).
   - 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. Source files related system software. Syntax manipulation (lex and yacc). Editors, version controllers. Version control on object and executable files (e.g. version support for modules in the Linux kernel).

4. **Support tools:**
   - Literate programming (weave, tangle), source browsers, documentation generators, make, GNU auto-conf, CVS, bug reporting systems. IDEs for systematic use of system tools. Flow graphers, Debuggers for analysis. Package builders, installers, package managers for deployment

5. The notion of binding time as instant of achieving the mapping between a symbol and a value. Overlays and remote procedure call as memory space influenced between symbol and value.

**References:**

Hopcroft, Sethi and Ullman, Compiler Principles, Addison Wesley


System Software: An Introduction to Systems Programming, Leland L. Beck Pearson Education

H. Abelson and G. Sussmann, Structure and Interpretation of Computer Programs (SICP), MIT Press

Hopcroft and Ullman, Introduction to Automata theory, Languages and Computation, Narosa Publishing

The details of the Pentium can be found in various manuals at ftp://developer.intel.com/design/Pentium4/manuals/


System Programming Guide: 24547212.pdf
Contents
• What is distributed computing?
• Why distributed computing?
• Concepts of time, logical and physical clocks
• Concurrency: including multithreading
  o barriers, locks, spinlocks, how and why
• Basics of communication
• Inter Process Communication: RPC, message passing, client-server systems ...
• Stateless and stateful C-S systems
• Transactions
• Web services
• Why do systems fail, and reliability issues
• High availability and scalability
• Membership services and group comm. protocols
• P2P systems
• Distributed Applications
• All the above is to be conveyed using the contemporary technology. Suggested technologies
  for the current period are LAMP, J2EE or .NET stacks

References:
George Coulouris, Jean Dollimore and Tim Kindberg, Distributed Systems: Concepts and Design, Addison-Wesley
Jie Wu, Scalable Computing: Practice and Experience, CRC Press
Gerard Tel, Introduction to Distributed Algorithms, Cambridge University Press
Nicolai Josuttis, SOA in Practice: The Art of Distributed System Design (In Practice), O'Reilly
CS-401 Computer Graphics

- **Contents:**
  2. Raster Graphics Features, raster algorithms including primitives like lines, circles, filling, clipping in 2D, etc.
  3. Geometric transformations in 2D for 2D object manipulation, coordinate transformations and their matrix representation, Postscript language to demonstrate these concepts.
  4. The 3rd dimension, it’s necessity and utility, transformations and modeling in 3D, geometric modeling with an introduction to curves and surfaces for geometric design, including but not restricted to Bezier, B’spline, Hermite representations of curves and surfaces.
  6. Rendering Techniques for Line Drawings, Rendering Techniques for Shaded Images, Aliasing and Anti-aliasing, Illumination Models local models like Phong, CookTorrance and global models like ray tracing and radiosity, shading detail like textures, their generation and mapping, bump mapping and similar techniques.
  7. Depending on time availability, one of volume rendering, modeling of natural objects, introduction to 3D animation may be covered depending on student and instructor inclination.

- **References:**
  The Rendering Equation, J. Kajiya, SIGGRAPH 1986, 143’150
CS-402 Modeling and Simulation

- **Contents**:
  1. Introduction to Systems modeling concepts, continuous and discrete formalisms
  2. Framework for Simulation and Modeling, modeling formalisms and their simulators, discrete time, continuous time, discrete event, process based.
  3. Hybrid systems and their simulators
  4. Review of basic probability, probability distributions, estimation, testing of hypotheses
  5. Selecting input probability distributions, models of arrival processes
  6. Random number generators, their evaluation, generating random variates from various distributions.
  7. Output analysis, transient behavior, steady state behavior of stochastic systems, computing alternative systems, variance reduction techniques.
  8. Verification and Validation

- **References**:
  Object Oriented Simulation with Hierarchical Modular Models, B. Zeigler, Academic Press, 1990
CS-403 Operations Research

- Contents:
  1. The nature of O.R., History, Meaning, Models, Principles Problem solving with mathematical models, optimization and the OR process, descriptive vs. simulation, exact vs. heuristic techniques, deterministic vs. stochastic models.
  3. Introduction to game theory
  4. Multi objective optimization and goal programming
  5. Shortest paths, CPM project scheduling, longest path, dynamic programming models
  6. Discrete optimization models: integer programming, assignment and matching problems, facility location and network design models, scheduling and sequencing models
  7. Nonlinear programming: unconstrained and constrained, gradient search, Newton's method,
  8. Nelder-Mead technique, KuhnTucker optimality conditions. These topics should only be covered only time permits.
  9. Discrete Time processes: Introduction, Formal definitions, Steady state probabilities, first passage and first return probabilities, Classification terminology, Transient processes, queuing theory introduction, terminology and results for the most tractable models like M/M/1
  10. Inventory Models ( Deterministic): Introduction, The classical EOQ, sensitivity analysis, Nonzero lead time, EOQ with shortages, Production of lot size model, EOQ with quantity discounts, EOQ with discounts, Inventory models ( Probabilistic): The newsboy problem : a single period model, a lot size reorder point model

- References:
  Linear Programming and Extensions, G Dantzig, Princeton University Press, 1963
CS-404 Software Engineering – I

Contents:

1. Introduction, Need, Software life cycles
2. Overview of Requirements Engineering, Processes, the requirements document
3. System Specification
   - Logic Sets and Types, Z specification structure
   - Relations, Functions, Sequences
4. Structured System Analysis Design
   - ER Diagrams, Data Flow Diagrams
5. Object Oriented Software Design using UML
6. Notations for Design
   - A brief reintroduction to Object Oriented Concepts and an overview of the UML notation
   - Characteristics of notations for design.
7. Requirements Analysis
   - User Requirements Gathering, Performing a Domain Analysis, Developing the Use Cases.
8. System Specification
   - Design and Analysis using UML
   - Class Diagrams
   - UML Activity Diagrams, Task Analysis
   - UML Interaction Diagrams
   - UML Object Diagrams
   - UML Deployment Diagrams, Collaboration diagrams, Data Flow Diagrams
9. SSAD Vs Object Oriented Design
10. CASE Tools
11. Forward Engineering and Reverse Engineering
12. Code Construction
   - UML to Code, Code to UML
   - Z to Code

References:

- Software Engineering A Beginner’s Approach, Roger S. Pressman, McGraw Hill
- The Engineering of Software, Dick Hamlet, Joe Maybee, Addison Wesley, 2001
- UML Distilled, 2nd Ed., Martin Fowler, Addison Wesley
- Using UML for Software Engineering, Pooley and Stevens, Addison Wesley, 1999
- Software Engineering Peters, Wiley India
- Specification Case Study, Hayes, Prentice Hall
CS-601 Programming Paradigms

**Contents**

1. GUI Programming
2. GUI Vs CUI
3. Event Driven Programming
4. Visual (Meta-GUI) Programming
5. Architecture of typical Application
6. VB Environment : Steps in creating and using controls
7. Database Connectivity, codeless programming
8. OO Paradigm
9. Modularity
10. Data Abstraction
11. Classes and Objects
12. Inheritance and interfaces
13. Polymorphism
14. Inner Classes
15. Use of AWT and Swing for GUIs
16. Applets (if time permits)
17. UML: Class Diagrams, Sequence Diagrams
18. UML to Java tools (ArgoUML)
19. HDL via Verilog or VHDL
20. Architectural behavioral and RT levels
21. Study of Waveforms
22. Differences between features used for testing and allowable in design
23. Notion of Scripting
24. Scripting via Perl/Guile/Python

**References:**

- Verilog HDL by S. Palnitker (Prentice Hall)
- Perl by Wall and Christiansen (O'reilly)
- Core Java 2 Vol I fundamentals and Vol II Advanced features by Cay S. Horstmann and Gery Cornell (Prentice Hall)
- Thinking in Java Vol 3 by Bruce Eckel at http://www.mindview.net/books/TIJ
- Scripting reference at http://home.pacbell.net/ouster/scripting.html
- Guile for scripting at http://gnuwww.epfl.ch/software/guile/guile.html
- The art of programming with Visual Basic by Mark Warhol (John Wiley & Sons)
- Visual Basic 6.0 programmer's guide (Microsoft Press)
- Visual Basic 6.0 database programming bible by Wayne Freeze (Hungry Minds)
- Dive into Python by Mark Pilgrim at http://diveintopython.org
- Programming Python by Mark Lutz, 2nd Edition (O'Reilly)
- Python Documentation at http://www.python.org/doc/
CS-602 Software Engineering - II

- **Prerequisites**: (Student should have undergone the prerequisite course) CS-404 (Software Engineering – I)
- **Contents**:
  1. Concepts of software management, The software crisis, principles of software engineering, programming in the small Vs programming in the large
  2. Software methodologies/processes, The software life cycle, the waterfall model and variations, introduction to evolutionary and prototyping approaches
  3. Software measurement
  4. Object-oriented requirements analysis and modeling: Requirements analysis, requirements
  5. Solicitation, analysis tools, requirements definition, requirements specification, static and dynamic specifications, requirements review. (just revisited)
  6. Software architecture
  7. 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
  9. User considerations, Human factors, usability, internationalization, user interface, documentation, user manuals
  10. Documentation, Documentation formats, tools
  11. Project management, Relationship to life cycle, project planning, project control, project organization, risk management, cost models, configuration management, version control, quality assurance, metrics
  12. Safety
  13. Maintenance, The maintenance problem, the nature of maintenance, planning for maintenance
  14. Configuration Management
  15. Tools and environments for software engineering, role of programming paradigms, process maturity
  16. Introduction to Capability Maturity Model
     - People Capability Maturity Model
     - Software Acquisition Capability Maturity Model
     - Systems Engineering Capability Maturity Model
  17. IEEE software engineering standards
- **References**:
  Software Engineering, 6th Edn., Ian Sommerville, Addison Wesley, 2001
  (Note : This is also the preferred textbook for the IEEE Software Engineering Certificate Program.)
  The Engineering of Software, Dick Hamlet, Joe Maybee, Addison Wesley, 2001
  Introduction to the Team Software Process, Watts S. Humphrey, Addison Wesley, 2000
  Practical Software measurement, Bob Hughes, McGraw Hill, 2000
  Human Computer Interaction, 2nd Edn., Dix, Finlay, Abowd and Beale, Prentice Hall, 1997
CS-603 Applications of Software Engineering and Programming Paradigms

- Contents:
  - Comparison between formal and informal ways of modeling software
  - Modeling a given software system using Z-specification
  - Modeling a given software system using UML
  - Study of other ways of specification and modeling
  - Study of Software Quality
    - CMM practices and CMM levels
    - Six Sigma practices
  - Study of Software Processes (e.g. Rational Unified Process)
  - Implementation of example software systems using different programming paradigms
  - Views of a software system from different paradigms
  - Comparative study of application of different programming paradigms to software development
  - Implementation of a typical software in order to appreciate advantages, disadvantages and limitations of different programming paradigms
  - Appropriateness of particular paradigm for a given kind of software
  - Using Python as multi-paradigm programming language
  - Implementation of higher order functions in non-functional languages
  - Implementation issues of event driven software systems (e.g. X Window System, VB software)

- References:

  Using UML for Software Engineering, Pooley and Stevens, Addison Wesley, 1999
  Rational Unified Process, [www.rational.com](http://www.rational.com)
  Practical Software measurement, Bob Hughes, McGraw Hill, 2000
  Thinking in Java Vol 3 by Bruce Eckel at [http://www.mindview.net/books/TIJ](http://www.mindview.net/books/TIJ)
  Thinking in C++ by Bruce Eckel
  Visual Basic 6.0 programmer’s guide (Microsoft Press)
  X Window System Documentation, [www.xfree86.org](http://www.xfree86.org)
  Boost Lambda Library for C++, [www.boost.org](http://www.boost.org)
CS-411 Software Engineering (M.Sc./M.Tech. only)

Contents:

1. Introduction, Need, Software life cycles
2. Overview of Requirements Engineering, Processes, the requirements document
3. System Specification
   Logic Sets and Types, Z specification structure
   Relations, Functions, Sequences
4. Structured System Analysis Design
   ER Diagrams, Data Flow Diagrams
5. Object Oriented Software Design using UML
6. Forward Engineering and Reverse Engineering
7. Code Construction
   UML to Code, Code to UML
   Z to Code
8. Concepts of software management, The software crisis, principles of software engineering, programming in the small Vs programming in the large
9. Software methodologies/processes, The software life cycle, the waterfall model and variations, introduction to evolutionary and prototyping approaches
10. Software measurement
11. Software architecture
12. Software design, Design for reuse, design for change, design notations, design evaluation and validation
13. Implementation, Programming standards and procedures, modularity, data abstraction, static analysis, unit testing, integration testing, regression testing, tools for testing, fault tolerance
14. User considerations, Human factors, usability, internationalization, user interface, documentation, user manuals
15. Documentation, Documentation formats, tools
16. Project management, Relationship to life cycle, project planning, project control, project organization, risk management, cost models, configuration management, version control, quality assurance, metrics
17. Maintenance, The maintenance problem, the nature of maintenance, planning for maintenance

• References :
   Software Engineering A Beginner's Approach, Roger S. Pressman, McGraw Hill
   Software Engineering, 6th Edn., Ian Sommerville, Addison Wesley, 2001
   The Engineering of Software, Dick Hamlet, Joe Maybee, Addison Wesley, 2001
   UML Distilled, 2nd Ed., Martin Fowler, Addison Wesley
   Using UML for Software Engineering, Pooley and Stevens, Addison Wesley, 1999
   The Unified Modeling Language Users Guide, 1st Ed., Grady Booch, James Rumbaugh and Ivar
   Introduction to the Team Software Process, Watts S. Humphrey, Addison Wesley, 2000
   Practical Software measurement, Bob Hughes, McGraw Hill, 2000
   Human Computer Interaction, 2nd Edn., Dix, Finlay, Abowd and Beale, Prentice Hall, 1997