Home > **Academic**

Academic Programs and Courses

Admissions to MCA & MSc. Programmes is through the common entrance test conducted by the Computer Science Department, University of Pune.

Admissions to MTech Programme is through GATE Score and Advertisement.

Degree Programmes

- MCA - 3 years
  The MCA degree primarily aims at training for professional practice in the industry. The programme is designed so that the graduate can adapt to any specific need with ease. The duration of the study is six semesters, which is normally completed in three years. Selection is through the Qualifying Exam and satisfying the eligibility criteria.

- MSc - 2 years
  The MSc degree prepares the student for higher studies in Computer Science. The duration of the study is four semesters, which is normally completed in two years. An year long project provides an opportunity to apply the principles to a significant problem. Selection is through the Qualifying Exam and satisfying the eligibility criteria.

- MTech - 2 years
  The MTech degree is a first level degree in Computer Science for graduates in any engineering discipline except Computer Science. This programme also primarily aims at training for professional practice in the industry. The programme is designed so that the graduate can adapt to any specific need with ease. The duration of the study is four semesters, which is normally completed in two years. An year long project provides an opportunity to apply the principles to a significant problem. Selection is through the Qualifying Exam and satisfying the eligibility criteria.

- **Eligibility :**
  GATE score in Engineering or any Mathematical or Physical Sciences or UGC/CSIR JRF qualification, valid in July of year of entrance exam.

NOTE:

- For information concerning GATE, contact the GATE office at any Indian Institute of Technology.

- Candidates qualifying GATE in Computer Science: please note that our M.Tech. programme is a first-level programme in Computer Science.

- Foreign nationals studying in Indian Universities will be judged by the same criteria as those applied to Indian nationals. In particular, they have to appear for the Entrance Exam. Additional Requirements for Reserved Categories

- Candidates belonging to the following categories are required to submit the following documents at the time of admission.

  Physically handicapped:
  A medical certificate from a registered physician. The handicapped status will be verified by a physician approved by the University of Pune.

  Kashmir Quota:
  Letter from Directorate of Higher and Technical Education, Government of Maharashtra.

  SC/ST:
  Attested copy of caste certicate.

  DT/NT/OBC:
  Attested photocopy of caste certificate issued by Govt. of Maharashtra, and creamy layer free certificate if applicant claims reservation under NT(C), NT(D) and OBC.

If selected candidates cannot submit these documents, their admission will be cancelled. Candidates of reserved categories recognised by states other than Maharashtra will not be considered for these reserved seats.

Common Courses (First two semesters)

- **Semester 1**

  CS-101 Introduction to Programming
  CS-102 Logical Organization of Computers
  CS-103 Mathematical Foundations
  CS-104 Concrete Maths & Graph Theory
  CS-105 Numerical Methods

- **Semester 2**

  CS-201 Data Structures & Algorithms
  CS-202 Theoretical Computer Science
  CS-203 Low-level Programming
  CS-205 Computer Architecture and Operating Systems
  CS-206 Programming Paradigms
  Courses Specific to M.C.A. (Last four semesters)

- **Semester 3**

CS-204 Design & Analysis of Algorithms
CS-301 Database Management System
CS-302 Computer Networks
CS-303 Systems Programming

- **Semester 4**

  CS-305 Computer Graphics
  CS-401 Modelling and Simulation
  CS-402 Operations Research
  CS-403 Systems Analysis and Design
  Elective-1

- **Semester 5**

  Full-time Industrial Training

- **Semester 6**

  CS-304 Science of Programming
  CS-601 Software Engineering
  Elective-2

Courses Specific to M. Sc. (Last two semesters)

- **Semester 3**

  CS-204 Design & Analysis of Algorithms
  CS-301 Database Management System
  CS-302 Computer Networks
  CS-303 Systems Programming
  CS-MSP Degree Project I

- **Semester 4**

  CS-304 Science of Programming
  CS-601 Software Engineering
  CS-MSP Degree Project II
  Elective-1

Courses Specific to M.Tech. (Last two semesters)

- **Semester 3**

  CS-204 Design & Analysis of Algorithms
  CS-301 Database Management System
  CS-302 Computer Networks
  CS-303 Systems Programming
  CS-MTP Degree Project I

- **Semester 4**

  CS-304 Science of Programming
  CS-601 Software Engineering
  CS-MTP Degree Project II
  Elective-1

Elective Courses (offered in the last few years)

- Advanced Computer Architecture
- Advanced Theoretical
- Computer Science
- Advanced Topics in DBMS
- Artificial Intelligence and Tools
- Category Theory
- Code Optimization
- Compiler Construction
- Data Warehousing and Mining
- Discrete Optimization
- Implementation of RDBMS
- Geometric Modelling
- Issues in Programming
- Logic Programming
- Parallel Algorithms
- Parallel Architectures

- Programming Languages:Theory and Implementation
- Soft Computing
- Software Tools
- Spatial Information Systems
- System Management and Modeling
- User Interface Design

CS-101 - Introduction to Programming

- **Aims and Objectives**
  To give students the grounding that makes it possible to approach problems and solve them on the computer.

- **The aspects covered range across:**

  1. Modelling a given problem domain appropriately
  2. Designing a solution
  3. Implementing the solution in a high level programming language

- **Contents**
  Two paradigms are used as vehicles to carry the ideas and execute practicals 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 highlevel tool for problem solving. The paradigm conveyed may be simply expressed as:

  A modern nonstrict 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, tuples. (MF)
  3. **calculus**
     A direct way for denoting functions.
  4. **First Classness**
     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 dataï¿½structures 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 admissable 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:

Worlds	**The Timeless World**	**World of Time**
Domain	Mathematics	Programming

Syntax	Expressions	Statements
Semantics	Values	Objects
Explicit	Data Structures	Control Structure
Think 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.

1. **Semantic relations** The central relation is that imperative programming's denotational semantics is FP, FP's operational semantics is imperative programming.
2. **Operational Thinking**
   IN FP data dependency implicitly determines sequencing whereas in Imp P it is done explicitly. Advantages and disadvantages of operational thinking.
3. **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 classness (the value of variables bound in environments could be other environments). Use of environments to model data abstraction, various object frameworks, module systems.
4. **Semi Explicit Continuation**
   Explicit in the sense that goto labels can be dealt with firstclassly (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'.
5. **Recursion iteration equivalence**
   General principles as well as scheme semantics of tailrecursion.
6. **Type Issues**
   Monomorphic, polymorphic and latent typing: translating one into another.
7. **Guile**
   A variety of vehicles have been used for the imperative paradigm, eg. 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 sexpressions is universal (aka XML) and thus imperative and functional thinking do not quarrel with datastructure issues.

   Orthogonal kinds of abstractions, which are usually considered 'advanced', such as functional, higherorder functional, objectoriented, streambased, datadriven, 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).

- **Bibliography**

  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 Maths with a computer, Hall and Donnell, Springer Verlag
  Guile Reference Manual, www.gnu.org

BACK TO TOP

CS-102 Logical Organization of Computers

- **Aims**
  There are two views of computer architecture. The traditional view, dating back to the IBM System/360 from the early 1960's, is that the architecture of a computer is the programmer-visible view of the machine, while its implementation is the province of the hardware designer. Nowadays this is known as instruction set architecture. In the more recent view, computer architecture has three parts: instruction set architecture, computer organization and hardware. Here computer organization refers to the high-level, abstract or essential aspects of a machine's structure and behavior. The CO course is intended to give the basic architecture background that software professionals need. In that sense it is a basic, first level course meant to provide the prerequisite for further study. However there is also an open, research oriented side to it. In the earliest days of computing the programmer and the hardware engineer were the same person but across the next 50 years the two fields have separated so rigidly that they can hardly understand each other today. Recent trends are bringing these two trends back together. So a second(ary) aim of this course is make it a preliminary towards appreciating and participating in these trends.

- **Contents**

  1. **From a calculator to a stored-program computer**
     The functionality of a calculator: electronics to perform arithmetic operations; memory to store partial results. Internal structure of a calculator that leads to this functionality. Machine language and programs writing a sequence of instructions to evaluate arithmetic expressions. Computer or computing assistant in the traditional sense of the word, i.e., human operating a calculator. Interpreting the computerï¿½s behavior when instructions are carried out: the fetch-decode-execute cycle as the basic or atomic unit of a computerï¿½s 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 (where computations take place) 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 1980ï¿½s, 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 Pentium is an instance of CISC, and hence is not considered in this course. The MIPS R2000, arguably the classic RISC machine, is the studentï¿½s first introduction to CO.

5. **Basic Electronics :**
Just those concepts needed to understand CO: combinational functions and their implementation with gates and with ROMï¿½s; 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.

- **Bibliography**
Computer Organization and Design, Patterson and Hennessey
Computer Structures, Ward and Halstead
Digital Design: Principles and Practices, Wakerley

BACK TO TOP

CS-103 Mathematical Foundations

- **Aims**

1. **Constructivity:** A view of mathematics as programming
2. **Impredicativity:** Limitations of constructivity
3. **Formality:** Habits of mathematical style: ''Definition... Theorem... Proof...''
4. **Informality:** Benefits and limitations of the above: Formality understanding
5. **Notation:** Use, elision and invention of appropriate notation

The following is too vast for a course. The instructor may select from it in keeping with the above aims and objectives and to achieve a smooth integration with the IP course.

- **Contents**

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. Selfreference, paradoxes, Godel's theorem Alternative Logics eg. 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. **Wellformed 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 powerset functions 3 basic types - equivalences, orders, functions - properties and applications 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 recipies
Philosophically - enriches the programmer's world by moving programs from syntax to firstclass semantics

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

- **Bibliography : Logic for CS by Gallier**
Discrete Maths by Tremblay Manohar
Discrete Maths by Stanat
Laws of Logical Calculi by Morgan
Category Theory tutorial by Hoare
Category Theory by Burstall and Rydheard
Computer modelling of mathematical reasoning by Bundy
Shape of mathematical reasoning by Gasteren
Predicate Calculus and Program Semantics by Dijkstra
Algebra of Programming by Richard Bird

Functional Programming with Bananas, Lenses and Barbed Wire by Fokkinga.
http://wwwhome.cs.utwente.nl/ï¿½fokkinga/#mmf91m
A Gentle Introduction to Category Theory the calculational approach by Fokkinga
http://wwwhome.cs.utwente.nl/ï¿½fokkinga/#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, combinato

BACK TO TOP

CS-104 Concrete Maths and Graph Theory

- **Aim**
  The aims of this course are to enable the student to

  1. Obtain mathematical formulations of real world combinational problems
  2. Solve them algorithmically
  3. Do simple analysis of efficiency of such algorithms
  4. Acquire the necessary mathematical background for doing deeper analysis of algorithms

     At the end of the course the student should be familier with

  1. The notion of a graph and the related concepts
  2. Algorithms to solve various graph theoretic problems
  3. Idea of efficiency of an algorithm and simple methods of estimating computing time of various algorithms
  4. tools of algorithm analysis such as solution of recurrence relations, asymptotic notation etc.

- **Contents**
  Graph Theory

  1. **Graphs**
     Definition and examples of graphs
     Incidence and degree, Handshaking lemma, Isomorphism
     Subgraphs, Weighted Graphs, Eulerian Graphs, Hamilitonian 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, maxflow mincut theorem
  3. **Planar Graphs**
     Combinational and geometric duals
     Kuratowski's graphs
     Detection of planarity, Thickness and crossings
  4. **Matrix Representation of Graphs**
     Incidence, Adjacency Matrices and their properties
  5. **Colouring**
     Chromatic Number, Chromatic Polynomial, the six and five color theorams, the four color theoram
  6. **Directed Graphs**
     Types of digraphs, directed paths and connectedness, Eular digraphs, Directed trees, Arborescence, Tournaments, Acyclic digraphs and decyclication
  7. **Enumeration of Graphs**
     Counting of labeled and unlabeled trees, Polya's theoram, Graph enumeration with Polya's theoram

- **Concrete Mathematics**

  1. **Sums**
     Sums and recurrences, Manipulation of sums, Multiple Sums, General methods of summation
  2. **Integer Functions**
     Floors and ceilings, Floor/Ceiling applications, Floor/Ceiling recurrences, Floor/Ceiling sum
  3. **Binomial Coefficients**
     Basic Identities, Applications, Generating functions for binomial coefficients
  4. **Generating Functions**
     Basic maneuvers, Solving recurrences, Convolutions, Exponential generating functions
  5. **Asymptotics**
     O notation, O manipulation, Bootstrapping, Trading tails

- **Bibliography**
  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
  Notes on Introductory Combinatorics, Polya, G. R. E. Tarjan & D. R. Woods [1983], BirkHauser
  Graph, Networks and Algorithms, Swamy, M. N. S. & K. Tulsiram [1981], John Willey

BACK TO TOP

CS-105 Numerical Methods

- **Aims**
  The stress in teaching Numerical Analysis should be to treat is as a mathematical discipline and not as an art. The stress should be on teaching unifying principles and to avoid teaching them as a bag of tricks. As such, the syllabus given below must be in followed in the order given below. Failure to follow this suggestion may result in Numerical Analysis slipping back into an unattractive subject.

  An Algorithm as a computational procedure should be differentiated from the theorem, which is a statement about what the Algorithm does.

  As many of our students are likely to come with a poor knowledge of complex variables, it is necessary to give an introduction to complex variable theory as part of numerical analysis course.

  We have omitted the conventional topics of numerical differentiation, numerical integration and numerical solution of ordinary differential equations. The argument in favour of these omissions is that these subjects involve an initial step of approximating the specific continuous process by replacing it by the approximating polynomial, difference equations or systems of equations . (These amount to modelling the continuous system by a discrete system).

  Next we proceed to work on these derived models. Our syllabus contains all the above mentioned topics needed for modelling. We feel that the topic of modelling should be kept out of our syllabus as it forms a major subject by itself. A casual mention about the methodologies of modelling mentioned above should be included for the students to realise that they have to learn it as a separate subject when necessary.

  With the above preliminaries we now list the topics of the syllabus with text books to guide us through.

- **Contents :**

  1. Introduction to Complex Variable theory
  2. Matrix Algebra
  3. Numerical Solution of Linear Equations. Direct Methods and Iterative Methods. Eigen value and Eigen vector calculation.
  4. Solutions of Systems of Nonlinear Equations
  5. Iteration : Convergence of iteration, Error, Accelerating Convergence, Aitkin's Method,Quodiotic Conveyance, Newton's Method, Diagonal Aitken's Method.
  6. Iteration for system of equations: Contraction Mapping, Lipschitz Condition, Quadratic Convergence, Newton's Methods, Bairstow's Method. Linear
  7. Difference Equations : Particular solution of Homogeneous Equation of order two, General Solution, Linear Dependence, Non Homogeneous Equation of order two, Linear Differnce Equation of Order N, System of Linearly independent Solutions.
  8. Propagation of roundoff error
  9. Interpolation and approximation
     Interpolating Polynomials, Existence, Error and Convergence of Interpolating. Polynomial Constuction 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 or these topics. The remaining topics will now be covered more or less in the same order as listed in the syllabus.

- **Bibliography**

  Elements of Numerical Analysis, Peter Henrici, John Wiley & Sons.
  Numerical Linear Algebra, Leslie Fox, Oxford University Press.
  Lecture Notes on Numerical Analysis, R. Sankar

BACK TO TOP

CS-201 Data Structures and Algorithms

- **Aims**
  To distinguish between and be able to relate the high level (mathematical) world of data structures and the low level (engineering) world of storage structures. To develop a vocabulary for algebraic manipulation of data structures and a calculus of systematic refinement to algorithms and storage structures in the low level world of C and machines.

  To round off the foundations laid in IP and MF by engineering slightly bigger software on realistic computer systems.

- **Prerequisites**
  Introduction to Programming, Mathematical Foundation & Logical Organization of Computers

- **Course Overview**

	Algebraic view	**Algorithmic view**
Data	Data Structures, Mathematical Definitions, Laws, Manipulations, MF relations	Storage Structures, Engineering Considerations related to CO, LLP
Code	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.	Explicit control through built in control structures like sequencing, if, while Engineering efficient implementation of correct specifications

- **Contents**
The course is organized according to the philosophy in the table below. The case studies/examples include but need not be limited to

  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

  Sorting and Searching: Specification and multiple refinements to alternative algorithms

  Polymorphic structures: Implementations (links with PP course)

  Complexity: Space time complexity corresponds to element reduction counts. Solving simple recurrences.

- **Course Organization**

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

- **Bibliography**

  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
  The Art of Programming Vol. 1. & Vol. 3, D. E. Knuth, Addison Wesley Inc

BACK TO TOP

CS-202 Theoretical Computer Science

- **Aims**

  1. To question informal techniques in programming bu inverting them into questions of programmability (computability). This can be achieved by exploring issues in computation and linking it with the logics of proff/decidability.

  2. To acquint with (i) Complexity (ii) Semantics and in this context systematically show the increase in power of Lower power formalisms, languages, automata in the direction of its very limits the notion of computability.

  3. To reformulate old, classical definitions of computability in the technologically relevant setting of modern programming languages, thus breaking the theory Vs practice divide.

  4. To build a conceptual glue that spans the triad automata, languages and computation.

- **Prerequisites**
Introduction to Programming, 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 Calculas/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 'nonVonNeumann' language.

5. **Self References :**
Use mention distinctions, 'escape methods' for selfreferencing quines, selfreferences in the expression domain, the formulation of the 'halting problem' and decidability in C and Scheme

6. **Recursive Data :**
Recursive, Enumerable sets, generators and recognisers formulated as recursive types in Haskell, 'S' expressions in Scheme.

7. Complexity Basic ideas measuring time usage, time hierarchies

8. Deterministic and Nondeterministic computations.

9. Ability od a mechanism to solve a problem. Formalization of the problem. Chruch Turing thesis.

10. Universality

11. Equations in language spaces
Operational approach
Denotational approach

- **Bibliography**
Introduction to the theory of computation, Sipser, Thompson Learning
Computabilities and complexity from a programming perspective, Niel Jones, MIT Press
The Quine page, Gary P. Thompson, at http://www.myx.net/ï¿½gthompso/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 Leeuvven ed, Elsevier

BACK TO TOP

CS-203 Low-Level Programming

- **Aims**
Modern Computer Systems have layer upon layer of s/w abstractions. These abstractions, though perhaps made with the best intentions, ultimately end up obscuring the actual workings of computers. The primary aim of the LLP course is to crack open this high level abstraction layer.

- **Prerequisites**
Computer Organization, Introduction to Programming

- **Objectives**
To understand the workings of a computer at the lowest levels where h/w and s/w meet.
C Programming (Basics here, advanced in Data structures and syspro).
To be able to write assembly language programs (in small doses) and to integrate C and assembly (in larger doses).
To be comfortable with low level system software.
The above to be done with respect to a specific OS depending on availability and instructors choice.

  Note: Originally this course was conducted entirely within MSï¿½DOS. But with DOS almost dead and other OS's not quite as convenient for low level hacking, there is some problem with infrastructure, and course material.

- **Contents**

1. C Language Basics

2. Assembly Language structure, syntax, macros

3. Use of linker, librarian, object editor(s), debugger

4. C Assembly Interfacing coding conventions, translations of arrays, structs, call return sequences. Mixed code.

5. 8086 architecture going up to P4. Survey of Intel architecture history

6. Machine language programing: Assembling and disassembling, clock cycle counting, instruction length calculation. Philosophy and history of instruction format choices. Combinatorial considerations and limitations.

7. I/O Classification: Memory mapped vs IP mapped. Polled, Interrupt, DMA

8. Interrupts: h/w and s/w. ISRs. Assembly and C. Minimization and handling of non determinism Separation of binding times: Hardcodings of chip, board, OS, system s/w,user levels

9. OS use: system call interface

10. OS implementation: Start up scripts, Basics of protected mode and device drivers
Chip Level Programming

- **Bibliography**
Art of Assembly, Randy Hyde
Intel Manuals
OS, chip manuals
Compiler and System S/w manuals
C Programming, Kernighan and Ritchie

BACK TO TOP

CS-205 Computer Architecture and Operating Systems

- **Aims:**
To describe the major architechtural styles of computer systems and the programmed abstract machine that is created over a given computer system via operating systems software.

- **Prerequisites :**
  Computer Organization, Introduction for Programming

- **Contents :**
  Register Transfer model of processors. Data paths and control structures. Comparison of architechtural styles for general purpose computers including RISC/CISC. Pipelining hazards and their resolution. Storage hierarchy in a computer: caches and virtual memory

- **I/O systems:**
  Polled and interrupt driven interfaces. Machine level devices like disks and serial/parallal ports, user level devices like keyboards and video units.

  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 simplecomputer systems. Memory management: segmentation, swapping, virtual memory and paging. Bootstraping 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 Ststems. Architechture 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.

- **Bibliography**

  D. A. Patterson & J. L. Hennessy, Computer Organization and Design: the hardware/software interface, MorganKaufmann.
  A. S. Tanenbaum, Structures Computer Organization, 3rd edition, PrenticeHall
  J. L. Hennessy & D. A. Patterson, Computer Architechture: a quantitative approach, MorganKaufmann
  A. S. Tanenbaum, Modern Operating Systems, PrenticeHall
  A. S. Tanenbaum, Distributed Operating Systems, PrenticeHall
  M. Singhal & N. Shivaratri, Advanced Concepts in Operating Systems, McGrawHill

BACK TO TOP

CS-206 Programming Paradigms

- **Aim :**
When students first study programming, the one style that they have learnt, they inevitably take to be THE style. The aim of the PP course is to convert that 'THE' into 'a'.

- **Prerequisites :**
Introduction to Programming

- **Objectives :**
A variety of different ways of thinking about programming are presented. The differnces are investigated so that the word 'paradigm' can begin to make sense the different languages covered are vehicles, not the goal.
The sense of the intellectual content for computer science as being not fixed but a melting pot of new ideas.
Some aspects of how these paradigms are implemented.
Note : Certain commonly covered paradigms such as functional paradigms are not here because they are integrated into the programming mainstream.

- **Contents**

 1. GUI Programming
 2. GUI Vs CUI
 3. Event Driven Programming
 4. Visual (Meta-GUI) Programming
 5. Architechture 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. Architechtural 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 Chistiansen (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 Docmentation at http://www.python.org/doc/

BACK TO TOP

CS-204 Design and Analysis of Algorithms

- **Aim**
This course focuses on fundamental techniques for the design and analysis of correct and efficient algorithms. After reviewing the applicable mathematics and introducing the basic concepts, the course presents several design techniques. First a technique is introduced in its full generality, and then it is illustrated by concrete examples drawn from several different application areas. Attention is given to the intregation of the design of an algorithm with the analysis of its efficiency and correctness. The course also introduces the concepts of computational complexity.

- **Prerequisites :**
Graph Theory and Concrete Mathematics, Data Structures and Algorithms

- **Contents :**

 1. String processing
 2. KnuthMorrisPlatt Algorithm, BoyerMoore Algorithm, pattern Matching.
 3. Graph and geometric Algorithms
 4. DFS, BFS, Biconnectivity, all pairs shortest paths, strongly connected components, network flow
 5. FordFulkerson Algorithm, MPN Algorithm, Karzanov Algorithm, Maximum Matching in bipartic 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
11. Introduction to the theory of NonPolynimial Completeness NonDeterministic Algorithms, Cook's Theoram, clique decision Problem, Node cover decision problem, chromatic number, directed Hamiltonian cycle, traveling salesman problem, scheduling problems.

- **Bibliography**
Introduction to Algorithms, Cormen, Leiserson, Rivest, MIT Press and McGraw Hill, 1990 Algorithms, Robert Sedgwick, Addison Wesley Publishing Company, 1988
The Design and Analysis of Computer Algorithms, A. V. Aho, J. E. Hopcroft, J. D. Ullman, Addison Weslay, Reading, Mass, 1974
Computer Algorithms: Introduction to Design & Analysis, Sara Baase, Allen Van Gelder, Addison Wesley Pub. Co., 2000
Computer Algorithms, Sara Baase, Addison Wesley, 1988
Combinational Algorithms (Theory and Practice) , F. M. Reingold, J. Nivergelt and N. Deo, Prentice Hall Inc., Engiewood Cliffs, N. J., 1977
Combinational Algorithms, T. C. Hu, Addison Wesley, 1982

BACK TO TOP

CS-301 Database Management System

- **Aim :**
Concepts in DBMS are taught in depth. The student will attain the ability to design Databases and understand the ACID principles and nittygritty involved in any DBMS development, through the implementation excercises carried out in the course.

- **Prerequisites :**
Data Structures and Algorithms

- **Contents**
DBMS objectives and architechtures

1. **Data Models**
Conceptual model, ER model, object oriented model, UML Logical data model, Relational, object oriented, objectrelational
2. **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.
3. **Relational database desing**
Schema design, Normalization theory, functional dependencies, lossless join property, join dependencies higher normal forms, integrity rules, Relational operators, relational completeness, Relational algebra, Relational calculas
4. **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
5. **Mapping mechanism**
conceptual to logical schema, Key issues related to for physical schema mapping
6. **DBMS concepts**
ACID Property, Concurrency control, Recovery mechanisms, case study Integrity, Views & Security, Integrity constraints, views management, data security
7. **Query processing, Query optimization -**
heuristic and rule based optimizers, cost estimates, Transaction Management
8. **Case Study**
ORACLE/POSTGRES DBMS package: understanding the transaction processing Concurrency and recovery protocols, query processing and optimization mechanisms througn appropriate queries in SQL and PLSQL.
9. **Web based data model -**
XML, DTD, query languages, Xpath, Xquery
10. **Advanced topics**
Other database systems, distributed, parallel and memory resident, temporal and spatial databases.
11. Introduction to data warehousing, OnLine Analytical Processing, Data Mining.

    Bench marking related to DBMS packages, database administration

- **References**
Database System Concepts, Silberschatz, Korth and Sudershan, McGraw Hill Company
Database Management Systems, Raghu Ramakrishnan, Johannes Gehrke, 2002
Principles of Database Systems Vol. I & Vol II, J. D. Ullman, Rockville, MD: Computer Science Press, 1998

BACK TO TOP

CS-302 Computer Networks

- **Aim :**
General principles and concepts of computer networks and the services built on top of them are covered. The student will attain the ability to design basic network services and implement network Systems.

- **Prerequisites :**
Computer Architechture & Operating Systems, Data Structures and Algorithms

- **Contents :**

1. Network architechture, ISO-OSI Reference model
2. Network Topology:
3. Topology design problem, connectivity analysis, delay analysis, backbone design, local access network design.
4. Physical Layer, Transmission media, digital transmission, transmission & switching,
5. Intregrated 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
    Local Area Networks, IEEE 802 standards for LAN (Ethernet, token ring, optical fiber, wireless)
11. Application layer environment
12. Application layer architechture, building applications with sockets, DNS, HTTP, SMTP, LDAP, NFS, NIS, SNMP, WAP
    Mobile computing
13. Internet, extranet, Virtual Private Network (includes tunneling, internet work routing and fragmentation)
14. Internet Security: Firewalls, SSL, Popular encryption protocols

- **References**
  Data and communications, 6th Edn., W. Stallings, Prentice Hall, 2000
  Computer networks: A systems approach, 2nd Edn., Peterson and Davie, Morgan Kaufman
  Computer Networks, 4th Edn., A. S. Tanenbaum, Prentice Hall
  UNIX Network Programming: Interprocess Communications, Stevens , Prentice Hall, 1999

BACK TO TOP

CS-303 Systems Programming

- **Aims:**
  To convey the idea that systems programs help in building an abstract machine over the raw machine, and are governed by the fundamental need of interpretation. Also attempt to demonstrate the mix of techniques from formal to heuristics that are used to write real programs.

- **Prerequisites**
  Computer Architechture and Operating Systems, Theoretical Computer Science, Data Structures and Algorithms

- **Contents :**

  The four dimensions of a programming activity as the basis for systems programming: concept, program generators (humans or other programs), sources and deliverables. For a variety of concepts, a set of program generators generate a set of (possibly overlapping) sources and produce a set of deliverables (executables, libraries, documentation).

  Interpretation as the fundamental activity in Software. Interpreters and interpretation. Program layout strategies on a Von Neumann machine (e.g. Pentium). Divison of the final interpretation goal into subtasks and establishing interface export by producer tool and import by consumer tool.

  Linkers and Loaders
  Linker as a layout specifying producer and loader as a layout specification consumer. Layout specification strategies: fixed and variable (relocatable and selfrelocatable). Layout calculations. Dynamic linking and overlays. Executable format definitions. Object file format as the interface betwen 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 kernal).

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

  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, AddisonWesley
  John Levine, Linkers and Loaders, http://www.iecc.com
  info lex and info bison on GNU/Linux Systems
  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/
  Basic Architechture: 24547012.pdf. Instruction Reference: 24547112.pdf
  System Programming Guide: 24547212.pdf

BACK TO TOP

CS-305 Computer Graphics

- **Aims**
Equip the student with the tools and techniquies required for the generation and manipulation of pictures/images. The device independent aspects as well as the device dependent quirks of color and gray scale devices is expected to be appreciated by the student at the end of this course.

- **Prerequisite(s)**
Students opting for this course should have a certain degree of programming experience. IP and DSA or their equivalent competence should suffice.

- **Contents**
Introduction, Image Processing as Picture Analysis and Computer Graphics as Picture Synthesis, Representative Uses of Computer Graphics, Classification of Applications.

  Raster Graphics Features, raster algorithms including primitiveslikelines, circles, filling, clipping in 2D, etc.

  Geometric transformations in 2D for 2D object manipulation, coordinate transformations and their matrix representation, Postscript language to demonstrate these concepts.

  The 3rd dimension, it's necessity and utility, transformations and modelling in 3D, geometric modelling with an introduction to curves and surfaces for geometric design, including but not restricted to Bezier, Bï¿½spline, hermite representations of curves and surfaces

  From 3D back to 2D projections, hidden surface elimination and the viewing pipeline. Achromatic Light, Chromatic Color, Color Models for Raster Graphics, Reproducing Color, Using Color in Computer Graphics

  Rendering Techniques for Line Drawings, Rendering Techniques for Shaded Images, Aliasing and Antialiasing, Illumination Models local models like Phong, CookTorrance and global models likeraytracing and radiosity, shading detail like textures, their generation and mapping, bump mapping and similar techniques.

  Depending on time availability, one of volume rendering, modelling of natural objects, introduction to 3D animation may be covered depending on student and instructor inclination

- **References**
Computer Graphics: Principles and Practice, J. Foley, A.van Dam, S. Feiner, J.Hughes, Addison Wesley Pub., 1997
Computer Graphics, D. Hearn, M. P.Baker, Prentice Hall, 1997
Computer Graphics, F. S. Hill Jr., Macmillan Pub, 1990
Curves and Surfaces for Computer Aided Geometric Design, 4th Edn., G. Farin, Academic Press, 1997
Mathematical Elements for Computer Graphics, 2nd Edn., D. Rogers, McGraw Hill Pub., 1990
The Mathematical Structure of Raster Graphics, E. Fiume, Academic Press, 1989
Graphics Gems , Vol. 15, Academic Press
The Rendering Equation, J. Kajiya, SIGGRAPH 1986, 143ï¿½150

BACK TO TOP

CS-401 Modeling and Simulation

- **Aims**
Introduce the student to practical/realworld systems which require understanding and defy complete (if any) analytical methods towards their analysis and hence the requirement for modelling and simulation. This will include the mathematical, statistical and language tools required for specifying a model, wunning the simulation and analysing the results. The course would emphasize DES while showing linkages to other types of simulation.

- **Prerequisites**
Introduction to Programming, Data Structures and Algorithms (at the discretion of the instructor)

- **Contents :**

  1. Introduction to Systems modelling concepts, contimous and discrete formalisms
  2. Framework for Simulation and Modelling, modelling formalisms and their simulators, dicrete time, contimous time, discrete ecevt, 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 behaviour, steadystate behaviour of stochastic systems, computing alternative systems, variance reduction techniques.
  8. Verification and Validation

- **References**
Discrete Event System Simulation, 3rd ed., J. Banks, J. Carson, B. Nelson, D. Nicol, Prentice Hall Pub., 2001
Simulation Modelling and Analysis, 3rd ed., A. Law, W. Kelton, McGraw Hill Pub., 2000
Simulation with Arena, 2nd ed., W. Kelton, R. Sadowski, D. Sadowski, McGraw Hill Pub., 2002
Theory of modelling and Simulation, 2nd ed., B. Zeigler, H. Praehofer, T. Kim, Academic Press, 2000
Object Oriented Simulation with Hierarchial Modular Models, B. Zeigler, Academic Press, 1990
Reality Rules, Vol. I and Vol. II, J. Casti, John Wiley Pub., 1996

BACK TO TOP

CS-402 Operations Research

- **Aims**
This course will focus on two aspects of OR, viz. deterministic (mathematical programming) and stochastic. At the end of the course the student should have developed or honed his/her skills at modelling (or at least problem formulation) and be able to choose an appropriate quantitative technique towards it's solution, or be aware that the problem on hand is *not* ammenable to quantitative techniques.

- **Prerequisite(s):**
No course related prerequisite but a background in basic differential and integral calculus is asumed along with familiarity with matrix algebra.

- **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.
2. Linear Programming, Introduction, Graphical Solution and Formulation of L.P.Models, SimplexMethod (Theory and Computational aspects), Revised Simplex, Duality Theory and applications Dual Simplex method, Sensitivity analysis in L.P., Parametric Programming, Transportation, assignment and leastcost transporation, interior point methods: scaling techniques, log barrier methods, dual and primaldual extensions
3. Introduction to game theory
4. Multiobjective 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. NelderMead 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, queing 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 newshoy problem : a single period model, a lot size reorder point model

- **References**
Operations Research: An Introduction, 7th Edn., H. Taha, Prenticeï¿½Hall, 2002
Operations Research: Principles and Practice, A. Ravindran, D, Phillips, J Solberg, John Wiley Pub, 1987
Linear Programming and Extensions, G Dantzig, Princeton University Press, 1963
Theory of Games and Economic Behaviour, J. von Neumann, O. Morgenstern, John Wiley Pub. 1967
Goal Programming: Methodology and Applications, M. Schniederjans, Kluwer Academic Pub, 1995

BACK TO TOP

CS-403 System Analysis and Design

- **Aims**
Software development is a complex process. Good quality software results by using disciplined and methodological approaches for requirement analysis, design and coding. In this course, the student is introduced to both formal and less formal techniques used for requirement and domain analysis. At the end of the course the student will

  Become more adapt in understanding a problem in terms of its processes and concepts and design a good solution using CASE tools. Have sound understanding of software engineering issues for large scale development through modelling and notation and provide a foundation for the Software Engineering course (which is taught later), so as to be able to develop a piece of quality software according to sound principles and notation.

- **Prerequisites**
Mathematical Foundation, Programming Paradigms, exposure to DBMS is preferred.

- **Syllabus**

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 Softawre 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**
The Engineering of Software, Dick Hamlet, Joe Maybee, Addison Wesley, 2001
UML Distilled, 2nd Ed., Martin Fowler, Addison Wesley
Introduction to the Personal Software Process, Watts S. Humphery, Addison Wesley, 1997
Using UML for Software Engineering, Pooley and Stevens, Addison Wesley, 1999
The Unified Modelling Language Users Guide, 1st Ed., Grady Booch, James Rumbaugh and Ivar Jacobdon, Addison Wesley, 1999
Specification Case Study, Hayes, Prentice Hall
Currie: The Essence of Z ISBN 013749839X, Prentice Hall
UML Toolkit, Eriksson, John Wiley, 1998

BACK TO TOP

CS-304 Science Of Programming

- **Aims :**
  To teach the use of formal methods in software development. Although formality should not be sacrificed too much, scaling up in software size with reduction in formality should be illustrated when possible .

- **Contents**

  1. **Verification :** verification of imperative programs as in Gries/Dijkstra.
  2. **Specific techniques :** Invariant assertive method, subgoal induction method.
  3. Verification of pointer programs.
  4. **Function Program verification:** Induction on datatypes, infinite datastructure induction.
  5. **Specification :** Use of 'Z' as an modeltheoretic language.
  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

- **Bibliography**

  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
  Partial Evaluation and Mixed Computation, Ershov, Bjorner & Jones, North Holland.
  Programs from Specifications, Morgan, Prentice Hall
  Lectures of constructive functional programming, Bird, Lecture notes, PRG Oxford
  Introduction to the theory of lists, Bird, Lecture notes, PRG Oxford
  A calculus of functions for program derivation, Bird, Lecture notes, PRG Oxford
  Introduction to Formal Program verification, Mili, Van Nostrand Reinhold

BACK TO TOP

CS-601 Software Engineering

- **Aims**
  Software engineering is concerned with the cost effective development and evolution of software systems. This course introduces the topics through lectures and by giving the students a chance, in the form of a class project (which is a group project), to develop a software product and to manage its development process. It is a combination of the System Analysis and Design course and covers all other issues that sre needed in Software Engineering.

- **Prerequisites**
  System Analysis and Design, Data Structures & Algorithms, Systems Programming and good technical writing skills.

- **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 protyping approaches
  3. Software measurement
  4. Objectoriented 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 architechture
  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 Meturity Model
      Software Acquisition Capability Maturity Model
      Systems Engineering Capability Maturity Model
  17. IEEE software engineering standards
      The course should consist of lectures and a weekly discussion section. Students should work in teams on problem

analysis and other assignments during the discussion section. The lecture part of the course may include individual and group activities.

- **Bibliography**

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
Software Engineering A Practitioner's Approach European Adaption, 5th Edn., Roger S. Pressman, adapted by Darrel Ince, McGraw Hill, 2000
Software Engineering Theory and Practice, Shari Lawrence Pfleeger, Prentice Hall, 1998
Practical Software measurement, Bob Huges, McGraw Hill, 2000
Human Computer Interaction, 2nd Edn., Dix, Finlay, Abowd and Beale, Prentice Hall, 1997
Software Project Management, 2nd Edn., Bob Huges & Mike Cotterell, McGraw Hill, 1999

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:

Worlds	**The Timeless World**	**World of Time**
Domain	Mathematics	Programming
Syntax	Expressions	Statements
Semantics	Values	Objects
Explicit	Data Structures	Control Structure
Think 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.

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
Guile Reference Manual, www.gnu.org

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

CS-103 Mathematical Foundations

Contents :

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 Rydheard
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
Functional Programming with Bananas, Lenses and Barbed Wire by Fokkinga.
http://wwwhome.cs.utwente.nl/'fokkinga/#mmf91m
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 :**
     Incidence, Adjacency Matrices and their properties
  5. **Coloring :**
     Chromatic Number, Chromatic Polynomial, the six and five color theorems, the four color theorem
  6. **Directed Graphs :**
     Types of digraphs, directed paths and connectedness, Euler digraphs, Directed trees, Arborescence, Tournaments, Acyclic digraphs and decyclication
  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
  Notes on Introductory Combinatorics, Polya, G. R. E. Tarjan & D. R. Woods [1983], BirkHauser
  Graph, Networks and Algorithms, Swamy, M. N. S. & K. Tulsiram [1981], John Willey

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
Database Management Systems, Raghu Ramakrishnan, Johannes Gehrke, 2002.
Relational Database Index Design and the Optimizers by Tapio Lahdenm¨aki Michael Leach, John Wiley
PostgreSQL, Sams Publications
Principles of Database Systems Vol. I & Vol II, J. D. Ullman, Rockville, MD: Computer Science Press, 1998

CS-201 Numerical Methods

- **Contents :**
  1. Introduction to Complex Variable theory
  2. Matrix Algebra
  3. Numerical Solution of Linear Equations. Direct Methods and Iterative Methods. Eigen value and Eigen vector calculation.
  4. Solutions of Systems of Nonlinear Equations
  5. Iteration : Convergence of iteration, Error, Accelerating Convergence, Aitkin's Method, Quodiotic Conveyance, Newton's Method, Diagonal Aitken's Method.
  6. Iteration for system of equations: Contraction Mapping, Lipschitz Condition, Quadratic Convergence, Newton's Methods, Bairstow's Method. Linear
  7. Difference Equations : Particular solution of Homogeneous Equation of order two, General Solution, Linear Dependence, Non Homogeneous Equation of order two, Linear Difference Equation of Order N, System of Linearly independent Solutions.
  8. Propagation of roundoff error
  9. Interpolation and approximation
     Interpolating Polynomials, Existence, Error and Convergence of Interpolating. Polynomial Constuction 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**

	Algebraic view	Algorithmic view
Data	Data Structures, Mathematical Definitions, Laws, Manipulations, MF relations	Storage Structures, Engineering Considerations related to CO, LLP
Code	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.	Explicit control through built in control structures like sequencing, if, while Engineering efficient implementation of correct specifications

- **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**

	Algebraic world	Algorithmic world
Correctness	Bird Laws, Category Theory	Refinement, Predicates
Transformation	Via Morgan Refinement	
ADTs and Views	○ Formulation as recursive data types ○ Data structure invariants ○ Principles of interface design ○ Algebraic Laws	○ C storage ○ Representation Invariants ○ Addressing Semantics ○ Use of struct, union and other assorted C stuff ○ Maximizing abstraction

		by macros, enums etc
Mapping	Via transforms and coupling invariants	
Code	○ Pattern Matching based recursive definitions ○ Exhaustive set of disjoint patterns correspond to total functions ○ Correspond to runtime bug free programs ○ Recursive Code structures follow from recursive data structures	○ Refinement of recursive definitions into iterative algorithms ○ Techniques (Bentley) for improving algorithms e.g. sentinel, double pointers, loop condition reduction, strength reduction etc.
Continuations	○ Control as Data ○ Co routines vs. subroutines ○ General framework for escape procedures, error handling	○ Loops ○ Functions @ ○ Stack based software architecture
Error Policy Types	○ Patterns ○ Laws ○ Deliberate Partiality	Predicate Transformer Semantics for control
Modules	Category Theory	Files, make

- **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
The Art of Programming Vol. 1. & Vol. 3, D. E. Knuth, Addison Wesley Inc

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
  4. C Assembly Interfacing coding conventions, translations of arrays, structs, call return sequences. Mixed code.
  5. 8086 architecture going up to P4. Survey of Intel architecture history
  6. Inline Assembly, Floating point operations
  7. Machine language programming: Assembling and disassembling, clock cycle counting, instruction length calculation. Philosophy and history of instruction format choices. Combinatorial considerations and limitations.
  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 non determinism 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.
  3. Memory management: segmentation, swapping, virtual memory and paging. Bootstrapping issues. Protection mechanisms.
  4. Abstract I/O devices in Operating Systems. Notions of interrupt handlers and device drivers. Virtual and physical devices and their management.
  5. Introduction to Distributed Operating Systems. Architecture designs for computer systems with multiple processors, memories and communication networks. Clocking problem and Lamport's solution.
  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
M. Singhal & N. Shivaratri, Advanced Concepts in Operating Systems, McGraw Hill
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.
  4. Function Program verification: Induction on data-types, infinite data structure induction
  5. Specification : Use of 'Z' as a model theoretic language.
  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
Partial Evaluation and Mixed Computation, Ershov, Bjorner & Jones, North Holland.
Programs from Specifications, Morgan, Prentice Hall
Lectures of constructive functional programming, Bird, Lecture notes, PRG Oxford
Introduction to the theory of lists, Bird, Lecture notes, PRG Oxford
A calculus of functions for program derivation, Bird, Lecture notes, PRG Oxford
Introduction to Formal Program verification, Mili, Van Nostrand Reinhold

CS-301 Design and Analysis of Algorithms

- **Contents :**
  1. String processing
  2. Knuth-Morris-Platt Algorithm, Boyer-Moore Algorithm, pattern Matching.
  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 bipartic 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
  11. Introduction to the theory of non-Polynomial Completeness Non-Deterministic Algorithms, Cook's Theorem, clique decision Problem, Node cover decision problem, chromatic number, directed Hamiltonian cycle, traveling salesman problem, scheduling problems.

- **References:**

Introduction to Algorithms, Cormen, Leiserson, Rivest, MIT Press and McGraw Hill, 1990
Algorithms, Robert Sedgwick, Addison Wesley Publishing Company, 1988
The Design and Analysis of Computer Algorithms, A. V. Aho, J. E. Hopcroft, J. D. Ullman, Addison Weslay, Reading, Mass, 1974
Algorithm Design: Foundations, Analysis, and Internet Examples Michael T. Goodrich, Wiley
Computer Algorithms: Introduction to Design & Analysis, Sara Baase, Allen Van Gelder, Addison Wesley Pub. Co., 2000
Computer Algorithms, Sara Baase, Addison Wesley, 1988
Combinational Algorithms (Theory and Practice) , F. M. Reingold, J. Nivergelt and N. Deo, Prentice Hall Inc., Engiewood Cliffs, N. J., 1977
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.
  9. Ability of a mechanism to solve a problem. Formalization of the problem. Church Turing thesis.
  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/’gthompso/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 Leeuvven 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 :**

  Data and communications, 6th Edn., W. Stallings, Prentice Hall, 2000
  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 Wesley2003

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). Layout calculations. Dynamic linking and overlays. Executable format definitions. Object file format as the interface between the compiler and the linker. Few Object file formats like MSDOS, Windows and ELF. Object file manipulation utilities. 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
  John Levine, Linkers and Loaders, http://www.iecc.com
  System Software: An Introduction to Systems Programming, Leland L. Beck Pearson Education
  info lex and info bison on GNU/Linux Systems
  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/
  Basic Architecture: 24547012.pdf. Instruction Reference: 24547112.pdf
  System Programming Guide: 24547212.pdf

CS-305 Distributed computing

Contents

- What is distributed computing?
- Why distributed computing?
- Concepts of time, logical and physical clocks
- Concurrency: including multithreading
    - 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
Sacha Krakowiak, Advances in Distributed Systems: Advanced Distributed Computing, From Algorithms to Systems, Springer
Nicolai Josuttis, SOA in Practice: The Art of Distributed System Design (In Practice), O'Reilly

CS-401 Computer Graphics

- **Contents:**
    1. Introduction, Image Processing as Picture Analysis and Computer Graphics as Picture Synthesis, Representative Uses of Computer Graphics, Classification of Applications.
    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
    5. From 3D back to 2D projections, hidden surface elimination and the viewing pipeline. Achromatic Light, Chromatic Color, Color Models for Raster Graphics, Reproducing Color, Using Color in Computer Graphics
    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 :**

Computer Graphics: Principles and Practice, J. Foley, A.van Dam, S. Feiner, J.Hughes, Addison Wesley Pub., 1997
Computer Graphics, D. Hearn, M. P.Baker, Prentice Hall, 1997
Computer Graphics, F. S. Hill Jr., Macmillan Pub, 1990
Curves and Surfaces for Computer Aided Geometric Design, 4th Edn., G. Farin, Academic Press, 1997
Mathematical Elements for Computer Graphics, 2nd Edn., D. Rogers, McGraw Hill Pub., 1990
The Mathematical Structure of Raster Graphics, E. Fiume, Academic Press, 1989
Graphics Gems , Vol. 15, Academic Press
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 :**

Discrete Event System Simulation, 3rd ed., J. Banks, J. Carson, B. Nelson, D. Nicol, Prentice Hall Pub., 2001
Simulation Modeling and Analysis, 3rd ed., A. Law, W. Kelton, McGraw Hill Pub., 2000
Simulation with Arena, 2nd ed., W. Kelton, R. Sadowski, D. Sadowski, McGraw Hill Pub., 2002
Theory of modeling and Simulation, 2nd ed., B. Zeigler, H. Praehofer, T. Kim, Academic Press, 2000
Object Oriented Simulation with Hierarchical Modular Models, B. Zeigler, Academic Press, 1990
Reality Rules, Vol. I and Vol. II, J. Casti, John Wiley Pub., 1996

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.
  2. Linear Programming, Introduction, Graphical Solution and Formulation of L.P. Models, Simplex Method (Theory and Computational aspects), Revised Simplex, Duality Theory and applications Dual Simplex method, Sensitivity analysis in L.P., Parametric Programming, Transportation, assignment and least cost transportation, interior point methods: scaling techniques, log barrier methods, dual and primal dual extensions
  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 newshoy problem : a single period model, a lot size reorder point model
- **References :**

  Operations Research: An Introduction, 7th Edn., H. Taha, Prentice'Hall, 2002
  Operations Research: Principles and Practice, A. Ravindran, D, Phillips, J Solberg, John Wiley Pub, 1987
  Linear Programming and Extensions, G Dantzig, Princeton University Press, 1963
  Theory of Games and Economic Behavior, J. von Neumann, O. Morgenstern, John Wiley Pub. 1967
  Goal Programming: Methodology and Applications, M. Schniederjans, Kluwer Academic Pub, 1995

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
Introduction to the Personal Software Process, Watts S. Humphery, Addison Wesley, 1997
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 Jacobdon, Addison Wesley, 1999
Software Engineering Peters, Wiley India
Specification Case Study, Hayes, Prentice Hall
Currie: The Essence of Z ISBN 013749839X, Prentice Hall
UML Toolkit, Eriksson, John Wiley, 1998

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 Chistiansen (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
  Software Engineering A Practitioner's Approach European Adaption, 5th Edn., Roger S. Pressman, adapted by Darrel Ince, McGraw Hill, 2000
  Software Engineering Theory and Practice, Shari Lawrence Pfleeger, Prentice Hall, 1998
  Practical Software measurement, Bob Huges, McGraw Hill, 2000
  Human Computer Interaction, 2nd Edn., Dix, Finlay, Abowd and Beale, Prentice Hall, 1997
  Software Project Management, 2nd Edn., Bob Huges & Mike Cotterell, McGraw Hill, 1999

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
Practical Software measurement, Bob Huges, McGraw Hill, 2000
Thinking in Java Vol 3 by Bruce Eckel at 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
Python Documentation at http://www.python.org/doc/
Boost Lambda Library for C++, 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
Introduction to the Personal Software Process, Watts S. Humphery, Addison Wesley, 1997
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
Software Engineering A Practitioner's Approach European Adaption, 5th Edn., Roger S. Pressman, adapted by Darrel Ince, McGraw Hill, 2000
Software Engineering Theory and Practice, Shari Lawrence Pfleeger, Prentice Hall, 1998
Practical Software measurement, Bob Huges, McGraw Hill, 2000
Human Computer Interaction, 2nd Edn., Dix, Finlay, Abowd and Beale, Prentice Hall, 1997
Software Project Management, 2nd Edn., Bob Huges & Mike Cotterell, McGraw Hill, 1999