Department of Computer Science, SPPU		
Course: M.C.A. (Master in Computer Application)		
Duration: 03 Years		
Total No. of credits for the Course per semester		
Semester	**Core Course Credits**	**Elective Course Credits**
1	**20**	**0**
2	**20**	**0**
3	**8**	**12**
4	**15**	**6**
5	**2**	**0**
6	**10**	**7**
Total Credits =120	**75**	**25**
M.C.A. Semester I		
Subject Code	**Subject Title**	**Number of Credits**
CS-101	**Programming – 1**	**5**
CS-102	**Systems- 1**	**4**
CS-103	**Mathematics of Computation– 1**	**4**
CS-104	**Mathematics of Computation- 2**	**4**
CS-105	**Database Management System**	**3**
UGC recommended	**Skill Development**	**2**
UGC recommended	**HRE**	**1**
Total Credits		**20+3**
M.C.A. Semester II		
Subject Code	**Subject Title**	**Number of Credits**
CS-201	**Programming – 2**	**5**
CS-202	**Programming – 3**	**4**
CS-203	**Systems-2**	**4**
CS-204	**Systems-3**	**4**
CS-205	**Theory of Computation**	**3**
UGC recommended	**Skill Development**	**2**
UGC recommended	**HRE**	**1**
Total Credits		**20+3**

Department of Computer Science, SPPU		
Course: M.C.A. (Master in Computer Application)		
M.C.A. Semester III		
Subject Code	**Subject Title**	**Number of Credits**
CS-301	**Systems-4**	**4**
CS-302	**Formal Methods-1**	**4**
CS-303	**Elective-1**	**4**
CS-304	**Elective-2**	**4**
CS-305	**Elective-3**	**4**
UGC recommended	**Cyber Security - 1 and 2**	**2**
Total Credits		**20+2**
M.C.A. Semester IV		
Subject Code	**Subject Title**	**Number of Credits**
CS-401	**Formal Methods-2**	**5**
CS-402	**Software Development -1**	**5**
CS-403	**Software Development -2**	**5**
CS-404	**Elective-4**	**3**
CS-405	**Elective-5**	**3**
UGC recommended	**Cyber Security - 3 and 4**	**2**
Total Credits		**20+2**
M.C.A. Semester V		
Subject Code	**Subject Title**	**Number of Credits**
CS-501	**Internship**	**16**
CS-502	**Seminar**	**2**
Total Credits		**18**
M.C.A. Semester VI		
Subject Code	**Subject Title**	**Number of Credits**
CS-601	**Programming Paradigms**	**5**
CS-602	**Software Development -3**	**5**
CS-603	**Elective-7**	**4**
CS-604	**Elective-8**	**4**
CS-605	**Elective-9**	**4**
Total Credits		**22**

Course contents in detail

CS-301 Systems - 4

Objective: This is the fourth course in the collection on systems. At the end of the course a successful student should be able to appreciate the issues involved in developing a software subsystem in which the primary activity is fundamentally concurrent and not only data but computations can be divided across multiple possibly heterogeneous computing devices.

- Overview of concurrency control – Serializable scheduling, correctness of schedules, generation of efficient and correct schedules; atomicity, consistency, isolability and durability (ACID) properties; lock-based concurrency control, transactions as a concurrency control mechanism.
- Concurrent programing - concurrency as a fundamental aspect of programming high performance systems, concurrent programing is not just parallel programing, concurrent programing is not just distributed programing.
- Composable concurrency – impossibility of composability of lock based concurrent programs, introduction to software transactional memory (STM), STM based composition.
- Distributed computations – concurrent systems with lack of global clock, Message passing model, consensus problem, mutual exclusion in distributed systems, problem of atomic commit, 2-phase commit, 3-phase commit.

References:

- Principles of concurrent and distributed programming - Mordechai Ben-Ari
- Concurrent programming: principles and practice - Gregory R Andrews
- On Concurrent Programming - Fred B. Schneider
- Concurrent Programming: Algorithms, Principles, and Foundations - Michel Raynal
- Principles of Transactional Memory (Synthesis Lectures on Distributed Computing Theory) - Rachid Guerraoui, Michal Kapalka
- Control Flow and Data Flow: Concepts of Distributed Programming: International Summer School - E. W. Dijkstra et el
- A Discipline of Multiprogramming: Programming Theory for Distributed Applications (Monographs in Computer Science) - Jayadev Misra

Note: the topic “Overview of concurrency control”, should be covered in brief as an overview of topics learnt in the earlier year.

Suggested further reading:

- Functional Programming for Loosely-Coupled Multiprocessors (Research Monographs in Parallel and Distributed Computing) - Paul H. J. Kelly
- Parallel and Concurrent Programming in Haskell: Techniques for Multicore and Multithreaded Programming - Simon Marlow
- Erlang Programming : a Concurrent Approach to Software Development - Simon Thompson
- Programming ERLANG: Software for a Concurrent World - Joe Armstrong
- Communicating sequential processes - C.A.R. Hoare

CS-302 Formal Methods - 1

Objective: This is the first course in the collection on formal methods. At the end of the course a successful student should be able to appreciate the basics of formalisms that play a crucial role in the verification and validation of software programs.

- Verification : verification of imperative programs as in Gries/Dijkstra.
- Specific techniques: Invariant assertive method, sub-goal induction method.
- Verification of pointer programs.
- Function Program verification: Induction on data-types, infinite data structure induction
- Homomorphic transformations, refinement Calculus Theory & application of List/Functional calculus
- Hoare Logic - correctness of computer programs, Hoare triple, Partial and total correctness, weakest precondition and strongest postcondition,
- Advanced topics– temporal logic, formal analysis of reactive systems using temporal logic.

References:

- Discipline of Programming, Dijkstra, Prentice Hall
- Method of Programming, Dijkstra & Feijen, Addison Wesley
- Logic of Programming and Calculi of Discrete Design: International Summer School - Richard S. Bird et el
- Specification Case Studies, Hayes, Prentice Hall
- Software Specification, Gehani & Mcgettrick, Addison Wesley
- Program Specifications & Transformations, Meertens, Prentice Hall
- Partial Evaluation and Mixed Computation, Ershov, Bjorner & Jones, North Holland.
- Programs from Specifications, Morgan, Prentice Hall
- Lectures of constructive functional programming, Bird, Lecture notes, PRG Oxford
- Introduction to the theory of lists, Bird, Lecture notes, PRG Oxford
- A calculus of functions for program derivation, Bird, Lecture notes, PRG Oxford

- Introduction to Formal Program verification, Mili, Van Nostrand Reinhold
- Logic in Computer Science: Modelling and Reasoning about Systems, by Michael Huth , Mark Ryan Cambridge Univeristy Press

Suggested further reading:

- First-Order Programming Theories (Monographs on Theoretical Computer Science) - Tamás Gergely, László Úry
- The Temporal Logic of Reactive and Concurrent Systems: Specification - Z. Manna and A. Pnueli
- Temporal Verification of Reactive Systems: Safety - Z. Manna and A. Pnueli
- Introduction to Mathematics of Satisfiability – V. W. Marek
- Value-Range Analysis of C Programs: Towards Proving the Absence of Buffer Overflow Vulnerabilities - A. Simon

CS-401 Formal Methods – 2

Objective: This is the second course in the collection on formal methods. At the end of the course a successful student should be able to apply formal techniques of program verification/validation to large scale software applications, and also appreciate the underlying theoretical basis.

- Software verification – Rice's theorem and the problem of undecidability of program verification; notion of non-trivial properties of partial functions, tractability issues in program verification.
- Introduction to program semantics - denotational semantics, axiomatic semantics, operational semantics.
- Static program analysis - Abstract interpretation, Symbolic execution, Data-flow analysis, Hoare logic, model checking.
- Abstract interpretation – sound approximations of semantics of programs, monotonic functions over ordered sets; abstraction function, concretization function, and the concept of valid abstraction;
- Symbolic execution – symbolic interpretation and constraint solving, problem of path explosion, problem of side effects, handling side effects with environment modeling and/or by creating virtual machines.
- Data-flow analysis – control flow graphs and fixpoints, the problem of reaching definition, blocks and computing entry and exit states of blocks; forward and backward analysis and convergence issues; liveness analysis as a subproblem in data-flow analysis.
- Model checking – state explosion problem, Symbolic model checking, binary decision diagrams.
- Alternative approaches to software verification - interactive theorem provers, automatic theorem provers, or satisfiability modulo theories (SMT) solvers.
- Concurrency and verification: Models of concurrency, The parallel random-access machine model, the actor model, bulk synchronous parallel (BSP) model, Petri nets model, Process calculi models (Communicating sequential processes (CSP) model)

Note: Teacher may introduce a suitable tool or two (e.g. by taking into consideration the teaching-learning activities happening in the Software Development 1 and Software Development 2 courses, or to complement the same) e.g. from the following non-exhaustive list: [Alloy analyzer, BLAST, FDR, Prism etc]

References:

- Semantics with Applications: An Appetizer - Flemming Nielson et el

- Path-Oriented Program Analysis - J. C. Huang
- Generation of Program Analysis Tools [PhD Thesis] (University of Amsterdam) - Frank Tip
- The Calculus of Computation: Decision Procedures with Applications to Verification - Z. Manna and A. Bradley

Suggested further reading:

- Principles of Program Analysis - Flemming Nielson et el
- Trustworthy compilers - Vladimir O. Safonov
- Introduction to Mathematics of Satisfiability – V. W. Marek
- Value-Range Analysis of C Programs: Towards Proving the Absence of Buffer Overflow Vulnerabilities - A. Simon
- Formal Methods: State of the Art and New Directions – Dines Bjørner

CS-402 Software Development – 1

Objective: This is the first course in the collection on software development. At the end of the course a successful student should be able to appreciate issues involved in large scale software development.

- Issues in software development - software development is hard, problems due to the discrete nature of software.
- Requirements Engineering - derivation of requirements prescriptions from domain models, principles and techniques for the refinement of requirements into software designs.
- Modern approaches to software development – Metaprogramming, software organization techniques, Domain Specific Languages and Test driven development.
- Metaprogramming: metaprogramming as a technique to move computations from run-time to compile time, metalanguage, attribute oriented programming, generics, limits of metaprogramming, template/macro based metaprogamming
- Domain Specific Language (DSL) based techniques - Domain-Specific Languages (DSLs) to complex code, design goals of DSLs, idioms vs DSLs, place and scope of DSLs.
- Software organization techniques – separation of concerns, modules and namespaces, libraries, frameworks, comparison of DSLs, frameworks and libraries, Code Refactoring
- Test driven development (TDD) – TDD is not agile development, TDD is not extreme programming, automatic test-case generation, programming to interface, testing properties of programs.

Note: Students are expected to carry out large scale software development as part of this course and should actively apply the concepts learnt.

References:

- Software Engineering 1: Abstraction and Modelling (Texts in Theoretical Computer Science) - Dines Bjørner
- Software Engineering 2: Specification of Systems and Languages (Texts in Theoretical Computer Science) - Dines Bjørner
- Software Engineering 3: Domains, Requirements, and Software Design (Texts in Theoretical Computer Science) - Dines Bjørner
- Generic Programming: Advanced Lectures - Ralf Hinze et el
- From Mathematics to Generic Programming - Alexander A. Stepanov, Daniel E. Rose

- Software Languages: Syntax, Semantics, and Metaprogramming - Ralf Lämmel
- Domain-Specific Languages - Martin Fowler
- DSL Engineering: Designing, Implementing and Using Domain-Specific Languages - Markus Voelter
- The Mythical man month – Fred Brooks

Suggested further reading:

- Domain-Specific Languages: IFIP TC 2 Working Conference, DSL 2009 Oxford, UK, July 15-17, 2009 Proceedings (Lecture Notes in Computer Science 5658)
- C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond - David Abrahams, Aleksey Gurtovoy
- Foundations of Algebraic Specification and Formal Software Development (Monographs in Theoretical Computer Science) - Donald Sannella, Andrzej Tarlecki
- Semantic Integration of Heterogeneous Software Specifications (Monographs in Theoretical Computer Science) - Martin Große-Rhode

CS-403 Software Development – 2

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

- Overview of software development as an engineering discipline: Is software development a true engineering discipline? – critical evaluation of the status of software engineering as an engineering discipline; comparison of software engineering with other branches of engineering; State of the art.

- Models of software systems - Hierarchical versus compositional, denotational versus computational, and configurational (abstracting and modeling contexts and states), modeling semiotics.

- Underpinnings of large scalable software – fundamental role of concurrency in scalable systems, understanding process interactions.

- Issues in the development of large scalable software – Concurrency issues, communicating sequential processes (CSP) as a model.

- CSP in detail – formal definition, Hoare's CSP notation, semantics of CSP, traces model, the stable failures model, and the failures/divergences model.

- CSP based software development – operators of CSP, introduction to Failures/Divergence Refinement (FDR), specifying and analyzing programs written in Hoare's CSP notation using FDR.

Note: Students are expected to carry out large scale software development as part of this course and should actively apply the concepts learnt.

References:

- Communicating sequential processes - C.A.R. Hoare

- "Model-checking CSP" In A Classical Mind: essays in Honour of C.A.R. Hoare – A. W. Roscoe

- Software Systems Reliability and Security – C. A. R. Hoare et el

- Software Performance and Scalability: A Quantitative Approach (Quantitative Software Engineering Series) - Henry Liu

- http://www.cs.ox.ac.uk/projects/fdr/

- Software Engineering 1: Abstraction and Modelling (Texts in Theoretical Computer Science) - Dines Bjørner

- Software Engineering 2: Specification of Systems and Languages (Texts in Theoretical Computer Science) - Dines Bjørner
- Software Engineering 3: Domains, Requirements, and Software Design (Texts in Theoretical Computer Science) - Dines Bjørner
- The Mythical man month – Fred Brooks

Suggested further reading:

- Fundamentals of a Theory of Asynchronous Information Flow – Carl Adam Petri
- The Temporal Logic of Reactive and Concurrent Systems: Specification - Z. Manna and A. Pnueli
- Temporal Verification of Reactive Systems: Safety - Z. Manna and A. Pnueli

CS 501 Industrial experience

Objective: To expose the students to the process of program development so as to enable to learn the "trade/skill of software development" (ideally) in the software industry.

Structure:

Each student has to work for a minimum of 5 months (ideally) in a software company.

CS 502 Seminar

Objective: This course is in line with enhancing the technical communication skills of students, which is a critical requirement in team as well as customer/client interactions.

Structure:

Each student shall be assigned a topic from emerging thrust areas from the state of the art in the discipline and should prepare and submit a write-up on the same and deliver a talk before audience.

CS-601 Programming Paradigms

- GUI Programming: GUI Vs CUI, Event Driven Programming, programing for the web
- Architecture of typical Application: Database Connectivity, codeless programming
- OO Paradigm: Modularity, Data Abstraction, Classes and Objects, Inheritance and interfaces, Polymorphism,
- Notion of Scripting: Scripting via Perl/Guile/Python/ClojureScript
- Functional paradigm: Functions as first class objects, typeclasses, kinds, datakinds, type-level programing, type level programing in Haskell/Agda/Idris,
- HDL via Verilog or VHDL: Architectural behavioral and RT levels, Study of Waveforms

References:

- Modern Java in Action: Lambda, streams, functional and reactive programming (Manning publishers) - Raoul-Gabriel Urma; Mario Fusco; Alan Mycroft (ISBN 9781617293566)
- Real World Haskell (O'Reilly Media) - Bryan O'Sullivan, John Goerzen, Don Stewart
- Reactive with ClojureScript Recipes: Functional Programming for the Web (Apress) - Nicolas Modrzyk (ISBN 9781484230084)
- Functional Reactive Programming (Manning publishers) - Stephen Blackheath Anthony Jones
- Developing Web Applications with Haskell and Yesod (O'Reilly Media) - Michael Snoyman
- Perl by Wall and Chistiansen (O'reilly)
- Thinking in Java Vol 3 by Bruce Eckel
- Programming Python by Mark Lutz, (O'Reilly)
- Verilog HDL by S. Palnitker (Prentice Hall)

Suggested further reading:

- Verified Functional Programming in Agda (ACM Books) - Aaron Stump (ISBN 9781970001242)
- Type-driven Development with Idris (Manning Publications) - Edwin Brady (ISBN 9781617293023)

CS-602 Software Development – 3

Motivation:

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

Objectives:

The broad objective of the course is:

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

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

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

Contents:

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

Note:

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

References:

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