CS2013 Recommended Curriculum

Curricula Hour is a lecture hour. Core Tier 1: should be covered 100%, Core Tier 2, can be covered from 80:100%. Electives are optional.

Knowledge Area (KA)	CS2013		CS2008	CC2001
	Tier1	Tier2	Core	Core
AL-Algorithms and Complexity	19	9	31	31
AR-Architecture and Organization	0	16	36	36
CN-Computational Science	1	0	0	0
DS-Discrete Structures	37	4	43	43
GV-Graphics and Visual Computing	2	1	3	3
HC-Human-Computer Interaction	4	4	8	8
IAS-Security and Information Assurance	2	6	--	--
IM-Information Management	1	9	11	10
IS-Intelligent Systems	0	10	10	10
NC-Networking and Communication	3	7	15	15
OS-Operating Systems	4	11	18	18
PBD-Platform-based Development	0	0	--	--
PD-Parallel and Distributed Computing	5	10	--	--
PL-Programming Languages	8	20	21	21
SDF-Software Development Fundamentals	42	0	47	38
SE-Software Engineering	6	21	31	31
SF-Systems Fundamentals	18	9	--	--
SP-Social and Professional Issues	11	5	16	16
Total Core Hours	**163**	**142**	**290**	**280**
All Tier1 + All Tier2 Total	**305**			
All Tier1 + 90% of Tier2 Total	**290.8**			
All Tier1 + 80% of Tier2 Total	**276.6**			

KA	Core Tier 1	Core Tier 2	Elective
AL (19 Core-Tier1 hours, 9 Core-Tier2 hours)	**BASIC ANALYSIS** [2 hours] • Differences among best, average, and worst case behaviors of an algorithm • Asymptotic analysis of upper and average complexity bounds • Big O notation: formal definition • Complexity classes, such as constant, logarithmic, linear, quadratic, and exponential • Empirical measurements of performance • Time and space trade-offs in algorithms	**BASIC ANALYSIS** [2 hours] • Big O notation: use • Little o, big omega and big theta notation • Recurrence relations and analysis of recursive algorithms • Some version of a Master Theorem	**Advanced Computational Complexity** • Review definitions of the classes P and NP; introduce EXP • NP-completeness (Cook's theorem) • Classic NP-complete problems • Reduction Techniques
	Algorithmic Strategies [5 hours] • Brute-force algorithms • Greedy algorithms • Divide-and-conquer (cross-reference SDF/Algorithms and Design/Problem-solving strategies) • Recursive backtracking • Dynamic Programming	**Algorithmic Strategies** [1 hour] • Branch-and-bound • Heuristics • Reduction: transform-and-conquer	
	Fundamental Data Structures and Algorithms [9 hours] Implementation and use of: • Simple numerical algorithms, such as computing the average of a list of numbers, finding the min, max, and mode in a list, approximating the square root of a number, or finding the greatest common divisor Sequential and binary search algorithms • Worst case quadratic sorting algorithms (selection, insertion)	**Fundamental Data Structures and Algorithms** [3 hours] • Graphs and graph algorithms o Shortest-path algorithms (Dijkstra's and Floyd's algorithms) o Minimum spanning tree (Prim's and Kruskal's algorithms) • Pattern matching and string/text algorithms (e.g., substring matching, regular expression matching, longest common subsequence algorithms)	

	• Worst or average case O(N log N) sorting algorithms (quicksort, heapsort, mergesort) • Hash tables, including strategies for avoiding and resolving collisions • Binary search trees • Common operations on binary search trees such as select min, max, insert, delete, iterate over tree Graphs and graph algorithms • Representations of graphs (e.g., adjacency list, adjacency matrix) o Depth- and breadth-first traversals		
	Basic Automata Computability and Complexity [3 hours] • Finite-state machines • Regular expressions • The halting problem	**Basic Automata Computability and Complexity** [3 hours] • Context-free grammars (cross-reference PL/Syntax Analysis) • P vs. NP (tractable and intractable problems) o Definition of P, NP, and NP-complete o Exemplary NP-complete problems (e.g., SAT, Knapsack)	**Advanced Automata Theory and Computability** • Sets and languages • Regular languages o Review of deterministic finite automata (DFAs) o Nondeterministic finite automata (NFAs) o Equivalence of DFAs and NFAs o Review of regular expressions; their equivalence to finite automata o Closure properties o Proving languages non-regular, via the pumping lemma or alternative means • Context-free languages o Push-down automata (PDAs) o Relationship of PDAs and context-free grammars o Properties of context-free languages • Turing machines, or an equivalent formal model of universal computation • Nondeterministic Turing machines • Chomsky hierarchy • The Church-Turing thesis • Computability • Rice's Theorem • Examples of uncomputable functions • Implications of uncomputability
			Advanced Data Structures Algorithms and Analysis • Balanced trees (e.g., AVL trees, red-black trees, splay trees, treaps) • Graphs (e.g., topological sort, Tarjan's algorithm, matching) • Advanced data structures (e.g., B-trees, tries, Fibonacci heaps) • Network flows (e.g., max flow [Ford-Fulkerson algorithm], max flow – min cut, maximum bipartite matching) • Linear Programming (e.g., duality, simplex method, interior point algorithms) • Number-theoretic algorithms (e.g., modular arithmetic, primality testing, integer factorization) • Geometric algorithms (e.g., points, line segments, polygons [properties, intersections], finding convex hull, spatial decomposition, collision detection, geometric search/proximity) • Randomized algorithms • Approximation algorithms • Amortized analysis • Probabilistic analysis • Online algorithms and competitive analysis
AR (0 Core-Tier 1 hours, 16 Core-Tier 2 hours)		**Digital logic and digital systems** [3 hours] • Overview and history of computer architecture • Combinational vs. sequential logic/Field programmable gate arrays as a fundamental combinational + sequential logic building block • Multiple representations/layers of interpretation (hardware is just another layer) • Computer-aided design tools that process hardware and architectural representations • Register transfer notation/Hardware Description Language (Verilog/VHDL) • Physical constraints (gate delays, fan-in, fan-out, energy/power)	**Functional Organization** [Note: elective for computer scientist; would be core for computer engineering curriculum] • Implementation of simple datapaths, including instruction pipelining, hazard detection and resolution • Control unit: hardwired realization vs. microprogrammed realization • Instruction pipelining • Introduction to instruction-level parallelism (ILP)
		Machine-level representation of data [3 hours] • Bits, bytes, and words • Numeric data representation and number bases • Fixed- and floating-point systems • Signed and twos-complement representations • Representation of non-numeric data (character codes, graphical data) • Representation of records and arrays	**Multiprocessing and alternative architectures** [Cross-reference PD/Parallel Architecture: The view here is on the hardware implementation of SIMD and MIMD architectures; in PD/Parallel Architecture, it is on the way that algorithms can be matched to the underlying hardware capabilities for these kinds of parallel processing architectures.] • Power Law: Energy as a limiting factor in processor design • Example SIMD and MIMD instruction sets and architectures • Interconnection networks (hypercube, shuffle-exchange, mesh, crossbar) • Shared multiprocessor memory systems and memory consistency • Multiprocessor cache coherence
		Assembly level machine organization [6 hours] • Basic organization of the von Neumann machine • Control unit; instruction fetch, decode, and execution	**Performance Enhancements** • Superscalar architecture • Branch prediction, Speculative execution, Out-of-order execution • Prefetching

		• Instruction sets and types (data manipulation, control, I/O) • Assembly/machine language programming • Instruction formats • Addressing modes • Subroutine call and return mechanisms • I/O and interrupts • Heap vs. Static vs. Stack vs. Code segments • Shared memory multiprocessors/multicore organization • Introduction to SIMD vs. MIMD and the Flynn Taxonomy	• Vector processors and GPUs • Hardware support for Multithreading • Scalability • Alternative architectures, such as VLIW/EPIC, and Accelerators and other kinds of Special-Purpose Processors
		Memory system organization and architecture [3 hours] [Cross-reference OS/Memory Management--Virtual Machines] • Storage systems and their technology • Memory hierarchy: importance of temporal and spatial locality • Main memory organization and operations • Latency, cycle time, bandwidth, and interleaving • Cache memories (address mapping, block size, replacement and store policy) • Multiprocessor cache consistency/Using the memory system for inter-core synchronization/atomic memory operations • Virtual memory (page table, TLB) • Fault handling and reliability Coding, data compression, and data integrity	
		Interfacing and communication [1 hour] [Cross-reference OS Knowledge Area for a discussion of the operating system view of input/output processing and management. The focus here is on the hardware mechanisms for supporting device interfacing and processor-to-processor communications.] • I/O fundamentals: handshaking, buffering, programmed I/O, interrupt-driven I/O Interrupt structures: vectored and prioritized, interrupt acknowledgment • External storage, physical organization, and drives • Buses: bus protocols, arbitration, direct-memory access (DMA) • Introduction to networks: networks as another layer of access hierarchy Multimedia support • RAID architectures	
CN (1-Tier1 hours, 0-Tier2 hours)	**CN Fundamentals** [1 hours] • Introduction to modeling and simulation • Simulation techniques and tools, such as physical simulations, human-in-the-loop guided simulations, and virtual reality. • Foundational approaches to validating models		**Modeling and Simulation** • Purpose of modeling and simulation including optimization; supporting decision making, forecasting, safety considerations; for training and education. • Tradeoffs including performance, accuracy, validity, and complexity. • The simulation process; identification of key characteristics or behaviors, simplifying assumptions; validation of outcomes. • Model building: use of mathematical formula or equation, graphs, constraints; methodologies and techniques; use of time stepping for dynamic systems. • Formal models and modeling techniques: mathematical descriptions involving simplifying assumptions and avoiding detail. The descriptions use fundamental mathematical concepts such as set and function. Random numbers. Examples of techniques including: o Monte Carlo methods o Stochastic processes o Queuing theory o Petri nets and colored Petri nets o Graph structures such as directed graphs, trees, networks o Games, game theory, the modeling of things using game theory o Linear programming and its extensions o Dynamic programming o Differential equations: ODE, PDE o Non-linear techniques o State spaces and transitions • Assessing and evaluating models and simulations in a variety of contexts; verification and validation of models and simulations. • Important application areas including health care and diagnostics, economics and finance, city and urban planning, science, and engineering. • Software in support of simulation and modeling; packages, languages.`
			Processing • Fundamental programming concepts: o The concept of an algorithm consisting of a finite number of well-defined steps, each of which completes in a finite amount of time, as does the entire process. o Examples of well-known algorithms such as sorting and searching. o The concept of analysis as understanding what the problem is really asking, how a problem can be approached using an algorithm, and how information is represented so that a machine can process it.

			○ The development or identification of a workflow. ○ The process of converting an algorithm to machine-executable code. ○ Software processes including lifecycle models, requirements, design, implementation, verification and maintenance. ○ Machine representation of data computer arithmetic, and numerical methods, specifically sequential and parallel architectures and computations. • Fundamental properties of parallel and distributed computation: ○ Bandwidth. ○ Latency. ○ Scalability. ○ Granularity. ○ Parallelism including task, data, and event parallelism. ○ Parallel architectures including processor architectures, memory and caching. ○ Parallel programming paradigms including threading, message passing, event driven techniques, parallel software architectures, and MapReduce. ○ Grid computing. ○ The impact of architecture on computational time. ○ Total time to science curve for parallelism: continuum of things. • Computing costs, e.g., the cost of re-computing a value vs. the cost of storing and lookup.
			Interactive Visualization • Principles of data visualization. • Graphing and visualization algorithms. • Image processing techniques. • Scalability concerns.
			Data, Information, and Knowledge • Content management models, frameworks, systems, design methods (as in IM. Information Management). • Digital representations of content including numbers, text, images (e.g., raster and vector), video (e.g., QuickTime, MPEG2, MPEG4), audio (e.g., written score, MIDI, sampled digitized sound track) and animations; complex/composite/aggregate objects; FRBR. • Digital content creation/capture and preservation, including digitization, sampling, compression, conversion, transformation/translation, migration/emulation, crawling, harvesting. • Content structure / management, including digital libraries and static/dynamic/stream aspects for: • Data: data structures, databases. • Information: document collections, multimedia pools, hyperbases (hypertext, hypermedia), catalogs, repositories. • Knowledge: ontologies, triple stores, semantic networks, rules. • Processing and pattern recognition, including indexing, searching (including: queries and query languages; central / federated / P2P), retrieving, clustering, classifying/categorizing, analyzing/mining/extracting, rendering, reporting, handling transactions. • User / society support for presentation and interaction, including browse, search, filter, route, visualize, share, collaborate, rate, annotate, personalize, recommend. • Modeling, design, logical and physical implementation, using relevant systems/software.
DS (37 Core-Tier1 hours, 4 Core-Tier2 `hours)	**Sets, Relations, and Functions** [4 hours] • Sets ○ Venn diagrams ○ Union, intersection, complement ○ Cartesian product ○ Power sets ○ Cardinality of finite sets • Relations ○ Reflexivity, symmetry, transitivity ○ Equivalence relations, partial orders • Functions ○ Surjections, injections, bijections ○ Inverses ○ Composition		
	Basic Logic [9 hours] • Propositional logic (cross-reference: Propositional logic is also reviewed in IS/Knowledge Based Reasoning) • Logical connectives • Truth tables • Normal forms (conjunctive and disjunctive) • Validity		

	• Propositional inference rules (concepts of modus ponens and modus tollens) • Predicate logic o Universal and existential quantification • Limitations of propositional and predicate logic (e.g., expressiveness issues)		
	Proof Techniques [10 hours] • Notions of implication, equivalence, converse, inverse, contrapositive, negation, and contradiction • The structure of mathematical proofs • Direct proofs • Disproving by counterexample • Proof by contradiction • Induction over natural numbers • Structural induction • Weak and strong induction (i.e., First and Second Principle of Induction) • Recursive mathematical definitions	**Proof Techniques** [1 hour] • Well orderings	
	Basics of Counting [5 hours] • Counting arguments o Set cardinality and counting o Sum and product rule o Inclusion-exclusion principle o Arithmetic and geometric progressions • The pigeonhole principle • Permutations and combinations o Basic definitions o Pascal's identity o The binomial theorem • Solving recurrence relations (cross-reference: AL/Basic Analysis) o An example of a simple recurrence relation, such as Fibonacci numbers o Other examples, showing a variety of solutions • Basic modular arithmetic		
	Graphs and Trees [3 hours] • Trees • Undirected graphs • Directed graphs • Weighted graphs • Traversal strategies	**Graphs and Trees** [1 hour] • Spanning trees/forests • Graph isomorphism	
	Discrete Probability [6 hours] • Finite probability space, events • Axioms of probability and probability measures • Conditional probability, Bayes' theorem • Independence • Integer random variables (Bernoulli, binomial) • Expectation, including Linearity of Expectation	**Discrete Probability** [2 hour] • Variance • Conditional Independence	
GV (2 Core-Tier1 hours, 1 Core-Tier2 hours)	**Fundamental Concepts** [2 hours] • Basics of Human visual perception (HCI Foundations). • Image representations, vector vs. raster, color models, meshes. • Forward and backward rendering (i.e., ray-casting and rasterization). • Applications of computer graphics: including game engines, cad, visualization, virtual reality.	**Fundamental Concepts** [1 hours] • Polygonal representation. • Basic radiometry, similar triangles, and projection model. • Use of standard graphics APIs (see HCI GUI construction). • Compressed image representation and the relationship to information theory. • Immediate and retained mode. • Double buffering.	**Basic Rendering** • Rendering in nature, i.e., the emission and scattering of light and its relation to numerical integration. • Affine and coordinate system transformations. • Ray tracing. • Visibility and occlusion, including solutions to this problem such as depth buffering, Paiter's algorithm, and ray tracing. • The forward and backward rendering equation. • Simple triangle rasterization. • Rendering with a shader-based API. • Texture mapping, including minification and magnification (e.g., trilinear MIP-mapping). • Application of spatial data structures to rendering. • Sampling and anti-aliasing. Scene graphs and the graphics pipeline.
			Geometric Modeling • Basic geometric operations such as intersection calculation and proximity tests • Volumes, voxels, and point-based representations. • Parametric polynomial curves and surfaces.

			• Implicit representation of curves and surfaces. • Approximation techniques such as polynomial curves, Bezier curves, spline curves and surfaces, and non- uniform rational basis (NURB) spines, and level set method. • Surface representation techniques including tessellation, mesh representation, mesh fairing, and mesh generation techniques such as Delaunay triangulation, marching cubes, . • Spatial subdivision techniques. • Procedural models such as fractals, generative modeling, and L-systems. • Graftals, cross referenced with programming languages (grammars to generated pictures). • Elastically deformable and freeform deformable models. • Subdivision surfaces. • Multiresolution modeling. • Reconstruction. • Constructive Solid Geometry (CSG) representation.
			Advanced Rendering • Solutions and approximations to the rendering equation, for example: o Distribution ray tracing and path tracing o Photon mapping o Bidirectional path tracing o Reyes (micropolygon) rendering o Metropolis light transport • Considering the dimensions of time (motion blur), lens position (focus), and continuous frequency (color). • Shadow mapping. • Occlusion culling. • Bidirectional Scattering Distribution function (BSDF) theory and microfacets. • Subsurface scattering. • Area light sources. • Hierarchical depth buffering. • The Light Field, image-based rendering. • Non-photorealistic rendering. • GPU architecture. • Human visual systems including adaptation to light, sensitivity to noise, and flicker fusion.
			Computer Animation • Forward and inverse kinematics. • Collision detection and response • Procedural animation using noise, rules (boids/crowds), and particle systems. • Skinning algorithms. • Physics based motions including rigid body dynamics, physical particle systems, mass-spring networks for cloth and flesh and hair. • Key-frame animation. • Splines. • Data structures for rotations, such as quaternions. • Camera animation. • Motion capture.
			Visualization • Visualization of 2D/3D scalar fields: color mapping, isosurfaces. • Direct volume data rendering: ray-casting, transfer functions, segmentation. • Visualization of: o Vector fields and flow data o Time-varying data o High-dimensional data: dimension reduction, parallel coordinates, o Non-spatial data: multi-variate, tree/graph structured, text • Perceptual and cognitive foundations that drive visual abstractions. • Visualization design. • Evaluation of visualization methods. • Applications of visualization.
HC (4 Core-Tier1 hours, 4 Core-Tier2 hours)	**Foundations** [4 hours] • Contexts for HCI (anything with a user interface: webpage, business applications, mobile applications, games, etc.) • Processes for user-centered development: early focus on users, empirical testing, iterative design. • Different measures for evaluation: utility, efficiency, learnability, user satisfaction. • Physical capabilities that inform interaction design: color perception, ergonomics • Cognitive models that inform interaction design: attention, perception and recognition, movement, and memory. Gulfs of expectation and execution. • Social models that inform interaction design: culture, communication, networks and	**Designing Interaction** [4 hours] • Principles of different styles of interface: e.g. command line, graphical tangible. • Basic two-dimensional design fundamentals as applied to the visual interface, including use of grid, typography, color and contrast, scale, ordering and hierarchy.) • Task analysis • Paper prototyping • Basic statistics and techniques for controlled experimentation (especially in regard to web data) • KLM evaluation • Help & documentation	**Programming Interactive Systems** • Software Architecture Patterns: Model-View controller; command objects, online, offline, (cross reference SE/Software Design) • Interaction Design Patterns: visual hierarchy, navigational distance • Event management and user interaction • Geometry management (cross reference GV/Geometric Modeling*)* • Choosing interaction styles and interaction techniques • Presenting information: navigation, representation, manipulation • Interface animation techniques (scene graphs, etc) • Widget classes and libraries • Modern GUI libraries (iOS, Android, JavaFX) GUI builders and UI programming environments

	organizations. • Principles of good design and good designers; engineering tradeoffs • Accessibility: interfaces for differently-abled populations (e.g. blind, motion-impaired) • Interfaces for differently-aged population groups (e.g. children, 80+)	• Handling human/system failure User interface standards	(cross reference to PBD/Mobile Platforms) • Declarative Interface Specification: Stylesheets and DOMs • Data-driven applications (database-backed web pages) • Cross-platform design Design for resource-constrained devices (e.g. small, mobile devices)
			User-centered design and testing • Approaches and characteristics of design process • Functionality and usability requirements (cross reference to SE Software Design) • Techniques for gathering requirements: interviews, surveys, ethnographic & contextual enquiry (cross reference to SE Requirements Engineering) • Techniques and tools for analysis & presentation of requirements: reports, personas • Prototyping techniques and tools: sketching, storyboards, low-fidelity prototyping, wireframes • Evaluation without users, using both qualitative and quantitative techniques: walkthroughs, GOMS, expert- based analysis, heuristics, guidelines, and standards • Evaluation with users: observation, think-aloud, interview, survey, experiment. • Challenges to effective evaluation: sampling, generalization. • Reporting the results of evaluations • Internationalization, designing for users from other cultures, cross-cultural evaluation
			Design for non-mouse interfaces • Choosing interaction styles and interaction techniques • Representing information to users: navigation, representation, manipulation • Approaches to design, implementation and evaluation of non-mouse interaction o Touch and multi-touch interfaces o New Windows (iPhone, Android) o Speech recognition and natural language processing – (cross reference IS/Perception and Computer Vision) o Wearable and tangible interfaces o Persuasive interaction and emotion o Ubiquitous and context-aware (Ubicomp) o Bayesian inference (e.g. predictive text, guided pointing) o Ambient/peripheral display and interaction
			Collaboration and Communication • Asynchronous group communication: e-mail, forums, Facebook • Synchronous group communication: chat rooms, conferencing, online games • Online communities • Software characters and intelligent agents, virtual worlds and avatars (cross reference IS/Agents) • Social psychology • Social networking • Social computing
			Statistical methods for HCI • t-tests • ANOVA • randomization (non-parametric) testing, within v. between-subjects design • calculating effect size • exploratory data analysis • presenting statistical data • using statistical data • using qualitative and quantitative results together
			Human factors and security • Applied psychology and security policies • Security economics • Regulatory environments – responsibility, liability and self-determination • Organizational vulnerabilities and threats • Usability design and security • Pretext, impersonation and fraud. Phishing and spear phishing (cross reference IAS/Fundamentals) • Trust, privacy and deception • Biometric authentication (camera, voice) • Identity management`
			Design-oriented HCI • Intellectual styles and perspectives to technology and its interfaces • Consideration of HCI as a design discipline: o Sketching o Participatory design • Critically reflective HCI

			○ Critical technical practice ○ Technologies for political activism ○ Philosophy of user experience ○ Ethnography and ethno-methodology • Indicative domains of application ○ Sustainability ○ Arts-informed computing
			Mixed, Augmented and Virtual Reality • Output ○ Sound ○ Stereoscopic display ○ Force feedback simulation, haptic devices • User input ○ Viewer and object tracking ○ Pose and gesture recognition ○ Accelerometers ○ Fiducial markers ○ User interface issues • Physical modeling and rendering ○ Physical simulation: collision detection & response, animation ○ Visibility computation ○ Time-critical rendering, multiple levels of details (LOD) • System architectures ○ Game engines ○ Mobile augmented reality ○ Flight simulators ○ CAVEs ○ Medical imaging • Networking ○ p2p, client-server, dead reckoning, encryption, synchronization o ○ Distributed collaboration
IAS (2 Core-Tier1 hours, 6 Core-Tier2 hours)	**Fundamental Concepts** [1 hours] • Nature of the Threats • Need for Information Assurance. • Basic Terminology that should be recognized by those studying the field. (Confidentiality, Integrity, Availability) • Information Assurance Concepts that are key to building an understanding of the IA area.	**Fundamental Concepts** [2 hours] • Industry and Government Guidelines and Standards concerning Information Assurance. • National and Cultural Differences including topics such as HIPAA, Safe Harbor, and data protection laws. • Legal, Ethical, and Social Issues (cross reference with SP KA) • Threats and Vulnerabilities. • Types of Attacks • Types of Attackers. • Defense Mechanisms. • Incident Response.	**Cryptography** • The Basic Cryptography Terminology covers notions pertaining to the different (communication) partners, secure/unsecure channel, attackers and their capabilities, encryption, decryption, keys and their characteristics, signatures, etc. • Cipher types:, Caesar cipher, affine cipher, etc. together with typical attack methods such as frequency analysis, etc. • Mathematical Preliminaries; include topics in linear algebra, number theory, probability theory, and statistics. (Discrete Structures) • Cryptographic Primitives include encryption (stream ciphers, block ciphers public key encryption), digital signatures, message authentication codes, and hash functions. • Cryptanalysis covers the state-of-the-art methods including differential cryptanalysis, linear cryptanalysis, factoring, solving discrete logarithm problem, lattice based methods, etc. • Cryptographic Algorithm Design covers principles that govern the design of the various cryptographic primitives, especially block ciphers and hash functions. (Algorithms and Complexity - Hash functions) • The treatment of Common Protocols includes (but should not be limited to) current protocols such as RSA, DES, DSA, AES, ElGamal, MD5, SHA-1, Diffie-Hellman Key exchange, identification and authentication protocols, secret sharing, multi-party computation, etc. Public Key Infrastructure deals with challenges, opportunities, local infrastructures, and national infrastructure.
	Network Security [1 hour] • Application of Cryptography ○ TLS ○ Secret-key algorithms ○ Public-key algorithms ○ Hybrid	**Network Security** [4 hours] • Network attack types: Denial of service, flooding, sniffing and traffic redirection, message integrity attacks, • Identity hijacking, exploit attacks (buffer overruns, Trojans, backdoors), inside attacks, infrastructure (DNS hijacking, route blackholing, misbehaving routers that drop traffic), etc.) • Authentication protocols • Digital signatures • Message Digest • Defense Mechanisms /Countermeasures. (Intrusion Detection, Firewalls, Detection of malware, IPSec, Virtual Private Networks, Network Address Translation.) • Network Auditing.	**Risk Management** • Risk Analysis involves identifying the assets, probable threats, vulnerabilities and control measures to discern risk levels and likelihoods. It can be applied to a program, organization, sector, etc. Knowledge in this area includes knowing different risk analysis models and methods, their strengths and benefits and the apropriateness of the different methods and models given the situation. This includes periodic reassessment. • Cost/Benefit Analysis is used to weigh private and/or public costs versus benefits and can be applied to security policies, investments, programs, tools, deployments, etc. • Continuity Planning will help organizations deliver critical services and ensure survival. • Disaster Recovery will help an organization continue normal operations in a minimum amount of time with a minimum amount of disruption and cost. • Security Auditing: a systematic assessment of an organization's system measuring the conformity vis-àvis a set of pre-established criteria. • Asset Management minimizes the life cost of assets and includes critical factors such as risk or business continuity. Risk communication Enforcement of risk management policies is critical for an organization.
			Security Policy and Governance • Strategies and Plans for creating security policies.

			• Policies, Guidelines, Standards and Best Practices for individuals or organizations, including national security policies. ○ Procedures for creating policies, guidelines, standards, specifications, regulations and laws. ○ Privacy Policies to help protect personal and other sensitive information. • Compliance and Enforcement of policies, standards, regulations, and laws. • Formal Policy Models such as Bell-LaPadula, Biba and Clark-Wilson, which provide precise specifications of security objectives. • Relation of national security policies, regulations, organizational security policies, formal policy models, and policy languages. • Policy as related to Risk Aversion.
			Digital Forensics • Basic Principles and methodologies for digital forensics. • Rules of Evidence – general concepts and differences between jurisdictions and Chain of Custody. • Search and Seizure of evidence, e.g., computers, including search warrant issues. • Digital Evidence methods and standards. • Techniques and standards for Preservation of Data. • Data analysis and validation. • Legal and Reporting Issues including working as an expert witness. • OS/File System Forensics • Application Forensics • Network Forensics • Mobile Device Forensics • Computer/network/system attacks.
			Security Architecture and Systems Administration • How to secure Hardware, including how to make hardware tokens and chip cards tamper-proof and tamper- resistance. • Configuring systems to operate securely as an IT system. • Access Control ○ Basic Principles of an access control system prevent unauthorized access. ○ Physical Access Control determines who is allowed to enter or exit, where the user is allowed to enter or exit, and when the user is allowed to enter or exit. ○ Technical/System Access Control is the process of preventing unauthorized users or services to utilize information systems. • Usability includes the difficulty for humans to deal with security (e.g., remembering PINs), social engineering, phishing, and other similar attacks. • Analyzing and identifying System Threats and Vulnerabilities • Investigating Operating Systems Security for various systems. • Multi-level/Multi-lateral Security • Design and Testing for architectures and systems of different scale ○ Penetration testing in the system setting ○ Products available in the marketplace • Supervisory Control and Data Acquisition (SCADA) ○ SCADA system uses. Communications protocols supporting data acquisition ○ Communications protocols supporting distributed control. ○ Data Integrity • Data Confidentiality
			Secure Software Design and Engineering • Building security into the Software Development Lifecycle • Secure Design Principles and Patterns (Saltzer and Schroeder, etc) • Secure Software Specification and Requirements deals with specifying what the program should and should not do, which can be done either using a requirements document or using a more formal mathematical specification. • Secure Coding involves applying the correct balance of theory and practice to minimize vulnerabilities in code. ○ Data validation ○ Memory handling ○ Crypto implementation • Secure Testing is the process of testing that security requirements are met (including Static and Dynamic analysis). • Program Verification and Simulation is the process of ensuring that a certain version of a certain implementation meets the required security goals, either by a mathematical proof or by simulation.
IM (1 Core-	**Information Management Concepts** [1 hour] • Basic information storage and retrieval (IS&R) concepts • Information capture and representation	**Information Management Concepts** [2 hours] • Information management applications • Declarative and navigational queries, use of links	**Indexing** • The impact of indexes on query performance • The basic structure of an index; [Robert: Not sure if this warrants a topic by itself] • Keeping a buffer of data in memory; [Robert: Why is this listed as a topic?]

Tier1 hour; 9 Core-Tier2 hours)	• Supporting human needs: Searching, retrieving, linking, browsing, navigating	• Analysis and indexing • Quality issues: Reliability, scalability, efficiency, and effectiveness	• Creating indexes with SQL • Indexing text Indexing the web (how search engines work)
		Database Systems [3 hours] • Approaches to and evolution of database systems • Components of database systems • DBMS functions • Database architecture and data independence • Use of a declarative query language • Systems supporting structured and/or stream content	**Relational Databases** • Mapping conceptual schema to a relational schema • Entity and referential integrity • Relational algebra and relational calculus • Relational Database design • Functional dependency • Decomposition of a schema; lossless-join and dependency-preservation properties of a decomposition • Candidate keys, superkeys, and closure of a set of attributes • Normal forms (1NF, 2NF, 3NF, BCNF) • Multi-valued dependency (4NF) • Join dependency (PJNF, 5NF) Representation theory
		Data Modeling [4 hours] • Data modeling • Conceptual models (e.g., entity-relationship and UML diagrams) • Relational data model • Object-oriented model • Semi-structured data model (expressed using DTD or XML Schema, for example)	**Query Languages** • Overview of database languages • SQL (data definition, query formulation, update sublanguage, constraints, integrity) • QBE and 4th-generation environments • Embedding non-procedural queries in a procedural language • Introduction to Object Query Language Stored procedures
			Transaction Processing • Transactions • Failure and recovery • Concurrency control
			Distributed Databases • Distributed data storage • Distributed query processing • Distributed transaction model • Concurrency control • Homogeneous and heterogeneous solutions • Client-server distributed databases (cross-reference SF/Computational Paradigms)
			Physical Database Design • Storage and file structure • Indexed files • Hashed files • Signature files • B-trees • Files with dense index • Files with variable length records • Database efficiency and tuning
			Data Mining • The usefulness of data mining • Data mining algorithms • Associative and sequential patterns • Data clustering • Market basket analysis • Data cleaning • Data visualization
			Information Storage and Retrieval • Characters, strings, coding, text • Documents, electronic publishing, markup, and markup languages • Tries, inverted files, PAT trees, signature files, indexing • Morphological analysis, stemming, phrases, stop lists • Term frequency distributions, uncertainty, fuzziness, weighting • Vector space, probabilistic, logical, and advanced models • Information needs, relevance, evaluation, effectiveness • Thesauri, ontologies, classification and categorization, metadata • Bibliographic information, bibliometrics, citations • Routing and (community) filtering • Search and search strategy, multimedia search, information seeking behavior, user modeling, feedback • Information summarization and visualization • Integration of citation, keyword, classification scheme, and other terms • Protocols and systems (including Z39.50, OPACs, WWW engines, research systems)

			• Digital libraries ○ Digitization, storage, interchange, digital objects, composites, and packages ○ Metadata, cataloging, author submission ○ Naming, repositories, archives ○ Spaces (conceptual, geographical, 2/3D, VR) ○ Architectures (agents, buses, wrappers/mediators), interoperability ○ Services (searching, linking, browsing, and so forth) ○ Intellectual property rights management, privacy, and protection (watermarking) • Archiving and preservation, integrity
IS (10 Core-Tier2 hours)		**Fundamental Issues** [1 hours] • Overview of AI problems, Examples of successful recent AI applications • What is intelligent behavior? ○ The Turing test ○ Rational versus non-rational reasoning ○ Nature of human reasoning • Nature of environments ○ Fully versus partially observable ○ Single versus multi-agent ○ Deterministic versus stochastic ○ Episodic versus sequential ○ Static versus dynamic ○ Discrete versus continuous • Nature of Agents ○ Autonomous versus Semi-Autonomous ○ Reflexive, Goal-based, and Utility-based ○ The importance of perception and environmental interactions • Philosophical and ethical issues [elective]	**Advanced Search** • Constructing search trees, dynamic search space, combinatorial explosion of search space • Stochastic search • Simulated annealing • Genetic algorithms • Implementation of A* search, Beam search • Minimax Search, Alpha-beta pruning Expectimax search (MDP-solving) and chance nodes
		Basic Search Strategies [4 hours] • Problem spaces (states, goals and operators), problem solving by search • Factored representation (factoring state into variables) • Uninformed search (breadth-first, depth-first, depth-first with iterative deepening) • Heuristics and informed search (hill-climbing, generic best-first, A*) • Space and time efficiency of search • Two-player games (Introduction to minimax search) • Constraint satisfaction (backtracking and local search methods)	**Advanced Representation and Reasoning** • Knowledge representation issues ○ Description logics ○ Ontology engineering • Non-monotonic reasoning ○ Non-classical logics ○ Default reasoning ○ Belief revision ○ Preference logics ○ Integration of knowledge sources ○ Aggregation of conflicting belief • Reasoning about action and change ○ Situation calculus ○ Event calculus ○ Ramification problems • Temporal and spatial reasoning • Rule-based Expert Systems • Model-based and Case-based reasoning • Planning: ○ Partial and totally ordered planning ○ Plan graphs ○ Hierarchical planning ○ Planning and execution including conditional planning and continuous planning Mobile agent/Multi-agent planning
		Basic Knowledge Representation and Reasoning [3 hours] • Review of propositional and predicate logic (cross-reference DS/Basic Logic) • Resolution and theorem proving, unification and lifting (propositional logic only) • Forward chaining, backward chaining • Review of probabilistic reasoning, Bayes theorem (cross-reference with DS/Discrete Probability)	**Reasoning Under Uncertainty** • Review of basic probability (cross-reference DS/Discrete Probability) ○ Unconditional/prior probabilities ○ Conditional/posterior probabilities • Random variables and probability distributions ○ Axioms of probability ○ Probabilistic inference ○ Bayes' Rule • Conditional Independence Knowledge representations ○ Bayesian Networks ▪ Exact inference and its complexity ▪ Randomized sampling (Monte Carlo) methods (e.g. Gibbs sampling) ○ Markov Networks ○ Relational probability models ○ Hidden Markov Models • Decision Theory ○ Preferences and utility functions

			Maximizing expected utility

		Basic Machine Learning
		[2 hours]
		• Definition and examples of machine learning for classification
		• Inductive learning
		• Simple statistical-based learning such as Naive Bayesian Classifier, Decision trees
		• Define overfitting problem
		• Measuring classifier accuracy
			Agents
			• Definitions of agents
			• Agent architectures
			  o Simple reactive agents
			  o Reactive planners
			  o Layered architectures
			  o Cognitive architectures
			  o Integrated architecture
			  o Example architectures and applications
			• Agent theory
			• Rationality, Game Theory
			  o Commitments
			  o Intentions
			  o Decision-theoretic agents
			  o Markov decision processes (MDP)
			• Software agents, personal assistants, and information access
			  o Collaborative agents
			  o Information-gathering agents
			  o Believable agents (synthetic characters, modeling emotions in agents)
			• Learning agents
			• Multi-agent systems
			  o Collaborating agents
			  o Agent teams
			  o Competitive agents
			    ▪ Game theory
			    ▪ Voting
			    ▪ Auctions
			Swarm systems and biologically inspired models

			Natural Language Processing
			• Deterministic and stochastic grammars
			• Parsing algorithms
			  o CFGs and chart parsers (e.g. CYK)
			  o Probabilistic CFGs and weighted CYK Representing meaning / Semantics
			• Logic-based knowledge representations
			  o Semantic roles
			  o Temporal representations
			  o Verbs and event types
			  o Beliefs, desires, and intentions
			  o Ambiguity
			  o Long-distance dependencies
			• Corpus-based methods
			• N-grams and HMMs
			• Smoothing and backoff
			• Perplexity
			• Zipf's law
			• Examples of use: POS tagging and morphology
			• Information retrieval (Cross-reference IM/Information Storage and Retrieval)
			  o Vector space model
			    ▪ TF & IDF
			  o Precision and recall
			• Information extraction
			• Language translation
			• Transfer-based models
			• Statistical, phrase-based models
			• Text classification, categorization
			• Bag of words model

			Advanced Machine Learning
			• Definition and examples of broad variety of machine learning tasks
			• General statistical-based learning, parameter estimation (maximum likelihood)
			• Inductive logic programming (ILP)
			• Supervised learning
			  o Learning decision trees
			  o Learning neural networks
			  o Support vector machines (SVMs)
			• Ensembles
			• Nearest-neighbor algorithms
			• Unsupervised Learning and clustering

			o EM o K-means o Self-organizing maps • Semi-supervised learning • Learning graphical models (Cross-reference IS/Reasoning under Uncertainty) • Performance evaluation (such as cross-validation, area under ROC curve) • Learning theory • The problem of overfitting, the curse of dimensionality • Reinforcement learning o Exploration vs. exploitation trade-off o Markov decision processes o Value and policy iteration o Application of Machine Learning algorithms to Data Mining (Cross-reference IM/Data Mining)
			Robotics • Overview: problems and progress o State-of-the-art robot systems, including their sensors and an overview of their sensor processing o Robot control architectures, e.g., deliberative vs. reactive control and Braitenberg vehicles o World modeling and world models o Inherent uncertainty in sensing and in control • Configuration space and environmental maps • Interpreting uncertain sensor data • Localizing and mapping • Navigation and control • Motion planning o Multiple-robot coordination
			Perception and Computer Vision • Computer vision o Image acquisition, representation, and properties o Image pre-processing via linear and nonlinear filtering o Foreground/background segmentation o Shape representation and object recognition o Image inference based on prior models, i.e., image understanding o Motion analysis • Other modes of sensing o Audio and speech recognition o Sensory transformations • Modularity in recognition o Raw signals, acquisition issues, and sources of noise o Task-independent features, e.g., image edges or phonetic frames o Percepts as collections of features, e.g., edge-based contours or word-level hypotheses o Task-dependent features and percepts: the importance and use of prior models • Approaches to pattern recognition [overlapping with machine learning] o o Classification algorithms and measures of classification quality o Statistical techniques
NC (3 Core-Tier1 hours, 7 Core-Tier2 hours)	**Introduction** [1.5 hours] • Organization of the Internet (Internet Service Providers, Content Providers, etc.) • Switching techniques (Circuit, packet, etc.) • Physical pieces of a network (hosts, routers, switches, ISPs, wireless, LAN, access point, firewalls, etc.) • Layering principles (encapsulation, multiplexing) • Roles of the different layers (application, transport, network, datalink, physical) •	**Reliable Data Delivery** [2 hours] • Error control (retransmission techniques, timers) • Flow control (acknowledgements, sliding window) • Performance issues (pipelining) TCP	
	Networked Applications [1.5 hours] • Naming and address schemes (DNS, IP addresses, Uniform Resource Identifiers, etc.) • Distributed applications (client/server, peer-to-peer, cloud, etc.) • HTTP as an application layer protocol • Multiplexing with TCP and UDP • Socket APIs	**Routing And Forwarding** [1.5 hours] • Routing versus forwarding • Static routing • Internet Protocol (IP) Scalability issues (hierarchical addressing)	
		Local Area Networks [1.5 hours] • Multiple Access • Local Area Networks	

		• Ethernet • Switching	
		Resource Allocation [1 hour] • Need for resource allocation • Fixed allocation (TDM, FDM, WDM) versus dynamic allocation • End-to-end versus network assisted approaches • Fairness • Principles of congestion control	
		Mobility [1 hour] • Principles of cellular networks • 802.11 networks • Issues in supporting mobile nodes (home agents)	
OS (4 Core-Tier1 hours; 11 Core Tier2 hours)	**Overview of Operating Systems** [2 hours] • Role and purpose of the operating system • Functionality of a typical operating system • Mechanisms to support client-server models, hand-held devices • Design issues (efficiency, robustness, flexibility, portability, security, compatibility) • Influences of security, networking, multimedia, windows	**Concurrency** [3 hours] • States and state diagrams (cross reference SF/State-State Transition-State Machines) • Structures (ready list, process control blocks, and so forth) • Dispatching and context switching • The role of interrupts • Managing atomic access to OS objects • Implementing synchronization primitives • Multiprocessor issues (spin-locks, reentrancy) (cross reference SF/Parallelism)	**Virtual Machines** • Types of virtualization (Hardware/Software, OS, Server, Service, Network, etc.) • Paging and virtual memory • Virtual file systems • Virtual file • Hypervisors • Portable virtualization; emulation vs. isolation • Cost of virtualization
	Operating System Principles [2 hours] • Structuring methods (monolithic, layered, modular, micro-kernel models) • Abstractions, processes, and resources • Concepts of application program interfaces (APIs) • Application needs and the evolution of hardware/software techniques • Device organization • Interrupts: methods and implementations • Concept of user/system state and protection, transition to kernel mode	**Scheduling and Dispatch** [3 hours] • Preemptive and nonpreemptive scheduling (cross reference SF/Resource Allocation and Scheduling, PD/Parallel Performance) • Schedulers and policies (cross reference SF/Resource Allocation and Scheduling, PD/Parallel Performance) • Processes and threads (cross reference SF/computational paradigms) • Deadlines and real-time issues	**Device Management** • Characteristics of serial and parallel devices • Abstracting device differences • Buffering strategies • Direct memory access • Recovery from failures
		Memory Management [3 hours] • Review of physical memory and memory management hardware • Working sets and thrashing • Caching	**File Systems** • Files: data, metadata, operations, organization, buffering, sequential, nonsequential • Directories: contents and structure • File systems: partitioning, mount/unmount, virtual file systems • Standard implementation techniques • Memory-mapped files • Special-purpose file systems • Naming, searching, access, backups • Journaling and log-structured file systems
		Security and Protection [2 hours] • Overview of system security • Policy/mechanism separation • Security methods and devices • Protection, access control, and authentication • Backups	**Real Time and Embedded Systems** • Process and task scheduling • Memory/disk management requirements in a real-time environment • Failures, risks, and recovery • Special concerns in real-time systems
			Fault Tolerance • Fundamental concepts: reliable and available systems (cross reference SF/Reliability through Redundancy) • Spatial and temporal redundancy (cross reference SF/Reliability through Redundancy) • Methods used to implement fault tolerance Examples of OS mechanisms for detection, recovery, restart to implement fault tolerance, use of these techniques for the OS's own services
			System Performance Evaluation • Why system performance needs to be evaluated (cross reference SF/Performance/Figures of performance merit) • What is to be evaluated (cross reference SF/Performance/Figures of performance merit) • Policies for caching, paging, scheduling, memory management, security, and so forth • Evaluation models: deterministic, analytic, simulation, or implementation-specific • How to collect evaluation data (profiling and tracing mechanisms)
PBD (Elect-			**Introduction** • Overview of platforms (Web, Mobile, Game, Industrial etc) • Programming via platform-specific APIs

ive)			• Overview of Platform Languages (Objective C, HTML5, etc) • Programming under platform constraints
			Web Platforms • Web programming languages (HTML5, Java Script, PHP, CSS, etc.) • Web platform constraints • Software as a Service (SaaS)
			Mobile Platforms • Mobile Programming Languages (Objective C, Java Script, Java, etc.) • Challenges with mobility and wireless communication • Location-aware applications • Performance / power tradeoffs • Mobile platform constraints • Emerging Technologies
			Industrial Platforms • Types of Industrial Platforms (Mathematic, Robotics, Industrial Controls, etc.) • Robotic Software and its Architecture • Domain Specific Languages • Industrial Platform Constraints
			Game Platforms • Types of Game Platforms (XBox, Wii, PlayStation, etc) • Game Platform Languages (C++, Java, Lua, Python, etc) • Game Platform Constraints
PD (5 Core-Tier1 hours, 9 Core-Tier2 hours)	**Parallelism Fundamentals** [2 hours] • Multiple simultaneous computations • Goals of parallelism (e.g., throughput) versus concurrency (e.g., controlling access to shared resources) • Programming constructs for creating parallelism, communicating, and coordinating • Programming errors not found in sequential programming o Data races (simultaneous read/write or write/write of shared state) o Higher-level races (interleavings violating program intention) o Lack of liveness/progress (deadlock, starvation)	**Parallel Algorithms, Analysis, and Programming** [3 hours] • Critical paths, work and span, and the relation to Amdahl's law (cross-reference SF/Performance) • Speed-up and scalability • Naturally (embarassingly) parallel algorithms • Parallel algorithmic patterns (divide-and-conquer, map and reduce, others) • Specific algorithms (e.g., parallel MergeSort)	**Parallel Algorithms, Analysis, and Programming** • Parallel graph algorithms (e.g., parallel shortest path, parallel spanning tree) (cross-reference AL/Algorithmic Strategies/Divide-and-conquer) • Producer-consumer and pipelined algorithms
	Parallel Decomposition [1 hour] • Need for communication and coordination/synchronization • Independence and partitioning	**Parallel Decomposition** [3 hours] • Basic knowledge of parallel decomposition concepts (cross-reference SF/System Support for Parallelism) • Task-based decomposition o Implementation strategies such as threads • Data-parallel decomposition o strategies such as SIMD and MapReduce • Actors and reactive processes (e.g., request handlers)	**Parallel Performance** • Load balancing • Performance measurement • Scheduling and contention (cross-reference OS/Scheduling and Dispatch) • Data management o Non-uniform communication costs due to proximity (cross-reference SF/Proximity) o Cache effects (e.g., false sharing) o Maintaining spatial locality • Impact of composing multiple concurrent components • Power usage and management
	Parallel Architecture [1 hour] • Multicore processors • Shared vs. distributed memory	**Parallel Architecture** [1 hour] • Symmetric multiprocessing (SMP) • SIMD, vector processing	**Parallel Architecture** • GPU, co-processing • Flynn's taxonomy • Instruction level support for parallel programming o Atomic instructions such as Compare and Set • Memory issues o Multiprocessor caches and cache coherence o Non-uniform memory access (NUMA) • Topologies o Interconnects o Clusters • Resource sharing (e.g., buses and interconnects)
	Communication and Coordination [1 hour] • Shared Memory o consistency, and its role in programming language guarantees for data-race-free programs	**Communication and Coordination** [3 hours] • Consistency in shared memory models • Message passing o Point-to-point versus multicast (or event-based) messages o Blocking versus non-blocking styles for sending and receiving messages o Message buffering (cross-reference PF/Fundamental Data Structures/Queues) • Atomicity o Specifying and testing atomicity and safety requirements o Granularity of atomic accesses and updates, and the use of constructs such as critical sections or transactions to describe them o Mutual Exclusion using locks, semaphores, monitors, or related constructs	**Communication and Coordination** • Consensus o (Cyclic) barriers, counters, or related constructs • Conditional actions o Conditional waiting (e.g., using condition variables)

		▪ Potential for liveness failures and deadlock (causes, conditions, prevention) ○ Composition ▪ Composing larger granularity atomic actions using synchronization ▪ Transactions, including optimistic and conservative approaches	
			Distributed Systems • Faults (cross-reference OS/Fault Tolerance) ○ Network-based (including partitions) and node-based failures ○ Impact on system wide guarantees (e.g., availability) • Distributed message sending ○ Data conversion and transmission ○ Sockets ○ Message sequencing ○ Buffering, retrying, and dropping messages • Distributed system design tradeoffs ○ Latency versus throughput ○ Consistency, availability, partition tolerance • Distributed service design ○ Stateful versus stateless protocols and services ○ Session (connection-based) designs ○ Reactive (IO-triggered) and multithreaded designs • Core distributed algorithms ○ Election, discovery • Scaling ○ Clusters, grids, meshes, and clouds
			Formal Models and Semantics • Formal models of processes and message passing, including algebras such as Communicating Sequential Processes (CSP) and pi-calculus • Formal models of parallel computation, including the Parallel Random Access Machine (PRAM) and alternatives such as Bulk Synchronous Parallel (BSP) • Models of (relaxed) shared memory consistency and their relation to programming language specifications • Algorithmic correctness criteria including linearizability • Models of algorithmic progress, including non-blocking guarantees and fairness • Techniques for specifying and checking correctness properties such as atomicity and freedom from data races
PL (8 Core-Tier1 hours, 20 Core-Tier2 hours)	**Fundamental Constructs** [9 hours] • Basic syntax and semantics of a higher-level language • Variables, types, expressions, and assignment • Simple I/O • Conditional and iterative control structures • Functions and parameter passing • Structured decomposition **Algorithmic Problem Solving** [2 hours] Problem-solving strategies	**Algorithmic Problem Solving** [4 hours] • The role of algorithms in the problem-solving process • Implementation strategies for algorithms • Debugging strategies • The concept and properties of algorithms	•
	Object-Oriented Programming [4 hours] • Object-oriented design ○ Decomposition into objects carrying state and having behavior ○ Class-hierarchy design for modeling • Definition of classes: fields, methods, and constructors • Subclasses, inheritance, and overriding • Dynamic dispatch: definition of method-call	**Object-Oriented Programming** [6 hours] • Subtyping (cross-reference PL/Type Systems) ○ Subtype polymorphism; implicit upcasts in typed languages ○ Notion of behavioral replacement ○ Relationship between subtyping and inheritance • Object-oriented idioms for encapsulation ○ Private fields ○ Interfaces revealing only method signatures ○ Abstract base classes Using collection classes, iterators, and other common library components	**Advanced Programming Constructs** • Lazy evaluation and infinite streams • Control Abstractions: Exception Handling, Continuations, Monads • Object-oriented abstractions: Multiple inheritance, Mixins, Traits, Multimethods • Metaprogramming: Macros, Generative programming, Model-based development • Module systems • String manipulation via pattern-matching • Dynamic code evaluation ("eval") Language support for checking assertions, invariants, and pre/post-conditions
		Event-Driven and Reactive Programming [2 hours] • Events and event handlers • Canonical uses such as GUIs, mobile devices, robots, servers • Using a reactive framework ○ Defining event handlers/listeners ○ Main event loop not under event-handler-writer's control • Externally-generated events and program-generated events • Separation of model, view, and controller	**Concurrency and Parallelism** • Constructs for thread-shared variables and shared-memory synchronization • Actor models • Futures • Language support for data parallelism • Models for passing messages between sequential processes • Effect of memory-consistency models on language semantics and correct code generation
	Functional Programming	**Functional Programming**	**Logic Programming**

	[3 hours] • Benefits of effect-free programming o Data can be freely aliased or copied without introducing unintended effects from mutation o Function calls have no side effects, facilitating compositional reasoning o Variables are immutable, preventing unexpected changes to program data by other code • Processing structured data (e.g., trees) via functions with cases for each data variant o Associated language constructs such as discriminated unions and pattern-matching over them o Compositional functions over structured data • First-class functions (taking, returning, and storing functions)	[4 hours] • Function closures (functions using variables in the enclosing lexical environment) o Basic meaning and definition -- creating closures at run-time by capturing the environment o Canonical idioms: call-backs, arguments to iterators, reusable code via function arguments o Using a closure to encapsulate data in its environment • Defining higher-order operations on aggregates, especially map, reduce/fold, and filter	• Clausal representation of data structures and algorithms • Unification • Backtracking and search
	Basic Type Systems [1 hour] • A type as a set of values together with a set of operations o Primitive types (e.g., numbers, Booleans) o Reference types o Compound types built from other types (e.g., records, unions, arrays, lists, functions) • Association of types to variables, arguments, results, and fields • Type safety and errors caused by using values inconsistently with their intended types • Goals and limitations of static typing o Eliminating some classes of errors without running the program o Inherent conservative approximation of static analysis due to undecidability	**Basic Type Systems** [4 hours] • Generic types (parametric polymorphism) o Definition o Use for generic libraries such as collections o Comparison with ad hoc polymorphism (overloading) and subtype polymorphism • Complementary benefits of static and dynamic typing o Errors early vs. errors late/avoided o Enforce invariants during code maintenance vs. postpone typing decisions while prototyping o Avoid misuse of code vs. allow more code reuse o Detect incomplete programs vs. allow incomplete programs to run	**Type Systems** • Compositional type constructors, such as product types (for aggregates), sum types (for unions), function types, quantified types, and recursive types • Type checking • Type safety as preservation plus progress • Type inference • Static overloading
		Program Representation [1 hour] • Programs that take (other) programs as input such as interpreters, compilers, type-checkers, documentation generators, etc. • Abstract syntax trees; contrast with concrete syntax • Data structures to represent code for execution, translation, or transmission	**Compiler Semantic Analysis** • High-level program representations such as abstract syntax trees • Scope and binding resolution • Type checking • Declarative specifications such as attribute grammars
		Language Translation and Execution [3 hours] • Interpretation vs. compilation to native code vs. compilation to portable intermediate representation • Language translation pipeline: parsing, optional type-checking, translation, linking, execution o Execution as native code or within a virtual machine o Alternatives like dynamic loading and dynamic code generation • Run-time representation of core language constructs such as objects (method tables) and first-class functions (closures) • Run-time layout of memory: call-stack, heap, static data o Implementing loops, recursion, and tail calls • Automated vs. manual memory management; garbage collection as an automatic technique using the notion of reachability	**Code Generation** • Instruction selection • Procedure calls and method dispatching • Register allocation • Separate compilation; linking • Instruction scheduling • Peephole optimization
			Syntax Analysis • Scanning (lexical analysis) using regular expressions • Parsing strategies including top-down (e.g., recursive descent, Earley parsing, or LL) and bottom-up (e.g., backtracking or LR) techniques; role of context-free grammars • Generating scanners and parsers from declarative specifications
			Runtime Systems • Target-platform characteristics such as registers, instructions, bytecodes • Dynamic memory management approaches and techniques: malloc/free, garbage collection (mark-sweep, copying, reference counting), regions (also known as arenas or zones) • Data layout for objects and activation records • Just-in-time compilation and dynamic recompilation • Other features such as class loading, threads, security, etc.
			Static Analysis • Relevant program representations, such as basic blocks, control-flow graphs, def-use chains, static single assignment, etc. • Flow-insensitive analyses, such as type-checking and scalable pointer and alias analyses • Flow-sensitive analyses, such as forward and backward dataflow analyses • Path-sensitive analyses, such as software model checking • Tools and frameworks for defining analyses • Role of static analysis in program optimization • Role of static analysis in (partial) verification and bug-finding
			Formal Semantics • Syntax vs. semantics

			• Lambda Calculus • Approaches to semantics: Operational, Denotational, Axiomatic • Proofs by induction over language semantics • Formal definitions and proofs for type systems • Parametricity
			Language Pragmatics • Principles of language design such as orthogonality • Evaluation order, precedence, and associativity • Eager vs. delayed evaluation • Defining control and iteration constructs • External calls and system libraries

SDF (42 Core-Tier1 hours)	**Algorithms and Design**
	[11 hours]
	• The concept and properties of algorithms
	  o Informal comparison of algorithm efficiency (e.g., operation counts)
	• The role of algorithms in the problem-solving process
	• Problem-solving strategies
	  o Iterative and recursive mathematical functions
	  o Iterative and recursive traversal of data structure
	  o Divide-and-conquer strategies
	• Implementation of algorithms
	• Fundamental design concepts and principles
	  o Abstraction
	  o Program decomposition
	  o Encapsulation and information hiding
	  o Separation of behavior and implementation		

	Fundamental Programming Concepts [10 hours] • Basic syntax and semantics of a higher-level language • Variables and primitive data types (e.g., numbers, characters, Booleans) • Expressions and assignments • Simple I/O • Conditional and iterative control structures • Functions and parameter passing • The concept of recursion		
	Fundamental Data Structures [12 hours] • Arrays • Records/structs (heterogeneous aggregates) • Strings and string processing • Stacks, queues, priority queues, sets & maps • References and aliasing • Simple linked structures • Strategies for choosing the appropriate data structure		

	Development Methods
	[9 hours]
	• Program correctness
	  o The concept of a specification
	  o Defensive programming (e.g. secure coding, exception handling)
	  o Code reviews
	  o Testing fundamentals and test-case generation
	  o Test-driven development
	  o The role and the use of contracts, including pre- and post-conditions
	  o Unit testing
	• Modern programming environments
	  o Programming using library components and their APIs
	• Debugging strategies
	• Documentation and program style		

SE (6 Core-Tier1 hours;	**Software Processes** [1 hours] • Systems level considerations, i.e., the interaction of software with its intended environment • Phases of software life-cycles • Programming in the large vs. individual programming	**Software Processes** [2 hours] • Software process models (e.g., waterfall, incremental, agile)	**Software Processes** • Software quality concepts • Process improvement • Software process capability maturity models • Software process measurements

21
Core-
Tier2
hours)
		Software Project Management
		[3 hours]
		• Risk
		  ○ The role of risk in the life cycle
		  ○ Risk categories including security, safety, market, financial, technology, people, quality, structure and process
		  ○ Risk identification
		  ○ Risk tolerance (e.g., risk-adverse, risk-neutral, risk-seeking)
		  ○ Risk planning
		  ○ Risk removal, reduction and control
		• Team participation
		  ○ Team processes including responsibilities for tasks, meeting structure, and work schedule
		  ○ Roles and responsibilities in a software team
		  ○ Team conflict resolution
		  ○ Risks associated with virtual teams (communication, perception, structure)
		• Effort Estimation (at the personal level)
			Software Project Management
			• Team management
			  ○ Team organization and decision-making
			  ○ Role identification and assignment
			  ○ Individual and team performance assessment
			• Project management
			  ○ Scheduling and tracking
			  ○ Project management tools
			  ○ Cost/benefit analysis
			• Software measurement and estimation techniques
			• Software quality assurance and the role of measurements
			• Principles of risk management
			• Risk analysis and evaluation
			• System-wide approach to risk including hazards associated with tools

		Tools and Environments [2 hours] • Software configuration management and version control; release management • Requirements analysis and design modeling tools • Testing tools including static and dynamic analysis tools • Programming environments that automate parts of program construction processes (e.g., automated builds) • Tool integration concepts and mechanisms	
	Requirements Engineering [1 hour] • Fundamentals of software requirements elicitation and modeling	**Requirements Engineering** [3 hours] • Properties of requirements including consistency, validity, completeness, and feasibility • Software requirements elicitation • Describing functional requirements using, for example, use cases or users stories • Non-functional requirements and their relationship to software quality • Describing system data using, for example, class diagrams or entity-relationship diagrams • Evaluation and use of requirements specifications	**Requirements Engineering** • Requirements analysis modeling techniques • Acceptability of certainty / uncertainty considerations regarding software / system behavior • Prototyping • Basic concepts of formal requirements specification • Requirements specification • Requirements validation • Requirements tracing
	Software Design [4 hours] • Overview of design paradigms • System design principles: divide and conquer (architectural design and detailed design), separation of concerns, information hiding, coupling and cohesion, re-use of standard structures. • Appropriate models of software designs, including structure and behavior. • Software architecture concepts	**Software Design** [4 hours] • Design Paradigms such as structured design (top-down functional decomposition), object-oriented analysis and design, event driven design, component-level design, data-structured centered, aspect oriented, function oriented, service oriented. • Relationships between requirements and designs: transformation of models, design of contracts. • Architectural design: standard architectures (e.g. client-server, n-layer, transform centered, pipes-and- filters, etc). • Refactoring designs and the use of design patterns. • The use of components in design: component selection, design, adaptation and assembly of components, components and patterns, components and objects, (for example, build a GUI using a standard widget set).	**Software Design** • Internal design qualities, and models for them: efficiency and performance, redundancy and fault tolerance, traceability of requirements. • External design qualities, and models for them: functionality, reliability, performance and efficiency, usability, maintainability, portability. • Measurement and analysis of design quality. • Tradeoffs between different aspects of quality. • Application frameworks. • Middleware: the object-oriented paradigm within middleware, object request brokers and marshalling, transaction processing monitors, workflow systems.

		Software Construction
		[2 hours]
		• Coding practices: techniques, idioms/patterns, mechanisms for building quality programs
		  ○ Defensive coding practices
		  ○ Secure coding practices
		  ○ Using exception handling mechanisms to make programs more robust, fault-tolerant
		• Coding standards
		• Integration strategies
			Software Construction
			• Robust And Security Enhanced Programming
			  ○ Defensive programming
			  ○ Principles of secure design and coding:
			  ○ Principle of least privilege
			  ○ Principle of fail-safe defaults
			  ○ Principle of psychological acceptability
			• Potential security problems in programs
			  ○ Buffer and other types of overflows
			  ○ Race conditions
			  ○ Improper initialization, including choice of privileges
			  ○ Checking input
			  ○ Assuming success and correctness
			  ○ Validating assumptions
			• Documenting security considerations in using a program

		Software Verification Validation [3 hours] • Verification and validation concepts • Inspections, reviews, audits • Testing types, including human computer interface, usability, reliability, security, conformance to specification	**Software Verification Validation** • Static approaches and dynamic approaches to verification • Regression testing • Test-driven development • Validation planning; documentation for validation • Object-oriented testing; systems testing

		• Testing fundamentals o Unit, integration, validation, and system testing o Test plan creation and test case generation o Black-box and white-box testing techniques • Defect tracking • Testing parallel and distributed systems	• Verification and validation of non-code artifacts (documentation, help files, training materials) • Fault logging, fault tracking and technical support for such activities • Fault estimation and testing termination including defect seeding
		Software Evolution [1 hour] • Software development in the context of large, pre-existing code bases • Software evolution • Characteristics of maintainable software • Reengineering systems • Software reuse	
			Formal Methods • Role of formal specification and analysis techniques in the software development cycle • Program assertion languages and analysis approaches (including languages for writing and analyzing pre-and post-conditions, such as OCL, JML) • Formal approaches to software modeling and analysis o Model checkers o Model finders • Tools in support of formal methods
		Software Reliability [1 hour] • Software reliability engineering concepts • Software reliability, system reliability and failure behavior (cross-reference SF9/Reliability Through Redundancy) • Fault lifecycle concepts and techniques	**Software Reliability** • Software reliability models • Software fault tolerance techniques and models • Software reliability engineering practices • Measurement-based analysis of software reliability
SF (18 core Tier 1, 9 core Tier 2 hours, 27 total)	**Computational Paradigms** [3 hours] • A computing system as a layered collection of representations • Basic building blocks and components of a computer (gates, flip-flops, registers, interconnections; Datapath + Control + Memory) • Hardware as a computational paradigm: Fundamental logic building blocks (logic gates, flip-flops, counters, registers, PL); Logic expressions, minimization, sum of product forms • Application-level sequential processing: single thread [xref PF/] • Simple application-level parallel processing: request level (web services/client-server/distributed), single thread per server, multiple threads with multiple servers • Basic concept of pipelining, overlapped processing stages • Basic concept of scaling: going faster vs. handling larger problems	**Resource Allocation and Scheduling** [2 hours] • Kinds of resources: processor share, memory, disk, net bandwidth • Kinds of scheduling: first-come, priority • Advantages of fair scheduling, preemptive scheduling	
	Cross-Layer Communications [3 hours] • Programming abstractions, interfaces, use of libraries • Distinction between application and OS services, remote procedure call • Interactions between applications and virtual machines • Reliability	**Proximity** [3 hours] [Cross-reference: AR/Memory Management, OS/VM/Virtual Memory] • Speed of light and computers (one foot per nanosecond vs. one GHz clocks) • Latencies in computer systems: memory vs. disk latencies vs. across the network memory • Caches, spatial and temporal locality, in processors and systems • Elementary introduction into the processor memory hierarchy: registers and multi-level caches, and the formula for average memory access time	
	State-State Transition-State Machines [6 hours] • Digital vs. analog/discrete vs. continuous systems • Simple logic gates, logical expressions, Boolean logic simplification • Clocks, state, sequencing • Combinational Logic, Sequential Logic, Registers, Memories • Computers and Network Protocols as examples of State Machines	**Virtualization and Isolation** [2 hours] • Rationale for protection and predictable performance • Levels of indirection, illustrated by virtual memory for managing physical memory resources • Methods for implementing virtual memory and virtual machines	
	System Support for Parallelism [3 hours] • Execution and runtime models that distinguish Sequential vs. Parallel processing • System organizations that support Request and Task parallelism and other parallel processing paradigms, such as Client-Server/Web Services, Thread parallelism(Fork-Join), and Pipelining • Multicore architectures and hardware support for parallelism	**Reliability through Redundancy** [2 hours] • Distinction between bugs and faults, and how they arise in hardware vs. software • How errors increase the longer the distance between the communicating entities; the end-to-end principle as it applies to systems and networks Redundancy through check and retry • Redundancy through redundant encoding (error correcting codes, CRC/Cyclic • Redundancy Codes, FEC/Forward Error Correction) • Duplication/mirroring/replicas	
	Performance [3 hours] • Figures of performance merit (e.g., speed of execution, energy consumption, bandwidth vs. latency, resource cost) • Benchmarks (e.g., SPEC) and measurement methods		

	• CPI equation (Execution time = # of instructions * cycles/instruction * time/cycle) as tool for understanding tradeoffs in the design of instruction sets, processor pipelines, and memory system organizations. • Amdahl's Law: the part of the computation that cannot be sped up limits the effect of the parts that can		
SP (11 Core-Tier1 hours, 5 Core-Tier2 hours)	**Social Context** [1 hour] • Social implications of computing in a networked world • Impact of social media on individualism, collectivism and culture.	**Social Context** [2 hours] • Growth and control of the Internet • The digital divide (including gender, class, ethnicity, underdeveloped countries) • Accessibility issues, including legal requirements • Context-aware computing	
	Analytical Tools [2 hours] • Ethical argumentation • Ethical theories and decision-making • Moral assumptions and values		
	Professional Ethics [2 hours] • Community values and the laws by which we live • The nature of professionalism including care, attention and discipline, fiduciary responsibility, and mentoring • Keeping up-to-date as a professional in terms of knowledge, tools, skills, legal and professional framework as well as the ability to self-assess and computer fluency • Codes of ethics, conduct, and practice such as the ACM/IEEE, SE, AITP, IFIP and international societies • Accountability, responsibility and liability	**Professional Ethics** [2 hours] • The role of the professional in public policy • Maintaining awareness of consequences • Ethical dissent and whistle-blowing • Dealing with harassment and discrimination • Forms of professional credentialing • Acceptable use policies for computing in the workplace • Ergonomics and healthy computing environments • Time to market versus quality professional standards	
	Intellectual Property 2 hours] • Philosophical foundations of intellectual property • Intellectual property rights • Intangible digital intellectual property (IDIP) • Legal foundations for intellectual property protection • Digital rights management • Copyrights, patents, trademarks • Plagiarism		**Intellectual Property** • Foundations of the open source movement • Software piracy
	Privacy and Civil Liberties [2 Core-Tier1 hours] • Philosophical foundations of privacy rights • Legal foundations of privacy protection • Privacy implications of widespread data collection for transactional databases, data warehouses, surveillance systems, and cloud computing • Ramifications of differential privacy • Technology-based solutions for privacy protection		**Privacy and Civil Liberties** • Privacy legislation in areas of practice • Civil liberties • Freedom of expression and its limitations
	Professional Communication [1 hour] • Reading, understanding and summarizing technical material, including source code and documentation • Writing effective technical documentation and materials • Dynamics of oral, written, and electronic team and group communication • Communicating professionally with stakeholders • Utilizing collaboration tools		**Professional Communication** • Dealing with cross-cultural environments • Tradeoffs of competing risks in software projects, such as technology, structure/process, quality, people, market and financial
	Sustainability [1 hour] • Being a sustainable practitioner, e.g., consideration of impacts of issues, such as power consumption and resource consumption • Explore global social and environmental impacts of computer use and disposal (e-waste)	**Sustainability** [1 hour] • Environmental impacts of design choices in specific areas such as algorithms, operating systems, networks, databases, programming languages, or human-computer interaction (cross-reference: HCI/Embedded and Intelligent Systems/Energy-aware interfaces)	**Sustainability** • Guidelines for sustainable design standards • Systemic effects of complex computer-mediated phenomena (e.g. telecommuting or web shopping) • Pervasive computing. Information processing that has been integrated into everyday objects and activities, such as smart energy systems, social networking and feedback systems to promote sustainable behavior, transportation, environmental monitoring, citizen science and activism. • Conduct research on applications of computing to environmental issues, such as energy, pollution, resource usage, recycling and reuse, food management, farming and others.
			History • Prehistory—the world before 1946 • History of computer hardware, software, networking • Pioneers of computing • History of Internet

			Economies of Computing • Monopolies and their economic implications • Effect of skilled labor supply and demand on the quality of computing products • Pricing strategies in the computing domain • The phenomenon of outsourcing and off-shoring; impacts on employment and on economics • Differences in access to computing resources and the possible effects thereof • Costing out jobs with considerations on manufacturing, hardware, software, and engineering implications • Cost estimates versus actual costs in relation to total costs • Entrepreneurship: prospects and pitfalls • Use of engineering economics in dealing with finances
			Security Policies, Laws and Computer Crimes • Examples of computer crimes and legal redress for computer criminals • Social engineering and identity theft (cross-reference: HCI/Human Factors and Security/social engineering) • Issues surrounding the misuse of access and breaches in security • Motivations and ramifications of cyber terrorism and criminal hacking, “cracking” • Effects of malware, such as viruses, worms and Trojan horses • Crime prevention strategies • Security policies