PROJECT:
A MAP OF COMPUTER SCIENCE
~ Research Guidelines ~

Deadline for research:

Because the implementation for the project (interactive website, interactive LaTeX document, *we haven't decided fully yet*) will take more time than the research, the **research part of the project (by every member of the team and their chosen topic) needs to be finished ideally before the Easter holidays (27-28 April) (maximum just before week 10 (6-10 May)**

General method to research:

- Our two main sources are going to be Denning's article and the ACM/IEEE-CS/AAAI Computer Science Curricula.
- To avoid using too many sources and potentially writing irrelevant information, **try to only use these two main sources (mainly Denning's article) to answer all of the questions** written here.
- Only use other sources (other articles, other classifications and curricula) **if the two main sources do not provide enough information** to answer the questions. If you use other sources, try your best to keep track of what sources you use, so we can list our references in the end.
- We have to understand what we write, because we will have to present it. Don't focus on adding as much content as possible, but focus on answering the questions in an understandable, cohesive way (*in other words, not too much bla bla*).

What to actually write about:

In Craciun's words:

The following need to be represented on the map:

- ***the areas of computer science (according to the article of Denning - Computer Science: the Discipline, or alternatively, according to similar classifications - IEEE, ACM, AAAI),***
- ***for each area, represent the main activities (theory, experiment, design),***
- ***for each area, the relations/connections to other areas (dependency, influence, etc),***
- ***for each area, important problems, open problems (with examples that give an intuition on the nature of the problem),***
- ***for each area, the important people,***
- ***for each area, the important venues (journals, conferences),***
- ***the map should address both the local dimension (the Department of Computer Science @ UVT) and the global dimension.***

In detail:

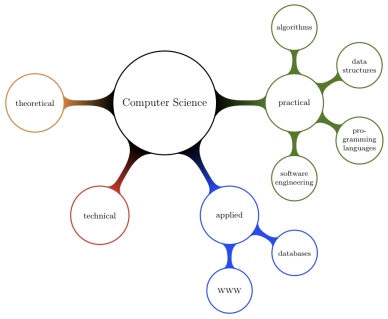
- **Activities in the field/area**
  - **Theory (scientific research topics of the field)**
  - **Experiments (what are scientists of this field experimenting on)**
    - **We can talk about the important, open problems in the field in this section (***for example, experiments in the Algorithms and Data Structures field of CS include researching, designing new algorithms, new data structures, to solve problems of higher sizes***). Because, with research and experimentation, scientists try to find solutions to these problems.**
      - But, not all problems in a field can be solved (*e.g. P=NP problem*)
  - **Design (how new, discovered and experimented things are being implemented in the real world) (***e.g. using Machine Learning, Data Science, and other advancing subfields of Artificial Intelligence, chatbots such as Gemini, ChatGPT are made widely available to the general public***)**
- **People, Places**
  - **Important innovators in the field today and their impact in the field, what are they researching, what are they trying to improve, etc.**

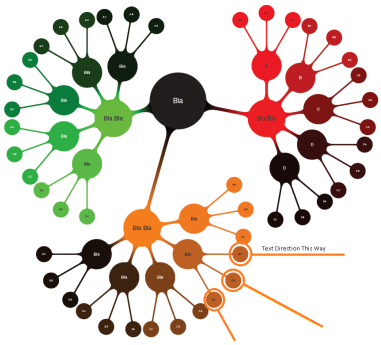
- **Conferences, journals (news sites, magazines, etc.) related to the field, basically how people in the field meet up and stay up to date with what's going on in the field.**

- **"It's all connected!"**
  - **All of these fields are connected, obviously. But some fields are more related to one another than others. Make sure to mention this.**

Presentation ideas:

- Connections between the various fields can be represented visually, using a diagram (these are all made in LaTeX).





~ RESEARCH START ~

Leave this page empty.

Algorithms and Data Structures

STEFAN

Introduction:

Algorithms and data structures are fundamental concepts in computer science, essential for solving computational problems efficiently. An algorithm is a step-by-step procedure or formula for solving a problem, while data structures organize and store data to enable efficient access and modification. Together, they form the backbone of software development, allowing for optimized performance and resource management.

Theory

The theory of algorithms encompasses computability theory, computational complexity theory, concurrency theory, probabilistic algorithm theory, database theory, randomized algorithms, pattern-matching algorithms, graph and network algorithms, algebraic algorithms, combinatorial optimization, and cryptography. It is supported by discrete mathematics (graph theory, recursive functions, recurrence relations, combinatorics), calculus, induction, predicate logic, temporal logic (a calculus of time dependent events), semantics, probability, and statistics.

Moreover, this understanding provides insight into the intrinsic nature of computation, computational problems, and computational problem-solving as well as possible solution techniques independent of programming language, programming paradigm, computer hardware, or other implementation aspects.

Experimentation

Experimentation is a vital part of research, especially when attempting to find the characteristics of an algorithm that is too complex to achieve by known theoretical analysis.
To evaluate the performance, an algorithm can be applied to various test cases.
In the real world testing has gathered information about certain methods such as divide-and-conquer, greedy algorithms, dynamic programming, finite state machine interpreters, stack machine interpreters, heuristic searches, and randomized algorithms.

- Benchmarking:
    - Performance Testing: Implementing algorithms and running them on different datasets to measure their actual runtime and memory usage.
    - Comparative Analysis: Comparing the performance of different algorithms for the same problem to determine which is more efficient in practice.
- Simulation:
    - Scenario Testing: Simulating various real-world scenarios to observe how algorithms perform under different conditions.
    - Stress Testing: Running algorithms under extreme conditions (e.g., large datasets, limited memory) to evaluate their robustness and scalability.
- Profiling:
    - Runtime Profiling: Using tools to analyze which parts of the algorithm consume the most resources during execution.
    - Memory Profiling: Measuring memory usage to identify and optimize inefficient memory usage.

Design

Many useful, practical algorithms have been designed and placed in program libraries -- for example, mathematical software, searching, sorting, random-number generation, textual pattern matching, hashing, graphs, trees, communication network protocols, distributed-data updates, semaphores, deadlock detectors, synchronizers, storage managers, lists, tables, and paging algorithms. Many theoretical results have been translated into useful and practical systems, such as the RSA public key cryptosystem, production-quality compilers, and VLSI circuit layout.

There are many steps involved in writing a computer program to solve a given problem. The steps go from problem formulation and specification, to design of the solution, to implementation, testing and documentation, and finally to evaluation of the solution.

- Algorithm Development:
    - Problem Identification: Defining the problem clearly and understanding the requirements and constraints.
    - Algorithm Design: Developing new algorithms or modifying existing ones to meet the problem's requirements.
- Data Structure Implementation:
    - Choosing the Right Data Structure: Selecting appropriate data structures (e.g., arrays, linked lists, trees, graphs) based on the problem requirements.
    - Custom Data Structures: Designing custom data structures tailored to specific applications.
- Optimization:
    - Code Optimization: Refining the implementation of algorithms to enhance performance (e.g., reducing time complexity, minimizing memory usage).
- Application Development:
    - Integration: Integrating algorithms and data structures into software applications.
    - User Testing: Conducting user testing to ensure that the algorithms meet real-world needs and performance expectations.

Relation Ship

Dependence

In many fields, Mathematics sits at the base for many algorithms. For example, number theory guides cryptographic algorithms, linear algebra supports machine learning algorithms, and combinatorics aids optimization algorithms. Techniques from Operations Research, like linear programming, heavily influence the development of

optimization algorithms. Algorithms used in simulations, such as those modeling particle systems and fluid dynamics, are influenced by principles from Physics. Additionally, insights from Biology, especially evolutionary principles, inspire algorithms used in optimization problems and machine learning.

Influences

Algorithms and Data Structures have implication
In Theoretical Computer Science, Complexity Theory relies on algorithms to understand computational complexity, while Automata Theory employs data structures like trees and graphs. In Software Engineering, efficient algorithms and data structures are crucial for robust systems and code optimization. Artificial Intelligence and Machine Learning utilize algorithms for search and data handling tasks. Databases depend on algorithms for indexing and query optimization. Networks use algorithms for routing and load balancing. Cryptography relies on efficient algorithms for security. Bioinformatics utilizes algorithms for sequence alignment and phylogenetic tree modeling. Computational Geometry employs algorithms and data structures for spatial problem-solving. Computer Graphics and Visualization rely on efficient algorithms and specialized data structures for rendering and spatial partitioning.

In UVT

Problems

Algorithms and Data Structures addresses day to day problems such as sorting and searching elements in a given data structure. Other problems are presented in Graphs, like finding the shortest paths (Dijkstra's Algorithm) or detecting cycles. Languages also create the problem of pattern matching (Knuth-Morris-Pratt Algorithm (KMP)), substring search and string manipulation

Alongside solved problems, there remain the ones that have been too difficulty or we simply lack the technology to solve them:

1. P vs NP Problem:
    - Problem: Determine whether every problem whose solution can be verified quickly (in polynomial time) can also be solved quickly (in polynomial time).
    - Example: The Traveling Salesman Problem, Subset Sum Problem.
2. Optimization Problems:
    - Problem: Find the best solution from all feasible solutions.
    - Example: Maximum Flow Problem, Minimum Spanning Tree Problem.
3. Data Structure Design:
    - Problem: Design efficient data structures for specific applications or improve existing ones.
    - Example: Designing data structures for spatial data, dynamic sets, or persistent data.
4. Parallel Algorithms:
    - Problem: Develop efficient algorithms for parallel and distributed computing environments.
    - Example: Parallel Matrix Multiplication, Parallel Sorting Algorithms.
5. Approximation Algorithms:
    - Problem: Find near-optimal solutions to optimization problems in polynomial time.
    - Example: Approximate Traveling Salesman Problem, Approximate Vertex Cover.

People

1. Donald Knuth:
    - Often regarded as the "father of algorithms," Knuth is the author of *The Art of Computer Programming*, which is an informational treasure on algorithms and data structures.
    - Key Works: Developed the Knuth-Morris-Pratt (KMP) string matching algorithm and contributed to the analysis of algorithms and computational complexity.
2. Edsger W. Dijkstra:

- Made foundational contributions to algorithm design and programming methodology. Known for Dijkstra's algorithm for shortest paths in graphs and the concept of structured programming.
- Key Works: Dijkstra's algorithm, the concept of semaphores in concurrent computing.

3. Robert Tarjan:
    - Known for his work on graph theory and data structures, including efficient algorithms for finding strongly connected components and union-find data structures.
    - Key Works: Tarjan's algorithm for finding strongly connected components, Fibonacci heaps, and splay trees.
4. Robert Sedgewick:
    - Contributions: Robert Sedgewick is renowned for his contributions to algorithmic education and research. He co-authored textbooks like "Algorithms," widely used in computer science education.
    - Key Works: Sedgewick's research focuses on practical applications, particularly in graph algorithms and algorithm analysis.
5. Michael Mitzenmacher:
    - Contributions: Michael Mitzenmacher specializes in probabilistic methods and data structures, contributing significantly to algorithms for large-scale data processing.
    - Key Works: Mitzenmacher's research includes hash functions and Bloom filters, essential in developing efficient data storage systems.
6. Eva Tardos:
    - Contributions: Eva Tardos is known for her work in algorithmic game theory and network algorithms, particularly in network optimization problems.
    - Key Works: Tardos' research has advanced algorithms for flow and matching problems, contributing to the understanding of algorithmic game theory.

Important Journals

1. Journal of the ACM (JACM):

- Description: The flagship journal of the Association for Computing Machinery (ACM), covering all aspects of computer science, including algorithms and data structures.
- Impact: Known for publishing high-quality, foundational research papers.

2. SIAM Journal on Computing (SICOMP):
    - Description: Published by the Society for Industrial and Applied Mathematics (SIAM), it covers theoretical computer science, including algorithms, computational complexity, and data structures.
    - Impact: Highly respected for rigorous mathematical and theoretical research.
3. Algorithmica:
    - Description: Focuses on algorithm design, analysis, and implementation, as well as related aspects of computer science.
    - Impact: Known for detailed studies of algorithms and their applications.
4. ACM Transactions on Algorithms (TALG):
    - Description: Publishes original research of the highest quality dealing with all aspects of algorithms.
    - Impact: A key venue for significant algorithmic research.

Important Conferences

1. Annual ACM Symposium on Theory of Computing (STOC):
    - Description: One of the most prestigious conferences in theoretical computer science, covering a wide range of topics including algorithms and data structures.
    - Impact: Known for presenting groundbreaking research in the field.
2. IEEE Symposium on Foundations of Computer Science (FOCS):
    - Description: Along with STOC, it is one of the top conferences in theoretical computer science.
    - Impact: Highly regarded for its rigorous selection process and high-impact papers.
3. Annual ACM-SIAM Symposium on Discrete Algorithms (SODA):
    - Description: Focuses specifically on research in discrete algorithms and data structures.
    - Impact: One of the primary venues for high-quality research in algorithms.
4. European Symposium on Algorithms (ESA):
    - Description: Covers research in the design and analysis of algorithms, experimental evaluation, and real-world applications.

- Impact: A leading European conference in the field.

5. Symposium on Experimental Algorithms (SEA):
    - Description: Focuses on the experimental evaluation of algorithms and data structures.
    - Impact: Important for research that emphasizes practical performance and empirical results.

RESOURCES:

1. Aho, A. V., Laboratories, B., & Hill, M. (n.d.). Data Structures and Algorithms.
2. Body-of-Knowledge-v1-bookmarksv2.pdf. (n.d.).
3. Denning, P. J. (n.d.). COMPUTER SCIENCE: THE DISCIPLINE.
4. https://en.wikipedia.org/wiki/List_of_computer_science_conferences

Programming Languages

BIA

1 Theory

The field of programming languages theory examines machines' models that produce and interpret languages as well as grammars for representing legitimate strings within those languages. Instances consist of formal language models and automata, Turing machines, lambda-calculus and propositional logic. The theory focuses on semantics, examining the connections between language strings and virtual machine states.
A programming language includes syntax and semantics, and more broadly, it also includes a standard library and an ecosystem. The standard library offers a range of tools needed for applications, including data structures such as lists and maps, functions for managing file and network input/output, and more. Programmers also consider the programming language's ecosystem to be crucial. The language ecosystem consists of developers, companies, third-party libraries, and other related components.

2 Experiments

To choose wisely between languages that allow the use of multiple complementary techniques, programmers must be aware of different programming models, programming functions and structures, and the programming concepts that underlie them. To increase the efficiency of software execution and long-term maintenance, it would be useful to understand how the functions of programming languages are defined, assembled and used. Additionally helpful fundamental understandings are language translation, program analysis, run-time behavior, memory management, and the interaction of several processes that communicate with one another using shared memory, synchronization, and message-passing.

3 Design

Programming languages have been classified in various ways to understand their syntactic and semantic models, applications, and functional structures. Here are some key classifications:

- **By Syntax and Semantics:**
  - Typing: Static vs. dynamic typing
  - Paradigms: Functional, procedural, object-oriented, logic specification, message-passing, and dataflow
- **By Application:**
  - Business data processing
  - Simulation
  - List processing
  - Graphics

• By Functional Structure:
- Procedure hierarchies
- Functional composition
- Abstract data types
- Communicating sequential processes

• By Abstract Implementation Models:
- Imperative
- Object-oriented
- Logic and constraint
- Concurrent
- Distributed

Language designers have created many practical programming languages tailored to these classifications. Examples include:
- Procedural Languages: COBOL, Fortran, Algol, Pascal, Ada, C
- Object-Oriented Languages: CLU, Smalltalk, C++, Eiffel, Java
- Functional Languages: Lisp, ML, Haskell
- Dataflow Languages: Sisal, Val, Id Nouveau
- Logic Languages: Prolog
- String Processing Languages: Snobol, Icon
- Concurrency Languages: Concurrent Pascal, Occam, SR, Modula-3

These languages have been supported by various run-time models, execution models, and tools, such as: static and dynamic execution models, type checking, storage and register allocation, compilers and cross-compilers, interpreters, analyzers for parallelism, programming environments with tools for syntactic and semantic error checking, profiling, debugging, and tracing.
A significant achievement in this field is the development of programs that can automatically produce compilers from language descriptions, such as YACC and LEX in Unix environments, leading to the creation of very efficient compilers.
Programming languages are widely used across different application domains to handle tasks such as creating tables, graphs, chemical formulas, spreadsheets, equations, input and output, and data queries. For each application, designers often create specialized mini-languages and parsers to meet specific needs.

4 Connections to other areas

4.1 Dependency

The foundations of programming languages are firmly based in discrete mathematics, logic, and formal languages. Essential mathematical concepts in this context include:
• Predicate Logic: For formal program reasoning.
• Temporal Logic: For reasoning about time-dependent behaviors.
• Modern Algebra: For structuring and manipulating programming constructs.
• Mathematical Induction: For proving properties of programs.

4.2 Influence

With the rise of ubiquitous computing and real-time temporal computing applications in everyday life—such as in health, transportation, and smart homes—it is crucial to address the software

development aspects of event-driven and reactive programming, as well as parallel and distributed computing.
These advanced topics often overlap with concepts from other key areas, including:
• Architecture and Organization: Understanding the hardware-software interface.
• Operating Systems: Managing resources and processes.
• Systems Fundamentals: Core principles of computer system operations

5 Important problems

Programming languages have advanced to accommodate various paradigms like imperative, object-oriented, functional, logic, and event-driven programming. This incorporation enables a well-rounded strategy, utilizing the advantages of each framework to effectively cater to a variety of programming requirements. Important inquiries to comprehend programming languages consist of:
• Arrangement of Virtual Machines: How are data types, operations, and control structures specified in languages? What methods are available for incorporating new categories and functions?
• Execution: In what way do these conceptual ideas manifest on real physical devices?
• Syntax in Notation: How can instructions be effectively and efficiently communicated to the computer through syntax?
• Semantics of Functions: How are interpretations given to language elements?
• Restatement: What is the process machines use to transform code from one language to another?
Understanding how distribution, concurrency, and parallelism integrate with other paradigms has become essential due to the widespread use of multicore computing, cloud computing, and computer networking. This blending is crucial for creating effective and expandable systems, highlighting the significance of these ideas as key subjects in contemporary computer science training. The text is about paraphrasing a given text while maintaining the original language and word count.

6 Important people

6.1 Guido van Rossum

Known as the creator of Python, van Rossum continues to influence the development of this widely-used language, although he has stepped down from his leadership role. His contributions to Python's design and community have made it one of the most popular and versatile programming languages.

6.2 Brendan Eich

The creator of JavaScript, Eich's work remains crucial as JavaScript continues to dominate web development. His influence extends through his ongoing involvement in web standards and browser technologies.

6.3 Anders Hejlsberg

A key figure in the development of C, Hejlsberg's work at Microsoft has shaped the language into a robust and versatile tool for a variety of applications. His previous work on Turbo Pascal and Delphi also highlights his significant contributions to programming.

7 Important venues

Journals:
• ACM Transactions on Programming Languages and Systems (TOPLAS)
• Journal of Functional Programming

• Theoretical Computer Science

Conferences:
• ACM SIGPLAN Symposium on Principles of Programming Languages
(POPL)
• International Conference on Functional Programming (ICFP)
• Conference on Programming Language Design and Implementation (PLDI)

8 References

Denning, P. J. (n.d.). COMPUTER SCIENCE: THE DISCIPLINE.
Hong, Jaemin, and Sukyoung Ryu. Introduction to Programming Languages. Febru-
ary 23, 2023.
Johnston, Wesley M., J. R. Paul Hanna, and Richard J. Millar. ”Advances in Dataflow Programming Languages.” ACM Computing Surveys 36, no. 1 (2004): 1-34
Batty, Mark, Kayvan Memarian, Kyndylan Nien-huis, Jean Pichon-Pharabod, and Peter Sewell. ”The Problem of Programming Language Concurrency Semantics.” In Programming Languages and Systems (ESOP 2015), 283-307. 2015.
https://cstheory.stackexchange.com/questions/17868/research-and-open-challenges-in-programming-language-theory
https://brendaneich.com/
https://github.com/ahejlsberg

Computer Architecture

ARTIOM

Link to the LaTeX document: https://github.com/ALeliuhin/CAO

Software Engineering

RAUL

Full finished PDF made using LaTeX available here. ~~I will slowly be transferring the PDF's contents into this Google Doc.~~

Operating Systems and Networks

ELENA

Operating Systems and Networks

This area deals with control mechanisms that allow multiple resources to be efficiently coordinated in computations distributed over many computer systems connected by local and wide-area networks. **Fundamental questions** include: How can interfaces be organized so that users deal only with abstract versions of resources and not with physical details of hardware? What are the principles by which systems can be extended in function by repeated application of a small number of construction rules? How should distributed computations be organized, with the details of network protocols, host locations, bandwidths, and resource naming being mostly invisible?

Major **elements of theory** in operating systems include: concurrency theory (synchronization, determinacy, and deadlocks); scheduling theory; program behavior and memory management theory; network flow theory; performance modeling and analysis. Supporting mathematics include bin packing, probability, queueing theory, queueing networks, communication and information theory, temporal logic, and cryptography.

What Is an Operating System?

In the 1960s, the definition of an operating system might have been the software that controls the hardware. But the landscape of computer systems has evolved significantly since then, requiring a richer definition. An operating system is software that enables applications to interact with a computer's hardware. The software that contains the core components of the operating system is called the kernel.

Operating systems can be found in devices ranging from cell phones and automobiles to personal and mainframe computers. In most computer systems, a user requests that the computer perform an action (e.g., execute an application or print a document) and the operating system manages the software and hardware to produce the desired result.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, peripherals, and other resources. For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it.

Experimentation

The Problems of Testing a New OS:

- Testing new OS poses a unique set of challenges that demand a comprehensive understanding of the system and innovative problem-solving approaches.
- Unpredictability: A new OS is an intricate web of features, functions, and interactions. Testing such an ecosystem is unpredictable due to the many variables at play. One minor change can cause a ripple effect leading to unexpected behaviors elsewhere in the system.
- Complexity: Testing a new OS isn't just about verifying individual features. It requires assessing how these features work together, interact with different hardware, and stand up to varying user scenarios. This complexity makes it challenging to create comprehensive test scenarios and identify potential weaknesses.
- Resource Constraints: Time and resources are often limited. Every added feature or patch needs testing and achieving full coverage within constraints is an uphill task.
- Versioning and Compatibility: The frequent release of updates and patches can introduce new bugs and compatibility issues, making it a constant chase to ensure the OS remains bug-free and functional.

There exists a general need in the development of software/systems to ensure that the finished product is sufficiently tested prior to its release to the public. Such testing is performed to detect programming errors/flaws. Corrections are formulated and then incorporated into subsequent versions of the software. There are various levels of testing thoroughness. The more thorough software is tested prior to release of the software to users, the less likely bugs will be exposed in the subsequent use of the released software.

An application program interface of an operating system comprises a set of methods defining entry points by applications into the functionality (e.g., retrieving/opening files, launching executable programs, printing, etc.) Supported/provided by the operating system. Furthermore, the functionality of the operating system comprises groups of related functions referred to herein as components. In the Microsoft Windows.(R) operating system, examples of components include: bundled items such as Media Player and Shell (which includes multiple subcomponents such as Active Desktop. Address Bar, Common Dialogs, Control Panel, Desktop Folder, Explorer, File Association, etc.). Application program interface (API) testing ensures that operating system components (e.g., printing, shell, etc.). when called upon by application programs, execute in an intended manner. During API testing, an application program calls a set of methods included in the operating system API. The calls are modified during the course of testing to include various combinations of passed parameter values. The behavior of the called method is observed for each of the application calls.

API testing to ensure that an operating system will operate as intended, when properly performed, is both extensive and exhaustive. Such testing is extremely time consuming with regard to programming the calls to the API methods and with regard to the Subsequent analysis of the operating system responses to the test calls. By way of example, API testing comprises

Submitting various combinations of values via an API to operating system components under test. In many, if not most, cases a large number of input parameters, and the many different potential values assignable to each of those input parameters discourages complete testing of the interaction between applications and the operating system program code via the API. As a result, previous methods for testing operating system components generally settled for less than complete coverage of the potential interactions between all application programs and the components of the operating system accessed through the operating system API.

One general technique for testing operating system components involves writing applications to directly call Jul. 6, 2006 and test operating system components via their specified API methods. In such cases, the application programs are written for the sole purpose of exercising the operating system component(s) under test. Such an approach provides a high level of control over testing. By directly programming the calls themselves, a tester is able to tailor a test to include specific inputs to APIs under specific contexts. On the other hand, testing cannot be performed until the test program has been rendered.

Another general technique for testing an operating system API avoids the need to write a particular program to act as the caller to the components under test. This alternative approach involves running application programs and observing their operation (and errors/fault) with regard to the various called components of the operating system via the API. This approach avoids the need to author an application to Submit calls to the operating system components. Furthermore, this approach ensures that at least the application used to perform the testing operates properly with regard to the particular operating system API. A potential shortcoming of this approach is the wide variety of applications that execute on an operating system and call upon its functionality through the API.

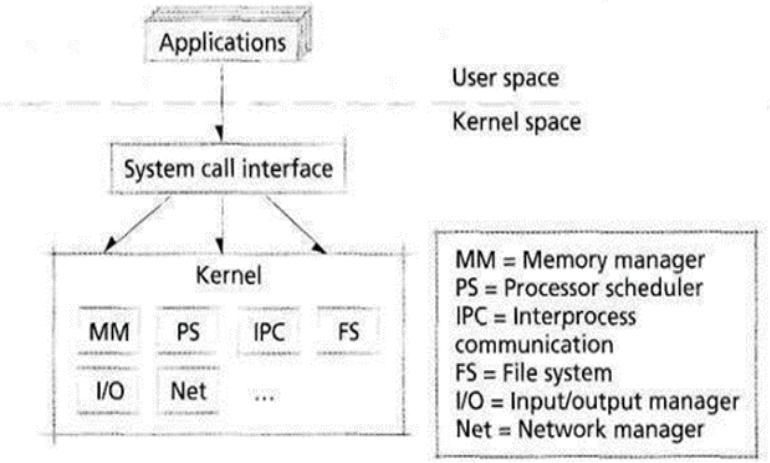
The above-described approaches to API testing, while offering particular advantages, also exhibit shortcomings including either an inability to exhaustively test an API or alternatively perform such exhaustive testing at a very high price in testing time and resources.

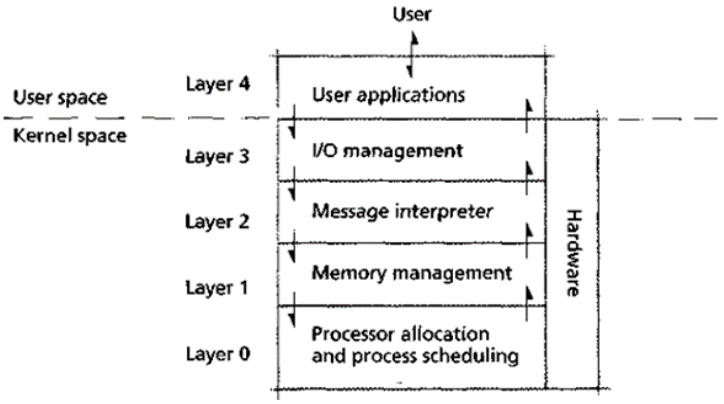
Operating System Components

A user interacts with the operating system via one or more user applications, and often through a special application called a shell, or command interpreter. The software that contains the core components of the operating system is referred to as the kernel. Typical operating system components include the processor scheduler, memory manager, I/O manager, interprocess communication (IPC) manager, and file system manager. Almost all modern operating systems support a multiprogrammed environment in which multiple applications can execute concurrently. The kernel manages the execution of processes. Program components, which execute independently but use a single memory space to share data, are called threads.

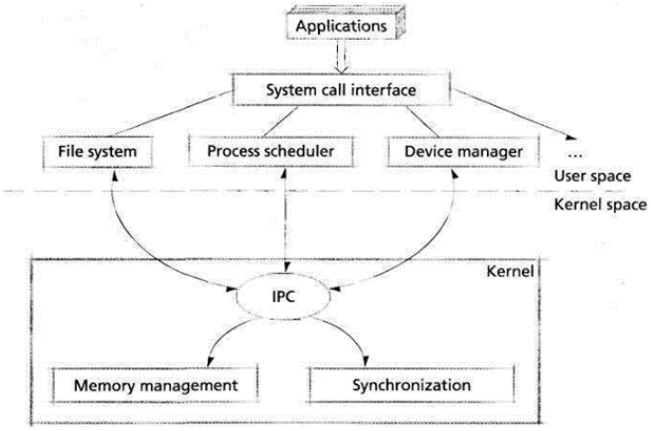
Operating System Architectures

The **monolithic operating system** is the earliest and most common operating system architecture. In a monolithic operating system, every component is contained in the kernel. As a result, any component can directly communicate with any other. Monolithic operating systems tend to be highly efficient. A disadvantage of monolithic designs is that it is difficult to determine the source of subtle errors.





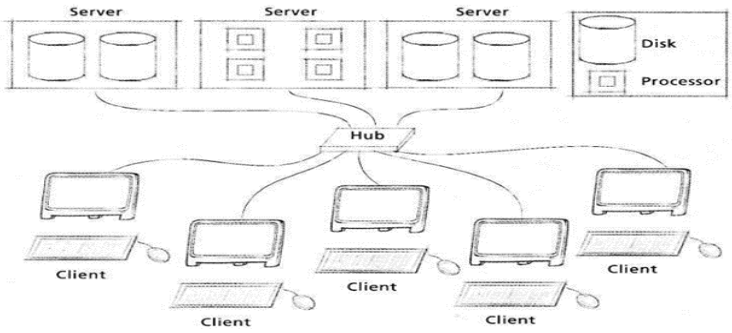
The **layered approach** to operating systems attempts to address this issue by grouping components that perform similar functions into layers. Each layer communicates exclusively with those immediately above and below it. In a layered approach, a user process's request may need to pass through many layers before completion. Because additional methods must be invoked to pass data and control from one layer to the next, system throughput decreases compared to that with a monolithic kernel, which may require only a single call to service a similar request. The THE operating system is an early example of a layered operating system.



A **microkernel operating system** architecture provides only a small number of services in an attempt to keep the kernel small and scalable. In microkernel designs, most operating system components—such as process management, networking, file system interaction and device management—execute outside the kernel with a lower privilege level. Microkernels exhibit a high degree of modularity, making them extensible, portable and scalable. Microkernels exhibit a high degree of modularity, making them extensible, portable and scalable.

However, such modularity comes at the cost of an increased level of intermodule communication, which can degrade system performance.

A **network operating system** runs on one computer and allows its processes to access resources such as files and processors on a remote computer. The structure of many networked and distributed operating systems is often based on the client/server model.



A **distributed operating system** is a single operating system that manages resources on more than one computer system. The goals of a distributed operating system include transparent performance, scalability, fault tolerance and consistency. Distributed operating systems are often difficult to implement and require complicated algorithms to enable processes to communicate and share data. Examples of distributed operating systems are MIT's Chord operating system and the Amoeba operating system from the Vrije Universiteit (VU) in Amsterdam.

Network Topology

Common topologies are:

- **Bus network**: all nodes are connected to a common medium along this medium. This was the layout used in the original Ethernet, called 10BASE5 and 10BASE2. This is still a common topology on the data link layer, although modern physical layer variants use point-to-point links instead, forming a star or a tree.
- **Star network**: all nodes are connected to a special central node. This is the typical layout found in a small switched Ethernet LAN, where each client connects to a central network switch, and logically in a wireless LAN, where each wireless client associates with the central wireless access point.
- **Ring network**: each node is connected to its left and right neighbor node, such that all nodes are connected and that each node can reach each other node by traversing nodes left- or rightwards. Token ring networks, and the Fiber Distributed Data Interface (FDDI), made use of such a topology.
- **Mesh network**: each node is connected to an arbitrary number of neighbors in such a way that there is at least one traversal from any node to any other.
- **Fully connected network**: each node is connected to every other node in the network.
- **Tree network**: nodes are arranged hierarchically. This is the natural topology for a larger Ethernet network with multiple switches and without redundant meshing.
- Networks can also be classified by the geographic dispersion of their hosts. A **local area network** (LAN) has limited geographic dispersion and is designed to optimize data transfer rates between its hosts. LANs interconnect resources using high-speed communication paths with optimized network protocols for local area environments. **Wide area networks** (WANs) are broader, connecting two or more LANs; the largest

WAN is the Internet. WANs generally employ a mesh topology, operate at lower speeds than LANs and have higher error rates, because they handle larger amounts of data being transmitted over far greater distances.

Examples of operating systems

1. Amoeba

Amoeba, developed at Vrije University, is an operating system using a microkernel design, supporting very fast message passing designed to utilize processor farms. A processor farm is a set of rack mounted single-board computers connected by regular networking (Ethernet). Amoeba makes the collection machines look like one fast timesharing system. It also provides support for threads, RPC, group communication, and all other facilities needed for networking. Amoeba supports a parallel programming language called Orca.

2. Clouds

Clouds, developed at Georgia Tech, is a system designed to support persistent objects that are large grained. Each object is an address space that is backed up on disk and hence is persistent. The system paradigm uses a thread-object model, where threads are distributed and can access objects via a modified RPC mechanism. The object invocation causes the thread to move between address spaces rather than use a server for processing the RPC request. The entire system is supported on top of a low-level Distributed Shared Memory mechanism thus making all objects available at all computers. Services are built into objects and can be accessed using the RPC mechanism. Message passing is not supported at the API level. Clouds has been used for research in reliability, transaction processing, replication and distributed debugging.

3. macOS

macOS (formerly "Mac OS X" and later "OS X") is a line of open core graphical operating systems developed, marketed, and sold by Apple Inc., the latest of which is pre-loaded on all currently shipping Macintosh computers. macOS is the successor to the original classic Mac OS, which had been Apple's primary operating system since 1984. Unlike its predecessor, macOS is a UNIX operating system built on technology that had been developed at NeXT through the second half of the 1980s and up until Apple purchased the company in early 1997. The operating system was first released in 1999 as Mac OS X Server 1.0, followed in March 2001 by a client version (Mac OS X v10.0 "Cheetah"). Since then, six more distinct "client" and "server" editions of macOS have been released, until the two were merged in OS X 10.7 "Lion".

4. Sprite

Sprite, developed at University of California – Berkeley, is an operating system that provides a single system image to a cluster of workstations. Much of the focus of research with Sprite has been directed at improving file system performance. As a result, Sprite provides a very

high-performance file system through client and server caching. It has process migration to take advantage of idle machines. It was used as a testbed for research in log-structured file systems, striped file systems, crash recovery, and RAID file systems.

5. Unix

Unix is a commercial product of Unix Systems Laboratories. Various other companies sell variants of Unix, using other trade names, the most well-known being SunOS/Solaris. SunOS was the first system to provide a commercial, robust, full-featured network file system (NFS). Linux is a free Unix compatible operating system. The kernel of Unix is monolithic and most network-based services are added as separate user processes. Unix is an older operating system, adapted for network use. Due to the prevalence of Unix in research institutions, all services developed for networking are developed on Unix platforms first. Hence, everything is available for Unix, though not from the commercial providers of Unix. Unix is the mainstay of network operating systems in the academic and research communities.

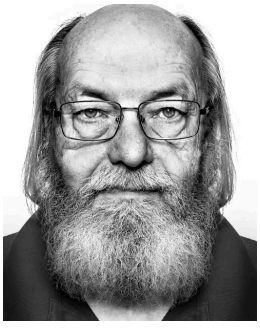
6. V-System

The V-System, developed at Stanford University, is a microkernel operating system with support for fast message passing. Services are added to V by running user-level servers. The innovative use of low latency protocols for inter-machine messaging provides V with excellent performance on a networked environment. Also innovative is the uniform support for input-output, a capability-based naming scheme and the clean design of the kernel.

7. Windows NT

Windows NT is a commercial product of Microsoft Corporation. This operating system has a hybrid kernel, that is the inner core of the operating system follows the microkernel technology, but the services are not at the user-level. Services are added to Windows NT as modules called DLLs (dynamically loadable libraries). The operating system is extensible and allows for a variety of pluggable modules at the level of device drivers, kernel extensions as well as services at the user level. Windows NT provides many of the services described in this article in a commercial product and competes with the various forms of Unix in the marketplace. Windows NT also has the ability of running applications written for DOS, Windows 3.1 and Windows 95, all of which are completely different operating systems. For network use, Windows NT provides file service, name service, replication service, RPC service and messaging using several protocols

Important people



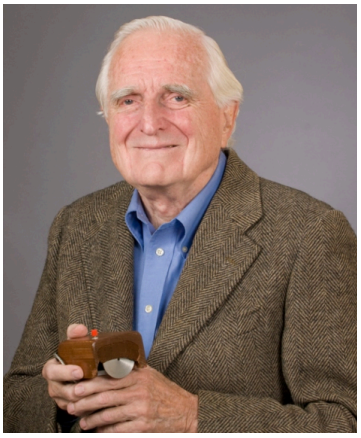
Ken Thompson and Dennis Ritchie

Ken Thompson and Dennis Ritchie are well known in the field of operating systems for their development of the UNIX operating system and the C programming language. They have received several awards and

recognition for their accomplishments, including the ACM Turing Award, the National Medal of Technology, the NEC C&C Prize, the IEEE Emmanuel Piore Award, the IEEE Hamming Medal, induction into the United States National Academy of Engineering and the Bell Labs National Fellowship. Ken Thompson attended the University of California at Berkeley, where he earned a B.S. and M.S. in Computer Science, graduating in 1966. After college Thompson worked at Bell Labs, where he eventually joined Dennis Ritchie on the Multics project. While working on that project, Thompson created the B language that led to Ritchie's C language. The Multics project eventually led to the creation of the UNIX operating system in 1969. Thompson continued to develop UNIX through the early 1970s, rewriting it in Ritchie's C programming language.

Dennis Ritchie attended Harvard University, earning a Bachelor's degree in Physics and a Ph.D. in Mathematics. Ritchie went on to work at Bell Labs, where he joined Thompson on the Multics project in 1968. Ritchie is most recognized for his C language, which he completed in 1972. Ritchie added some extra capabilities to Thompson's B language and changed the syntax to make it easier to use. Within the past 10 years he has created two new operating systems. Plan 9 and Inferno. The Plan 9 system is designed for communication and production quality. Inferno is a system intended for advanced networking.

Doug Engelbart



Doug Engelbart invented the computer mouse and was one of the primary designers of the original graphical displays and windows. Engelbart's background was in electronics. During World War II he worked as an electronics technician on a variety of systems including RADAR and SONAR. In 1968, at the Joint Computer Conference in San Francisco, Engelbart and his coworkers displayed their computer system, NLS (oNLine System) which featured Engelbart's computer mouse and a graphical interface with windows. This original mouse, called an X-Y Position Indicator for a Display Sys tern, had only one button. The mouse had two wheels on the bottom, one horizontal and one vertical, to detect movement. The mouse and the graphical windows were interdependent. Engelbart has dedicated his life to augmenting human intellect. His original idea behind the NLS system was to create a system that could help people solve problems faster. Engelbart founded the Bootstrap Institute to foster worldwide awareness of his mission. Today, Engelbart is still working with the Bootstrap Institute. He has received recognition for his work including the Lemelson-MIT Prize, the National Medal of Technology and induction into the National Inventors Hall of Fame.

Tim Berners-Lee

The World Wide Web was invented by Tim Berners-Lee in 1990. The Web allows computer users to locate and view multimedia-based documents (i.e., documents with text, graphics, animation, audio or video) on almost any subject. Berners-Lee graduated from Queen's College at Oxford University with a degree in Physics in 1976. In 1980 he wrote a program called Enquire, which used hypertext links to help him quickly navigate the numerous documents in a large project. He entered into a fellowship at the European Center for Nuclear Research (CERN) in 1984, where he gained experience in communication software for real-time networked systems. Berners-Lee invented HTTP (the HyperText Transfer Protocol), HTML (Hypertext Markup Language) and the first World Wide Web server and browser in 1989, while working at CERN. He intended the Web to be a mechanism for open, available access to all shared knowledge and experience. Until 1993, Berners-Lee individually managed changes and suggestions for HTTP and HTML, sent from the early Web users. By 1994 the Web community had grown large enough that he started the World Wide Web Consortium (W3C; www.w3.org) to monitor and establish Web technology standards. As director of the organization, he actively promotes the principle of freely available information accessed by open technologies.

Linus Torvalds

Linus Torvalds was born in 1969 in Helsinki, Finland. As a child he taught himself how to program by playing with a Commodore VIC-20. In 1988 he entered the University of Helsinki to study computer science. While there, he wrote a UNIX clone based on Professor Andrew Tanenbaum's Minix to run on his new PC. In 1991 he completed the first version of the basic Linux kernel, which ran on the Intel 80386 processor. He distributed Linux under the GNU Public License (GPL) as open source code and gladly accepted additions, corrections and free programs from other programmers. By 1994 Linux had accrued enough applications to be a complete, usable operating system, and version 1.0 was released. Programmers and professors using UNIX on large systems liked Linux because it brought the features and power of UNIX to inexpensive desktop systems for free. Linux has become one of the largest and best-known open-source developments in computing history and has become particularly successful in the server market.

Richard Stallman

Richard Stallman was the original developer of the GNU project, started in the 1980s to create free software. Stallman graduated from Harvard in 1974 with a degree in Physics. While at Harvard, he worked at the MIT Artificial Intelligence Lab. After graduating, he continued at the lab, where he and his colleagues worked with shared software. The idea was that someone receiving someone else's executable program would also receive its source code. This was advantageous because a programmer could add more functionality to someone else's program. Stallman's specific job was to modify and improve the ITS operating system the lab used. However, the lab's new operating system was not shared. In 1984 he left the MIT Artificial Intelligence Lab to work on a new shared operating system, which he called GNU (GNU's Not UNIX). As interest in Stallman's GNU project grew, he created the Free Software Foundation (FSF) in 1985 to promote free software and continue to develop the GNU operating system. Stallman and his associates at FSF created a number of GNU programs, including GNU Emacs (a text editor), GCC (a C compiler) and GDB (a debugger), to name a few. In 1992 Stallman used the Linux kernel to complete his system. The system, known as GNU/Linux, is a fully functional operating system and includes a variety of programs. Stallman has received numerous awards and recognition for his work, including the Grace Hopper Award, MacArthur Foundation Fellowship, the Electric Frontier Foundation Pioneer Award, the Yuri Rubinksi Award, the Takeda Award, election to the National Academy of Engineering and two honorary doctorates from the Institute of Technology in Sweden and the University of Glasgow.

Connection to other areas

Operating systems (OS) and networks intertwined with almost every other area of computer science. Together, they form the backbone of computational environments, ensuring that resources are utilized efficiently and communication remains seamless and secure. Here’s an overview of how they relate to other areas of computer science:

1. Programming Languages and Compilers

- OS: Programming languages and compilers need to interact closely with the operating system to manage resources such as memory, CPU, and I/O. Operating systems provide the runtime environment where programs execute.

- Networks: Network programming often requires specific libraries and protocols that are supported by both the programming language and the OS. Sockets and APIs are essential for enabling communication between distributed systems.

2. Computer Architecture

- OS: The design of an operating system is heavily influenced by the underlying hardware. OS must efficiently manage hardware resources, such as CPU scheduling, memory management, and I/O operations.

- Networks: Network interfaces and protocols are often implemented with a deep understanding of hardware capabilities, such as network cards and routers, to optimize data transfer rates and reduce latency.

3. Databases

- OS: Databases rely on operating systems for managing disk storage, memory, and process scheduling. The OS ensures data integrity and security through mechanisms like file systems and user permissions.

- Networks: Modern databases often run on distributed systems, necessitating robust network communication protocols for data replication, synchronization, and transaction management across different nodes.

4. Security

- OS: Operating systems are responsible for enforcing security policies, managing user authentication, access control, and protecting against malicious software through various security mechanisms like firewalls and antivirus programs.

- Networks: Network security is crucial for protecting data in transit. This includes the implementation of encryption protocols, secure communication channels (like VPNs), and intrusion detection systems.

5. Software Engineering

- OS: Software engineering practices must consider the operating system environment where the software will run. This includes ensuring compatibility and efficient resource usage.

- Networks: In distributed systems and cloud computing, network reliability and efficiency are critical. Software engineers need to design applications that can handle network failures and optimize data transfer.

6. Artificial Intelligence and Machine Learning

- OS: AI and ML applications often require significant computational resources, managed by the operating system to optimize performance, particularly in parallel and distributed computing environments.

- Networks: AI/ML models often require large datasets that might be stored in distributed systems. Efficient network protocols are necessary to transfer data for training and inference processes, especially in edge computing.

7. Human-Computer Interaction (HCI)

- OS: The operating system plays a key role in managing the graphical user interface (GUI) and ensuring that user inputs are processed efficiently.

- Networks: HCI in networked applications involves ensuring that communication latency does not degrade the user experience, which is particularly important in interactive and real-time applications like online gaming and video conferencing.

8. Distributed Systems

- OS: The operating system's role in distributed systems is to manage the local resources and support communication between different systems, enabling distributed computing environments.

- Networks: Networking is foundational for distributed systems, providing the protocols and infrastructure necessary for inter-node communication, data consistency, and fault tolerance.

9. Theoretical Computer Science

- OS: Theoretical models such as automata theory and formal verification methods can be applied to ensure the correctness and efficiency of operating systems.

- Networks: Theoretical concepts like graph theory and queuing theory are used to analyze and optimize network performance and reliability.

Important publications

A Protocol for Packet Network Intercommunication

- Vint Cerf, Robert Kahn
- IEEE Transactions on Communications, 1974.

Description: This paper contains a lot of the ideas that later became TCP and IP, two foundational protocols that make up the Internet. Cerf and Kahn received the ACM Turing Award, in part for the work contained in this paper.

Congestion Avoidance and Control

- Van Jacobson, Michael J. Karels
- ACM SIGCOMM, 1988.

Description: This paper identifies the problem of network congestion, and presents an algorithm for how protocols can reduce their sending rate to reduce congestion. This approach was incorporated into the TCP protocol, and influenced the design of many other data transport protocols.

An experimental timesharing system

- Fernando J. Corbató, M. Merwin-Daggett, and R.C. Daley
- Proceedings of the AFIPS FJCC, pages 335–344, 1962.

Description: This paper discusses time-sharing as a method of sharing computer resources. This idea changed the interaction with computer systems.

The Working Set Model for Program Behavior

- Peter J. Denning
- Communications of the ACM, Vol. 11, No. 5, May 1968, pp 323–333

Description: The beginning of cache. For more information see SIGOPS Hall of Fame.

Virtual Memory, Processes, and Sharing in MULTICS

- Robert C. Daley, Jack B. Dennis
- Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 306–312.

Description: The classic paper on Multics, the most ambitious operating system in the early history of computing. Difficult reading, but it describes the implications of trying to build a system that takes information sharing to its logical extreme. Most operating systems since Multics have incorporated a subset of its facilities.

The nucleus of a multiprogramming system

- Per Brinch Hansen
- Communications of the ACM, Vol. 13, No. 4, April 1970, pp. 238–242

Description: Classic paper on the extensible nucleus architecture of the RC 4000 multiprogramming system, and what became known as the operating system kernel and microkernel architecture.

Operating System Principles

- Per Brinch Hansen
- Prentice Hall, Englewood Cliffs, NJ, July 1973

Description: The first comprehensive textbook on operating systems. Includes the first monitor notation (Chapter 7).

The UNIX Time-Sharing System

- Dennis M. Ritchie and Ken Thompson
- Communications of the ACM 17(7), July 1974.

Description: The Unix operating system and its principles were described in this paper. The main importance is not of the paper but of the operating system, which had a tremendous effect on operating systems and computer technology.

A Fast File System for UNIX

- Marshall Kirk Mckusick, William N. Joy, Samuel J. Leffler, Robert S. Fabry
- IACM Transactions on Computer Systems, Vol. 2, No. 3, August 1984, pp. 181–197.

Description: The file system of UNIX. One of the first papers discussing how to manage disk storage for high-performance file systems. Most file-system research since this paper has been influenced by it, and most high-performance file systems of the last 20 years incorporate techniques from this paper.

Microkernel operating system architecture and Mach

- David L. Black, David B. Golub, Daniel P. Julin, Richard F. Rashid, Richard P. Draves, Randall W. Dean, Alessandro Forin, Joseph Barrera, Hideyuki Tokuda, Gerald Malan, David Bohman
- Proceedings of the USENIX Workshop on Microkernels and Other Kernel Architectures, pages 11–30, April 1992.

Description: This is a good paper discussing one particular microkernel architecture and contrasting it with monolithic kernel design. Mach underlies Mac OS X, and its layered architecture had a significant impact on the design of the Windows NT kernel and modern microkernels like L4.

UVT Relationship

The university's curriculum effectively covers the topics of operating systems and networks for several reasons:

Operating Systems

1. **Core Curriculum Inclusion**:
    - The courses "Operating Systems I" and "Operating Systems II" are included as mandatory parts of the curriculum. This ensures that every student receives foundational and advanced knowledge in this crucial area.
2. **Comprehensive Coverage**:
    - **Operating Systems I**: This course introduces fundamental concepts such as process management, memory management, and file systems, which are essential for understanding how modern operating systems function. This foundational knowledge is crucial for any computing professional.
    - **Operating Systems II**: The continuation course dives deeper into more complex topics like concurrency, advanced memory management techniques, and system

security. These are advanced topics that build on the basics covered in the first course, ensuring a well-rounded education in operating systems.

3. **Practical Application**:
    - Both courses incorporate practical lab sessions. This hands-on experience is vital for students to apply theoretical knowledge to real-world scenarios, such as configuring and managing operating systems.

Networks

1. **Focused Coursework**:
    - The university offers specific courses like "Computer Networks" and "Network Administration" which directly address the topic. These courses ensure that students gain both theoretical and practical knowledge about network design, implementation, and management.
2. **Depth and Breadth**:
    - **Computer Networks**: This course covers essential networking concepts including the OSI model, TCP/IP protocols, IP addressing, and subnetting. These foundational topics are crucial for understanding how different network components interact and communicate.
    - **Network Administration**: Building on the basics, this course delves into the administration and management of networks, covering advanced topics such as network security, wireless networks, and IoT. This prepares students to handle complex networking tasks and challenges.
3. **Hands-On Labs**:
    - Both courses include laboratory sessions where students engage in practical tasks such as configuring routers, analyzing network traffic with tools like Wireshark, and implementing client-server models. This practical approach ensures students can apply their knowledge in real-world networking environments.

Future of OS and Networks

Future operating systems are likely to integrate advanced security mechanisms, such as hardware-rooted security, secure enclaves, and AI-driven threat detection, to provide robust protection against evolving cyber threats. As quantum computing matures, there will be a need for operating systems to manage quantum processors, support quantum programming languages, and address new challenges in quantum algorithms and error correction.

As technology becomes more integrated into our lives, operating systems face increasing threats from cyber attacks, data breaches, and privacy infringements. Developing robust security measures and maintaining user privacy are ongoing challenges.

Artificial intelligence and automation will likely play a significant role in future operating systems, optimizing resource management, automating system maintenance tasks, and providing personalized user experiences. Future operating systems will need to support cutting-edge

innovations in Augmented Reality (AR), Virtual Reality (VR), and blockchain while also delivering seamless integration and improved performance for these novel paradigms.

Due to its reliance on human commands, the user interface for the next generation of operating systems is of utmost importance. The use of Command Line Interfaces (CLIs) in the past has been replaced by Graphical User Interfaces (GUIs) in modern systems. Along with the Graphical User Interface, the next generation of operating systems may potentially include Audio User Interfaces (AUIs) and User Gesture Interfaces (UGIs).

References:

1. Peter J. Denning, *Computer science: the discipline* (August 1997)
2. H. M. Deitel, P. J. Deitel, D. R. Choffnes, *Operating Systems Third Edition*
3. Partha Dasgupta *Network Operating Systems* (1997)
4. Stallings (2005). *Operating Systems, Internals and Design Principles*. Pearson: Prentice Hall. p. 6.
5. List of important publications in computer science - Wikipedia
6. Robert Bazuku, Anaamotulim Anab, Seth Gyemerah and Mohammed I. Daabo *An Overview of Computer Operating Systems and Emerging Trends*

Databases and Information Retrieval

SEBASTIAN

Link to LaTeX: https://www.overleaf.com/read/bfjrnwcsyfqt#80ab90

AI & Robotics

MAIA

1969maia/AI-and-Robotics at 74e4fca8924c6c18817393fc06aeef88cf3ee826 (github.com)

Graphics

ANA

GRAPHICS

What is graphics?

According to Donald Hearn, computer graphics is the creation of visual content through the use of algorithms and data structures, enabling the representation, manipulation, and rendering of images on digital devices.

Graphics can be divided into multiple categories based on various criteria such as their purpose, used techniques, and their application domains. The most common and popular categories are:

1. 2D Graphics

2D graphics revolve around visually representing objects and their motions on a computer's screen. It encompasses the creation of images, the projection of motions onto screens in real-time, and the presentation of complex data sets. It finds application in graphic design, digital art, user interface design, and the creation of virtual realities that closely mimic real-world scenarios.

2. 3D Graphics

3D graphics create images and animations with depth, simulating three-dimensional space. Unlike 2D graphics, which are flat, 3D graphics add depth and realism to images, making them suitable for games, movies, and virtual reality experiences.

3. Vector Graphics

Vector graphics use math to create shapes and lines, which can be resized without losing quality. They're great for logos, icons, and illustrations.

4. Raster Graphics

Raster graphics are made of pixels and are great for detailed images like photos and paintings. They're used in digital photography, web graphics, and printing.

5. Rendering Techniques

Rendering techniques are methods for turning 3D models into images. They add things like shading and lighting to make images look realistic or stylized.

6. Interactive Graphics

Interactive graphics let users interact with digital content in real-time. They're used in things like games, educational software, and interactive websites.

Essential Graphics Tools:
Techniques and Resources for Visual Creation:

The most common used tools to create and manage graphics are:

* **Graphics Software:** Programs like Adobe Photoshop, Illustrator, and CorelDRAW are widely used for creating and editing graphics.

* **3D Modeling Software**: Tools such as Autodesk Maya, Blender, and Cinema 4D are used for creating 3D models and animations.

* **Image Editing Software:** Programs like GIMP and Paint.NET offer features for editing and manipulating raster graphics.

* **Vector Graphics Editors:** Software such as Adobe Illustrator and Inkscape allows for the creation and editing of vector graphics.

* **3D Rendering Engines:** Engines like Unity and Unreal Engine are used for rendering 3D graphics in real-time applications, such as games and simulations.

* **Graphic Tablets:** Devices like Wacom tablets enable digital artists to draw directly onto the computer screen with precision and control.

* **Color Management Tools:** Tools for managing color profiles and calibration ensure consistent color representation across different devices and mediums.

The most common used techniques used by graphic artists are:

* **Rasterization:** The process of converting vector graphics or 3D models into raster images for display on screens or printing.

* **Ray Tracing**: A rendering technique that simulates the behavior of light to create realistic lighting and reflections in 3D scenes.

* **Shading:** Techniques for adding depth and realism to objects by applying colors and textures based on lighting conditions.

* **Texturing:** Adding surface textures to 3D models to enhance their appearance and realism.

* **Animation:** Creating sequences of images or frames to simulate motion or change over time.

* **Compositing:** Combining multiple images or elements to create a final composite image or scene.

* **Rendering Pipeline:** A series of stages that graphics data goes through, including geometry processing, rasterization, shading, and output.

* **Image Compression:** Techniques for reducing the size of image files without significant loss of quality, such as JPEG compression.

Donald Hearn and M. Pauline Baker claim that: "The best tool for graphics is the one that empowers creativity while seamlessly integrating with the artist's workflow." , Meaning that there is no ‘right’ tool or technique for graphic artists, that every artist should used that inspires them the most, helping them create something in their own style.

Important people in the graphic design field:

Ivan Sutherland

In 1963, Ivan Sutherland engineered a revolution in computer graphics with his highly-interactive program Sketchpad. It enabled users to design and draw in real time directly on the computer display, using a light pen. Among other innovative features, Sketchpad allowed users to manipulate, duplicate, store, and recall drawings for future use.

Publications: https://dblp.org/pid/35/5713.html

Ed Catmull

Edwin Earl "Ed" Catmull is one of the three founding fathers of Pixar. He was President and CTO of Pixar and later President of Walt Disney Animation Studios and Pixar Animation Studios, until his retirement in 2019. Ed Catmull has played a pivotal role in advancing computer graphics and computer-generated imagery (CGI) in animated films.

Publications: https://dblp.org/search?q=Ed+Catmull

Jim Blinn

Edwin Earl "Ed" Catmull is one of the three founding fathers of Pixar. He was President and CTO of Pixar and later President of Walt Disney Animation Studios and Pixar Animation Studios, until his retirement in 2019. Ed Catmull has played a pivotal role in advancing computer graphics and computer-generated imagery (CGI) in animated films.

Homepage: https://www.jimblinn.com
Publications: https://www.jimblinn.com/publications/
Contact: https://www.jimblinn.com/contact/

James Kajiya

Jim Kajiya is a pioneer in the field of computer graphics. He is perhaps best known for the development of the rendering equation. Kajiya received his PhD from the University of Utah in 1979, was a professor at Caltech from 1979 through 1994, and is currently a researcher at Microsoft Research

Publications: https://dblp.org/pid/43/808.html

Donald Greenberg

Donald Greenberg is a computer graphics researcher and teacher, who is professor of Cornell University's program of computer graphics. He was a computer graphics technical consultant for the animation studio Hanna-Barbera.

Homepage: https://www.graphics.cornell.edu/people/director
Publications: https://dblp.org/pid/g/DonaldPGreenberg.html
Email: dpg5@cornell.edu
phone number: (607) 255-7444

Choosing a career in graphics can be exciting and rewarding for many reasons. It can allow individuals to use their creativity, whether it's through designing eye-catching advertisements, creating immersive videos for ads or animations for tv series and cartoons, or adding visual effects to movies. Moreover, the field is always evolving, offering opportunities for continuous learning and professional growth. Graphics professionals also enjoy flexibility in their work environments, with options to freelance, work remotely, or collaborate with diverse teams across different industries.

However, it's important to be aware of the not-so-great sides in the graphics field. Competition is one of them, requiring individuals to build strong portfolios and continuously improve or adapt their skills to stand out or to fit in. Additionally, meeting tight deadlines and satisfying client expectations are a difficult task, recreating the exact image that the client invisions in their head.

sources:
"Computer Graphics: C Version" - Donald Hearn and M. Pauline Baker
National Inventors Hall Of Fame - https://www.invent.org
DBLP - https://dblp.org
Computer Science: The Discipline - Peter J. Denning
Creative Bloq - https://www.creativebloq.com/tag/graphic-design

Human-Computer Interaction

MIRUNA

What is this interaction?

Human-Computer Interaction, HCI for short, is a field that studies the efficient coordination and transfer of information between computers and humans with a keen interest in user interfaces and performance.

— MAIN ACTIVITIES —

Theory

In this area, theoreticians aim mainly at bringing order into a rapidly accumulating mass of experience through broad conceptual frameworks, taxonomies, and analytic methods.

The theory of HCI is mainly based on cognitive psychology and risk analysis. The first one is mandatory for understanding the human mind and how it reacts to different displays. In this way, designers can evaluate if humans misinterpret given information while being pressured. Meanwhile, risk analysis is important since most of the user interfaces monitor complex, safety-critical systems. Some of the main areas of Human-Computer Interaction are:

1. **Statistics:** Study of methods for collecting, analyzing, interpreting and presenting factual data.
2. **Probability:** Measurement of the chances of an event to occur.
3. **Queuing theory:** Study of how queues form, work and malfunction.
4. **Coordination theory:** Identification of the dependency between the tasks carried out by different group members and the coordination mechanism used by the group; coming up with alternative mechanisms.

Experiment

Exploring models of systems and architectures within given application domains and testing whether those models can predict new behaviors accurately (= abstraction). HCI relies heavily on laboratories and often stimulates new developments in computer design and use.

When it comes to Human-Computer Interaction, models and associated measurements are critical.

Computer-Aided Design (CAD)

CAD represents the use of computer systems in order to create, modify, optimize a design. The main purpose of this software is to increase the designer's productivity, while also improving the quality of the design and creating a database for manufacturing. The output of CAD is usually in the form of electronic files.

Sophisticated image processing and methods for enhancement have been developed to allow the interpretation of photographs from deep-space probes to human CT and MRI scans.

Experimental studies led to the deduction of principles for designing displays and control panels for resistance to human misinterpretation.

Design

Designers are concerned with building systems that meet clear specifications and satisfy their customers. They boast many significant accomplishments.

Represents the main focus of the field. Usability engineering = processes of engineering design used for the user interfaces. Input, output, multimedia, have been developed in a sophisticated manner.

In manufacturing (and computer chip design), CAD systems are still the most commonly used, and smaller versions of these systems are accessible for one's personal computer. The user interface design area has seen a lot of progress, now using icons and menus for display and a mouse for the input device, setting a popular standard to be followed.

However, it does not stop here as modern interfaces are now revolving around computers controlled using graphic tablets or voice input.

Trends in emerging HCI

- Speech recognition
- gesture recognition
- gaming platforms (for example, wii)
- emotion detection
- virtual assistants
- augmented reality (overlaying information on top of a real world view)

NASA has used flight simulation systems for years to train its future pilots. Smaller versions of it are available for personal computers.

Distributed Interactive Simulation (DIS)

DIS systems are usually used in defense applications to train people how to handle battlefield situations. These systems allow users to interact with one another, as well as their environment, over a network, by presenting a real-time image of the world seen from the participant's perspective.

— *IMPORTANT PROBLEMS* —

1. **Learnability:** creating designs that comply with the conventions of other similar ones, making them predictable in navigation and overall layout, to make it easier for the user to understand how to use it. In these cases, simplicity is key. A usability test may be used on the same user at a certain interval, testing how memorable the application is. Feedback can then be used to increase the chances of success on the market.
2. **Meaningful human control:** defined as one of the short-term research priorities for AI. This is a principle that has to do with humans, not computers, supporting the fact that in the end, we will be morally responsible for actions mediated by autonomous

systems. Important features are transparency, accountability, building trust between the user and the system.

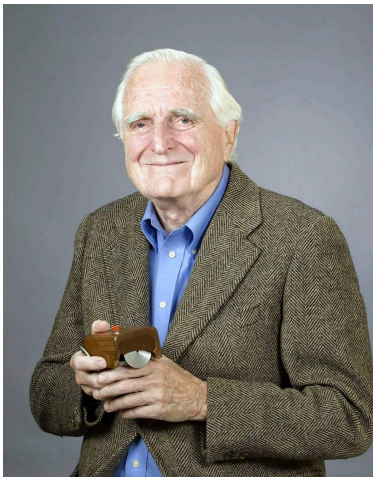
3. **Humane digital experience:** creating a technology that supports individual and social life, respects human rights, incorporates human values in the design choices, enabling humans to exploit their potential (creative, social or economic)

4. **Adaptation to human needs:** making the intelligent environments act and support humans by providing personalized, insightful, responsive services

5. **Emotion detection:** since humans are emotional beings, an important challenge for optimizing human-computer interaction is working out how technology can express emotions, as it has the potential to affect humans psychologically.

6. **Human safety:** the development of intelligent systems that have the potential to self-evolve come with some risk due to technology misuse or poor design. Therefore, the systems should be "safe-by-design".

— IMPORTANT PEOPLE —



Douglas Engelbart

Douglas Engelbart (1925 - 2013) was an American inventor whose work beginning in the 1950s led to his invention of the computer mouse, the development of the GUI (graphic user interface) and groupware.

In 1997, he won the A.M. Turing Award, the highest honor in Computer Science, for his "inspiring vision of the future of interactive computing and the invention of key technologies to help realize this vision."

Donald A. Norman

Don Norman is a cognitive scientist, an electrical and usability engineer known for his work on human-centered design. His book "The Design of

Everyday Things" has made significant contributions to the HCI field.

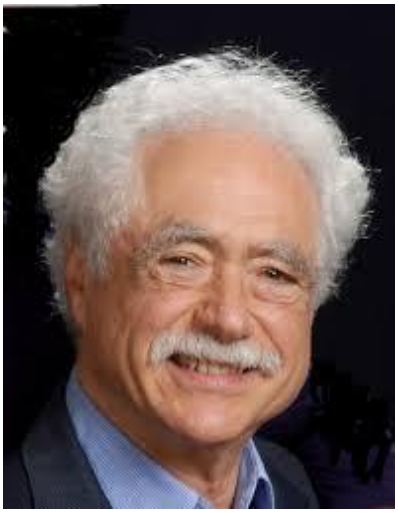
Website: https://jnd.org/

Ben Shneiderman

Ben Shneiderman is a professor at the University of Maryland. He was the one who developed the direct manipulation interface paradigm, which simplifies user interaction. Therefore, his work influenced modern GUI's, making them easier to use.
Website: https://www.cs.umd.edu/~ben/
E-mail: ben@cs.umd.edu

Sherry Turkle

Sherry Turkle is a psychologist and a professor at MIT, known for her research on the social aspects of human-computer interaction, especially the ways in which technology shapes human identities and relationships.
Website: https://sherryturkle.mit.edu/
E-mail: sturkle@medi.mit.edu



Terry Winograd

Terry Winograd is a professor at Stanford University, known for his work on natural language using the SHRDLU, an early AI program. His research has influenced the way we interact with computers using AI.
Website: https://hci.stanford.edu/winograd/
E-mail: winograd@cs.stanford.edu

Hiroshi Ishii

Hiroshi Ishii is a Japanese computer scientist and a professor at MIT. He is known for developing the concept of tangible user interfaces (TUI's). His work integrates physical objects with digital information, enhancing the interaction between the real world and technology.

Website:
https://web.media.mit.edu/~ishii/index.html%2097

E-mail: ishii@media.mit.edu

— IMPORTANT JOURNALS & PUBLICATIONS —

- ***ACM Transactions on Computer-Human Interaction (TOCHI)***
  Description: the journal covers the software, hardware and human aspects of interaction with computers. Topics include hardware and software architectures; interactive techniques, metaphors, and evaluation; user interface design processes; and users and groups of users.

- ***International Journal of Human-Computer Studies (IJHCS)***
  Description: in this journal, research is centered on the interaction between people and computers, emphasizing both the technical and human aspects of the interface.

- ***Human-Computer Interaction***
  Description: a multidisciplinary journal defining and reporting on fundamental research in human-computer interaction. It is known for its comprehensive reviews and empirical research studies.

- ***Interacting with Computers***
  Description: a journal that addresses the design and use of interactive systems, providing insight into user experience and interface design.

- ***Computers in Human Behavior***

  Description: a scholarly journal dedicated to examining the use of computers from a psychological perspective. The journal addresses both the use of computers in

psychology, psychiatry and related disciplines as well as the psychological impact of computer use on individuals, groups and society.

Publications:

	Publication	h5-index	h5-median
1.	Computer Human Interaction (CHI)	122	174
2.	Proceedings of the ACM on Human-Computer Interaction	71	113
3.	Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies	63	89
4.	IEEE Transactions on Affective Computing	62	106
5.	International Journal of Human-Computer Studies	62	96
6.	Behaviour & Information Technology	55	76
7.	ACM/IEEE International Conference on Human Robot Interaction	55	72
8.	International Journal of Human-Computer Interaction	54	95
9.	Virtual Reality	49	85
10.	International Conference on Intelligent User Interfaces (IUI)	47	73
11.	International Journal of Interactive Mobile Technologies	45	65
12.	Designing Interactive Systems Conference	44	65
13.	ACM Symposium on User Interface Software and Technology	44	62
14.	IEEE Virtual Reality Conference	44	55
15.	IEEE Transactions on Human-Machine Systems	42	65
16.	Universal Access in the Information Society	42	65

17.	ACM Transactions on Computer-Human Interaction (TOCHI)	42	58
18.	HCI International	38	55
19.	International Journal of Child-Computer Interaction	35	58
20.	Multimodal Technologies and Interaction	34	57

— CONNECTIONS TO OTHER AREAS —

Human-Computer Interaction is a field that covers a large range of areas, either in Computer Science:

- Artificial Intelligence: AI techniques can be used to create user interfaces that adapt to the user's behaviors and preferences, as well as enable the development of chatbots and voice-controlled interfaces.
- Robotics: HCI principles are used for interacting with robots, making them more user-friendly, while also having them respond to human social cues, thus improving their integration into human environments.
- Software Engineering: engineering contributes to HCI by optimizing performance and user experience
- Graphics: HCI research helps the design of GUI's that are visually appealing and interactive visualizations that help users understand data that's more complex
- Information Retrieval: HCI helps design search interfaces that improve an user's ability of efficiently finding information

Or outside of it:

- Cognitive sciences: these sciences study how the mind works, proving to be essential in designing interfaces that are intuitive and easy to use
- Psychology: crucial for designing interfaces that meet users' needs and preferences
-

— @UVT —

— *REFERENCES* —

1. Tucker, A. B. (2004). *Computer science handbook*

2. Crowston, K., Howison, J., & Rubleske, J. (2005) *Coordination Theory: A Ten-Year Retrospective*

3. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-hci/

4. Sadra Haghighi, Ideen. (2022). *Computer Aided Design (CAD)*

5. Fullford, D. A. (1996). Distributed interactive simulation. *Proceedings of the 28th Conference on Winter Simulation - WSC '96*

6. Raymond, O. U. (2016). *Human Computer Interaction:Overview and Challenges*.

7. Stephanidis, C., Salvendy, G., et al. (2019). Seven HCI Grand Challenges. *International Journal of Human-Computer Interaction*

Computational Science (Theoretical Computer Science)

IONUT

What is computational science?

Alongside theory and experiment, computation is considered to be the third approach to science, it being use to solve hard problems, sometimes referred to as “grand challenges” (e.g., demonstrating existence of certain quarks, numerical simulation of the air flow field around an airplane in flight, joining DNA sequence fragments into the full human genome, microscopy, tomography, crystallography, and protein folding).

Computational science is an interdisciplinary field which deals with studying problems in science and engineering that require high-performance computation and communications. It deals with general methods of efficiently and accurately solving equations resulting from mathematical models of physical systems, for example: airflow around wings, water flow around obstacles, petroleum flow in earth's crust, plasma flow from stars, weather progression, and galactic collisions.

Computational science makes great use of mathematics:

- number theory deals with finite, binary representations of numbers and error propagation in arithmetic calculations;
- linear algebra deals with solving systems of linear equations that expressed as matrices;
- nonlinear dynamics deals with chaotic systems.

Supporting mathematics include: calculus, real analysis, complex analysis, discrete mathematics, and linear algebra.

Other supporting theory: parallel algorithms, optimizing compilers, distributed computation, organization of large data sets, automatic discovery in data, computational geometry, graphics (often, in this context, called scientific visualization), statistics.

Uses of computational science:

Computational science has led scientists to creating many useful systems such as: Chem, Web, Linpack, Eispack, Ellpack, Macsyma, Mathematica, Maple, and Reduce. They have also contributed models and algorithms in a multitude of other fields, such as:

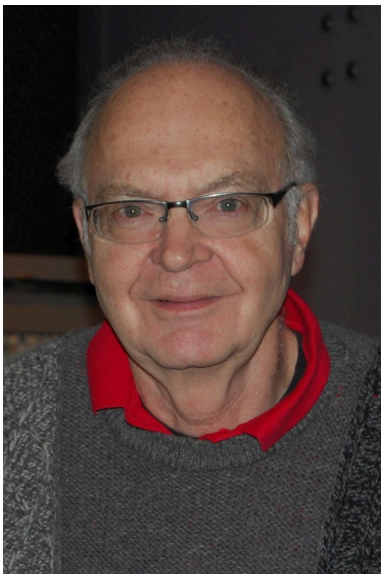
- **physics** (e.g., demonstrating existence of certain quarks)
- **aerodynamics and flow dynamics** (e.g., numerical simulation of the air flow field around an airplane in flight)
- **chemistry** (e.g., designing enzymes and proteins that selectively attack viruses)
- **biology** (e.g., joining DNA sequence fragments into the full human genome, microscopy, tomography, crystallography, and protein folding)
- **geology** (e.g., predicting earthquakes), astronomy (e.g., locating the missing mass of the universe), meteorology (e.g., long term weather forecasting)
- **engineering** (e.g., interaction between control surfaces and dynamic stress movements in structures)

Many of these “grand challenges” have been solved thanks to the support offered by the USA.

Important people in computational science:

John von Neumann, often considered the father of modern computing, developed the architecture of digital computers, known as the von Neumann architecture, which describes a system where a computer's program and the data it processes are stored in the same memory.

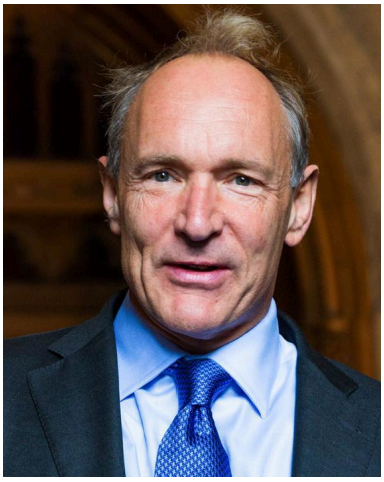
Alan Turing, he was a pioneer in theoretical computer science and artificial intelligence. His concept of the Turing machine provided a formal foundation for the theory of computation and algorithms. He also played a crucial role in breaking the Enigma code during World War II.



Donald Knuth, best known as the author of “The Art of Computer Programming,” also developed the TeX typesetting system.

Leslie Valiant is known for his work in computational complexity theory, particularly the theory of PAC (Probably Approximately Correct) learning, which is fundamental to machine learning and artificial intelligence.

Tim Berners-Lee is the inventor of the World Wide Web, which revolutionized how information is shared and accessed across the globe.



Important conferences and journals in computational science:

- The *International Conference on Computational Science (ICCS), first held in 2001, and has been organized yearly ever since*
- The Journal of Computational Science, published in 2010
- The Journal of Open Research Software, launched in 2012
- The ReScience C initiative, started on GitHub in 2015

Connections to other fields in computer science:

- **Predictive computational science**
    - Predictive computational science is a scientific discipline concerned with the formulation, calibration, numerical solution, and validation of mathematical models designed to predict specific aspects of physical events, given initial and boundary conditions, and a set of characterizing parameters and associated uncertainties.
- **Urban complex systems**
    - Urban complex systems try to understand how to continue the future development of cities, in order to prevent disasters or tragedies and to mitigate future challanges.
- **Computational finance**
    - Computational finance deals with problems of practical interest in finance
- **Computational biology**
    - Computational biology refers to the use of data analysis, mathematical modeling and computational simulations to understand biological systems and relationships.
- **Computational science and engineering**
    - Computational Engineering is an emerging discipline that deals with the development and application of computational models for engineering, known as Computational Engineering Models or CEM.

References:

Rüdiger U. Seydel, *Tools for Computational Finance*, Springer; 3rd edition (May 11, 2006) 978-3540279235

"NIH working definition of bioinformatics and computational biology" (PDF). Biomedical Information Science and Technology Initiative. 17 July 2000. Archived from the original (PDF) on 5 September 2012. Retrieved 18 August 2012.

"Computational Engineering Models for the Design of Mechanical Counterpressure Spacesuits". 2022-12-21. Archived from the original on 2022-12-21. Retrieved 2023-06-27.

https://www.rasmussen.edu/degrees/technology/blog/famous-computer-scientists-who-impacted-the-industry/

https://www.turing.com/blog/famous-computer-scientists-and-their-inventions/

Organizational Informatics

IRINEL

What does organizational informatics study?

Information systems are the primary focus of study for organizational informatics.

An information system (IS) is a formal, sociotechnical, organizational system designed to collect, process, store, and distribute information. From a sociotechnical perspective, information systems are composed by four components: task, people, structure (or roles), and technology. Information systems can be defined as an integration of components for collection, storage and processing of data of which the data is used to provide information, contribute to knowledge as well as digital products that facilitate decision making.

A computer information system is a system that is composed of people and computers that processes or interprets information.[1] The term is also sometimes used to simply refer to a computer system with software installed.

"Information systems" is also an academic field study about systems with a specific reference to information and the complementary networks of computer hardware and software that people and organizations use to collect, filter, process, create and also distribute data. An emphasis is placed on an information system having a definitive boundary, users, processors, storage, inputs, outputs and the aforementioned communication networks.

In many organizations, the department or unit responsible for information systems and data processing is known as "information services".

Any specific information system aims to support operations, management and decision-making. An information system is the information and communication technology (ICT) that an organization uses, and also the way in which people interact with this technology in support of business processes.

Some authors make a clear distinction between information systems, computer systems, and business processes. Information systems typically include an ICT component but are not purely concerned with ICT, focusing instead on the end-use of information technology. Information systems are also different from business processes. Information systems help to control the performance of business processes.

Alter argues for advantages of viewing an information system as a special type of work system. A work system is a system in which humans or machines perform processes and activities using resources to produce specific products or services for customers. An information system is a work system whose activities are devoted to capturing, transmitting, storing, retrieving, manipulating and displaying information.

As such, information systems interrelate with data systems on the one hand and activity systems on the other. An information system is a form of communication system in which data represent and are processed as a form of social memory. An information system can also be considered a semi-formal language which supports human decision making and action..

Important experiments

Minnesota Experiments

The use of computer based information-decision systems to support decision making in organizations has increased significantly in the last decade. Very little effort has been devoted, however, to determine what relationships exist between the structure of information presented for decision making and the ensuing effectiveness of the decision. This article summarizes a series of experiments. The Minnesota Experiments, which were conducted to examine the significance of various information system characteristics on decision activity. Several research programs administered in the period 1970–1975 are discussed in this paper. By varying the manner in which information was provided to participants in each experiment, the impact of various information system characteristics and individual differences on decision effectiveness was investigated. Analysis of the results shows that, in many cases, the

decisions/decision-making process of the participants was affected by the information system structure and/or attributes of individual decision makers. The results suggest guidelines for the designers of information systems and fruitful avenues for continued research.

Online controlled experiments at large scale

Web-facing companies, including Amazon, eBay, Etsy, Facebook, Google, Groupon, Intuit, LinkedIn, Microsoft, Netflix, Shop Direct, StumbleUpon, Yahoo, and Zynga use online controlled experiments to guide product development and accelerate innovation. At Microsoft's Bing, the use of controlled experiments has grown exponentially over time, with over 200 concurrent experiments now running on any given day. Running experiments at large scale requires addressing multiple challenges in three areas: cultural/organizational, engineering, and trustworthiness. On the cultural and organizational front, the larger organization needs to learn the reasons for running controlled experiments and the tradeoffs between controlled experiments and other methods of evaluating ideas. We discuss why negative experiments, which degrade the user experience short term, should be run, given the learning value and long-term benefits. On the engineering side, we architected a highly scalable system, able to handle data at massive scale: hundreds of concurrent experiments, each containing millions of users. Classical testing and debugging techniques no longer apply when there are billions of live variants of the site, so alerts are used to identify issues rather than relying on heavy up-front testing. On the trustworthiness front, we have a high occurrence of false positives that we address, and we alert experimenters to statistical interactions between experiments. The Bing Experimentation System is credited with having accelerated innovation and increased annual revenues by hundreds of millions of dollars, by allowing us to find and focus on key ideas evaluated through thousands of controlled experiments. A 1% improvement to revenue equals more than $10M annually in the US, yet many ideas impact key metrics by 1% and are not well estimated a-priori. The system has also identified many negative features that we avoided deploying, despite key stakeholders' early excitement, saving us similar large amounts.

How are new experiments implemented?

Experiments in information systems are implemented using a structured approach that involves designing, executing, and analyzing studies to understand the effects of different variables on information systems. Here is a step-by-step guide on how these experiments are typically implemented:

1. Define the Research Question and Hypothesis

2. Design the Experiment

3. Choose the Experimental Design

4. Select Participants

5. Develop Materials and Procedures

6. Pilot Testing

7. Conduct the Experiment

8. Data Analysis

9. Report Findings

10. Ethical Considerations

Connections between Information Systems and Computer Science

Information systems (IS) and computer science (CS) are closely connected, with significant dependencies and influences between them. Here's an exploration of their connections:

1. **Software Development**:
    - **Computer Science**: Provides the foundational knowledge for software engineering, including algorithms, data structures, and programming languages.

    - **Information Systems**: Utilizes these software engineering principles to develop systems that meet business needs, focusing on the application and management of software to solve organizational problems.
2. **Data Management**:
    - **Computer Science**: Covers database theories, data structures, data mining, and big data technologies.
    - **Information Systems**: Applies these theories to manage and analyze organizational data, ensuring its integrity, security, and usefulness in decision-making processes.
3. **Networking and Security**:
    - **Computer Science**: Studies network protocols, cybersecurity principles, and cryptography.
    - **Information Systems**: Implements these concepts to protect organizational information, manage network infrastructures, and ensure secure communication channels.
4. **Human-Computer Interaction (HCI)**:
    - **Computer Science**: Examines user interface design, user experience, and usability testing.
    - **Information Systems**: Applies HCI principles to design user-friendly systems that enhance productivity and user satisfaction within an organization.
5. **Artificial Intelligence (AI) and Machine Learning (ML)**:
    - **Computer Science**: Develops algorithms and models for AI and ML.
    - **Information Systems**: Integrates AI and ML solutions to automate processes, predict trends, and support decision-making in business environments.
6. **Systems Analysis and Design**:
    - **Computer Science**: Focuses on methodologies and tools for system development, such as UML, Agile, and Scrum.
    - **Information Systems**: Applies these methodologies to design and implement information systems that align with business processes and requirements.

Dependencies

1. **Theoretical Foundations**: IS relies on CS for the theoretical underpinnings of information processing, algorithms, and software development.
2. **Technological Advancements**: Innovations in CS, such as new programming paradigms, AI advancements, and cybersecurity measures, directly influence the capabilities and approaches in IS.

3. **Methodologies and Tools**: Many system development methodologies, tools, and best practices in IS are derived from CS research and advancements.

Influence

1. **Practical Application**: IS provides feedback to CS by highlighting practical challenges and requirements, driving the development of more user-centric and application-oriented technologies.
2. **Business Needs**: The demands and trends in IS can influence the research agenda in CS, such as the focus on big data technologies or enterprise-level cybersecurity solutions.
3. **Interdisciplinary Research**: The collaboration between IS and CS can lead to interdisciplinary research that addresses both technical and organizational challenges, fostering innovations that benefit both fields.

Some important people

1. **Alan Turing (1912-1954): The father of theoretical computer science**

Alan Turing, one of the most famous computer scientists, was a mathematician, logician, and cryptanalyst during World War II.

Turing made significant contributions to artificial intelligence, developing the Turing Test as a method for determining a machine's ability to exhibit intelligent behavior.

As a pioneer in theoretical computer science, he designed the Turing machine, which laid the foundation for modern computers.

2. **Grace Hopper (1906-1992): A pioneer in computer programming**

Grace Hopper, an American computer scientist, was instrumental in the development of computer programming languages.

She created the first compiler, a program that translates high-level language code into machine-readable code.

Hopper was also involved in the creation of COBOL, one of the earliest high-level programming languages.

As a software engineer, her innovative work paved the way for many computer scientists who followed.

3. **John von Neumann (1903-1957): The electronic digital computer visionary**

John von Neumann, a Hungarian-American mathematician and computer scientist, played a crucial role in the development of the electronic digital computer.

His groundbreaking work on the architecture of modern computers, including the concept of stored-program computers, influenced the design of most computers today.

He also contributed to fields such as game theory, quantum mechanics, and operations research, making him one of the most famous computer scientists in history.

4. **Sir Tim Berners-Lee (b. 1955): The inventor of the World Wide Web**

What would the world be like today without “www.”? Sir Tim Berners-Lee, a British computer scientist, invented the World Wide Web, revolutionizing the way people access and share information.

He created the first web browser, web server, and the Hypertext Markup Language (HTML) that enables web pages to be displayed.

He laid out his vision for the internet-based sharing tool in 1989 while working at CERN and by 1991 others outside of CERN were invited to join the web.

Sir Tim not only invented the “www” that the internet still thrives on, but he fought to make sure the web stayed free and became accessible for anyone.

As the founder of the World Wide Web Foundation, Berners-Lee continues to advocate for an open and accessible internet for everyone.

5. **Ada Lovelace (1815-1852): The first computer programmer**

Ada Lovelace, an English mathematician, is considered the first computer programmer.

She worked alongside Charles Babbage on his Analytical Engine, a general-purpose mechanical computer.

Lovelace wrote the first algorithm intended for processing by a machine, demonstrating the potential of a computing device to perform complex calculations.

Her visionary work continues to inspire computer scientists today.

References

https://www.rasmussen.edu/degrees/technology/blog/famous-computer-scientists-who-impacted-the-industry/

'COMPUTER SCIENCE: THE DISCIPLINE' by Peter J. Denning

https://pubsonline.informs.org/doi/abs/10.1287/mnsc.23.9.913

https://en.wikipedia.org/wiki/Information_system

https://dl.acm.org/doi/abs/10.1145/2487575.2488217

Bioinformatics

ECATERINA

Bioinformatics

What is bioinformatics?

Bioinformatics is an interdisciplinary field that combines biology, computer science, statistics, and mathematics to analyse and interpret biological data. Some of the major areas of bioinformatics include:

1. Genomics: Analysis of the structure, function, and evolution of genomes.
2. Proteomics: Analysis of the structure, function, and interaction of proteins.
3. Metabolomics: Study of the small molecules present in cells, tissues, and organisms.
4. Systems biology: Integration and analysis of multiple types of data to understand biological systems as a whole.
5. Structural biology: Study of the three-dimensional structure of biological macromolecules.
6. Computational biology: Development of algorithms and computational methods to analyse and interpret biological data.
7. Drug Design: Good understanding of the disease mechanism and design of drugs based on the biological targets.

Bioinformatics software

Biological Databases:

Biological databases are archives of biological data, including genetic and protein sequences, annotations, pathways, and disease information. These databases are used for the storage and organisation of data in a way that allows easy retrieval of information.

Types of Biological Databases

Biological databases can be classified into the following three types based on their contents:

Primary Databases

Primary databases are collections of unprocessed biological data, consisting of raw sequences or structural information. These databases are repositories of original information and are not modified in any way. Examples of primary databases include GenBank, PDB, and DDBJ.

Secondary Databases

Secondary databases contain information that has been processed or curated using computational or manual methods. The information in these databases is based on the original data from primary databases. Examples of secondary databases include PIR, SWISS-PROT, and Pfam.

Specialised Databases

Specialized databases are databases that are designed to serve a specific research interest. These databases are created with a particular focus on a specific organism or type of data. Examples of specialized databases include Flybase, the HIV sequence database, and the Ribosomal Database Project.

Some of the most popular biological databases are discussed below:

- **GenBank** is a comprehensive and well-annotated collection of nucleic acid sequence data developed by the National Center for Biotechnology Information (NCBI). It contains data for nearly all types of organisms.
- **EMBL** (European Molecular Biology Laboratory) is a nucleotide sequence database managed by the European Bioinformatics Institute (EBI). It is an extensive repository of primary nucleotide sequences that stores data on DNA and RNA, gene expression, protein, structure, pathways, and literature.
- **DDBJ** (DNA Data Bank of Japan) is a nucleotide sequence database that collects and maintains nucleotide sequence data from researchers. It is operated by the National Institute of Genetics in Japan, collaborating with the National Center for Biotechnology Information (NCBI) and the European Molecular Biology Laboratory (EMBL).

- **PDB** (Protein Data Bank) is a biological database that contains structural data of biological macromolecules. PDB stores the three-dimensional structural data for large biological molecules such as proteins, DNA, and RNA, determined by experimental methods such as X-ray crystallography and NMR spectroscopy.
- **PIR** (Protein Information Resource) is a publicly accessible database of protein informatics. PIR maintains three other databases: the Protein Sequence Database (PSD), the Non-redundant Reference (NREF) database, and the integrated Protein Classification (iProClass) database.
- **PROSITE** is a protein database that contains a large collection of protein patterns or profiles. These patterns are linked to documentation providing useful biological information on the protein family, domain, or functional site.
- **Pfam:** Pfam is a database of protein families and domains represented by multiple sequence alignments, profile hidden Markov models (HMMs), and annotations. The database is accessible online and is used by researchers worldwide for various applications, including genome annotation, protein classification, and protein structure prediction.
- **KEGG** (Kyoto Encyclopedia of Genes and Genomes) is a biological database that contains genomic, chemical, and systemic functional information used to study molecular-level information about various cellular processes, including metabolism, signaling, and diseases.
- **OMIM:** Online Mendelian Inheritance in Man (OMIM) is a freely available database of human genes and genetic disorders that contains detailed and referenced overviews of all known Mendelian genetic disorders and over 16,000 genes.

Bioinformatics Tools:

Along with the construction and curation of biological databases, bioinformatics also consists of the development of computational tools for sequence, structure, and function analysis.

Bioinformatics tools are user-friendly software programs that allow researchers to analyse biological data.

Types of Bioinformatics Tools:

Bioinformatics tools can be classified into various categories based on their functionality, purpose, and complexity. Some of the widely used tools are:

Sequence Analysis Tools

These tools are used for analyzing nucleotide or protein sequences. They are also used for identifying homologous sequences and understanding the evolutionary relationships between different organisms. They include tools used for sequence alignment, sequence database searching, motif discovery, phylogeny, and genome assembly and comparison. Some popular sequence analysis tools include:

- **BLAST** (Basic Local Alignment Search Tool) is a widely used sequence similarity search tool that compares query sequences to a database of known sequences. It can identify similar sequences, infer evolutionary relationships, and identify potential functional domains within a sequence.
- **ClustalW** is a multiple sequence alignment program for DNA and protein sequences.
- **T-Coffee** is another widely used multiple sequence alignment tool that uses a combination of progressive and consistency-based alignment algorithms to produce accurate alignments. It is particularly useful for aligning distantly related sequences.
- **MEME** (Multiple EM for Motif Elicitation) is used for motif discovery and search. The MEME Suite is a software toolkit that performs four types of motif analysis: motif discovery, motif–motif database searching, motif-sequence database searching, and assignment of function.
- **MEGA** (Molecular Evolutionary Genetics Analysis) is a user-friendly software that provides many tools for phylogenetic analysis, including multiple sequence alignment, model selection, and tree inference.
- **PHYLIP** (Phylogeny Inference Package) is a collection of software applications used to determine the evolutionary relationships among species.
- **FASTA** is a DNA and protein sequence alignment software package.

Structure Analysis Tools

These tools are used for analyzing the structure of proteins and nucleic acids. Structural analysis tools include tools for nucleic acid and protein structure comparison, classification, and prediction. Some popular structure analysis tools include:

- **CN3D** is a software package used to view and analyze three-dimensional structures of macromolecules, including nucleic acids and proteins. It provides tools for visualizing and manipulating the 3D structure of macromolecules.
- **PyMOL** is a molecular visualization tool used for three-dimensional molecular structure visualization, analysis, and animation. It can display molecules in various ways, including as ball-and-stick models, cartoons, or surface representations.
- **RasMol** is a molecular graphics program designed to visualize and display proteins, nucleic acids, and small molecules in a graphical format.
- **ODELLER** is a comparative protein structure modeling tool that is used for predicting protein structures by comparing them to a known protein structure.

Function Analysis Tools

These tools are essential for understanding the functions and relationships between different genes and proteins and for identifying key pathways involved in diseases. They include tools that are used for profiling gene expression, predicting protein-protein interaction, predicting protein subcellular localization and modeling metabolic pathways.

- **GEO** (Gene Expression Omnibus) is a public repository of gene expression data that provides tools for searching, downloading, and analyzing gene expression datasets.

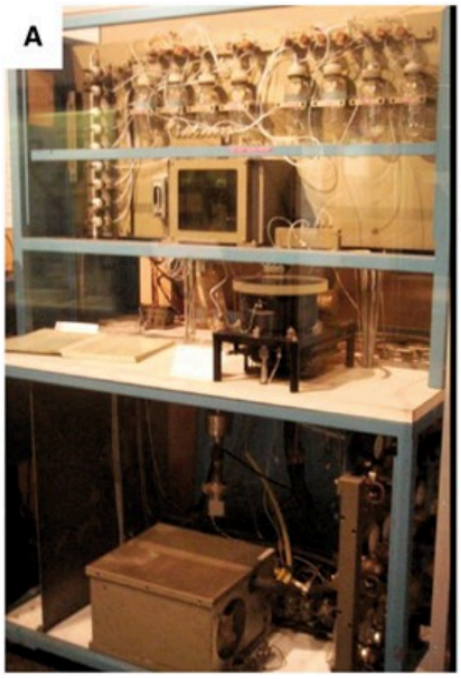
- **InterProScan** is a software package that scans protein sequences against multiple databases of protein domains and families.
- **COBRA Toolbox** is a software package for constraint-based metabolic modeling that provides a suite of tools for simulating and analyzing metabolic networks.
- **Pathway Tools** is a software package for constructing and analyzing metabolic pathway models, which includes a database of curated metabolic pathways and tools for metabolic engineering.

Notable achievements

Protein analysis was the starting point

The first crystallographic applications of stored-program computers were done on EDSAC and the Manchester Mark II in 1952–1953. However, these were used for inorganic structures. The first application of computers to protein crystallography, which some consider the real forerunner of today's bioinformatics, was in fact for the first high-resolution structure, that of myoglobin, in 1958.

In the late 1950s, the first sequence (i.e. amino acid chain arrangement) of a protein, insulin, was published. This major leap settled the debate about the polypeptide chain arrangement of proteins. The Edman degradation method emerged as a simple method that allowed protein sequencing. Coupled with automation (Figure 1 - One of the first automated peptide sequencers, designed by William J. Dreyer.), more than 15 different protein families were sequenced over the following 10 years.



A major issue with Edman sequencing was obtaining large protein sequences. Edman sequencing works through one-by-one cleavage of N-terminal amino acid residues with phenylisothiocyanate. However, the yield of this reaction is never complete. Because of this, a theoretical maximum of 50–60 amino acids can be sequenced in a single Edman reaction. Larger proteins must be cleaved into smaller fragments, which are then separated and individually sequenced.
The issue was not sequencing a protein in itself but rather assembling the whole protein sequence from hundreds of small Edman peptide sequences. For large proteins made of several hundreds (if not thousands) of residues, getting back the final sequence was cumbersome. In the early 1960s, one of the first known bioinformatics software was developed to solve this problem.

The success of Human Genome Project

In April 2003, sequencing of all three billion nucleotides in the human genome was declared complete. This landmark of modern science brought with it high hopes for the understanding and treatment of human genetic disorders. There is plenty of evidence to suggest that the hopes will become reality—1631 human genetic diseases are now associated with known DNA sequences, compared to the less than 100 that were known at the initiation of the Human Genome Project (HGP) in 1990.

Second-generation sequencing

DNA sequencing was democratised with the advent of second-generation sequencing (also called next-generation sequencing or NGS) that started with the '454' pyrosequencing technology. This technology allowed sequencing thousands to millions of DNA molecules in a single machine run, thus raising again the old computational challenge. The gold-standard tool to handle reads from 454 (the high-throughput sequencer) is still today the proprietary tool Newbler, that was maintained by Roche until the phasing out of the 454 in 2016.

The Integration of Artificial Intelligence and Machine Learning

Integrating AI and ML in bioinformatics has already led to significant breakthroughs. For example, researchers at the University of California, San Francisco, used machine learning algorithms to analyse genomic data from more than 10,000 tumours. The algorithms were able to identify patterns that could predict which tumours were likely to respond to a particular type of immunotherapy, improve patient outcomes and reduce the need for expensive and time-consuming clinical trials.

Another example of the potential of AI and ML in bioinformatics is the development of deep learning algorithms for predicting protein structures. Proteins are the building blocks of life and play a critical role in many biological processes. However, predicting the structure of a protein from its amino acid sequence is a complex and challenging task. Deep learning algorithms have shown promise in this area, with researchers at the University of Washington developing a deep learning algorithm that can predict the structure of a protein with near-atomic accuracy.

In drug discovery, AI and ML algorithms have already been used to identify new drug candidates, predict the efficacy of drugs, and reduce the risk of adverse side effects. These algorithms can analyse large data sets of drug-target interactions, drug structures, and molecular properties to identify potential drug candidates and help pharmaceutical companies to streamline the drug discovery process, reducing the time and cost of developing new drugs.

Another area where AI and ML are applied in bioinformatics is developing predictive models for disease diagnosis and prognosis. These models can analyse patient data, such as genomic data,

medical history, and clinical data, to predict the likelihood of developing a particular disease, the severity of the disease, and the likely response to treatment.

Problems in bioinformatics

Integrating artificial intelligence (AI) and machine learning (ML) into bioinformatics tools and workflows is revolutionising the field. These technologies are helping to address some of the most pressing challenges in bioinformatics:

1. **Data Integration:**
    - **Diverse Data Sources:** Bioinformatics involves the analysis of data from various sources, including genomics, transcriptomics, proteomics, metabolomics, and more. Each of these data types often comes with its own unique challenges, such as differences in data formats, scales, and levels of granularity.
    - **Interoperability:** Ensuring interoperability between different data sources and platforms is challenging. Integrating diverse datasets requires developing methods to harmonise data, accounting for variations in experimental techniques, and normalising data to a common scale.
    - **Biological Complexity:** Biological systems are inherently complex, and the integration of data across multiple levels (e.g., genes, proteins, metabolites) requires sophisticated computational models to capture the intricate relationships and dependencies within the data.

2. **Scalability:**
    - **Explosive Growth of Data:** Advances in high-throughput technologies have led to an exponential increase in biological data generation. Dealing with large-scale datasets, such as those from next-generation sequencing or high-throughput screening, poses challenges in terms of storage, processing speed, and computational resources.
    - **Algorithmic Efficiency:** As datasets grow, the efficiency of algorithms becomes crucial. Traditional algorithms may become impractical for large datasets, necessitating the development of scalable algorithms capable of handling big data efficiently.
    - **Parallel Computing:** To address scalability issues, the implementation of parallel and distributed computing approaches becomes essential. Utilising cloud computing and parallel processing architectures helps in efficiently analysing vast amounts of biological data.

3. **Standardisation:**
    - **Data Formats and Protocols:** The lack of standardised formats and protocols for data representation, storage, and sharing hinders seamless collaboration between research groups and institutions. Standardisation challenges are

particularly evident in diverse omics data, where each platform may have its own proprietary format.

- **Metadata Standards:** Metadata, crucial for understanding the context and quality of biological data, often lack standardised definitions and formats. Inconsistent metadata can complicate data integration efforts and lead to misinterpretations.
- **Analysis Pipelines:** Standardisation extends to the methods and workflows used in bioinformatics analyses. Establishing common frameworks for analysis pipelines allows for better reproducibility and comparability of results across different studies.

People and institutions that influenced the field

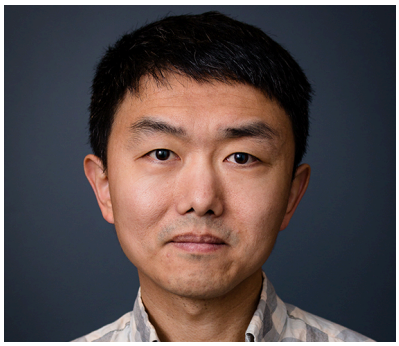
Margaret Dayhoff

Margaret Dayhoff (1925–1983) was an American physical chemist who pioneered the application of computational methods to the field of biochemistry. Dayhoff's contribution to this field is so important that David J. Lipman, former director of the National Center for Biotechnology Information (NCBI), called her 'the mother and father of bioinformatics'.

From 1958 to 1962, Margaret Dayhoff and Robert S. Ledley combined their expertise and developed COMPROTEIN, 'a complete computer program for the IBM 7090' designed to determine protein primary structure using Edman peptide sequencing data, entirely coded in FORTRAN on punch-cards.

In the COMPROTEIN software, input and output amino acid sequences were represented in three-letter abbreviations (e.g. Lys for lysine, Ser for serine). In an effort to simplify the handling of protein sequence data, Dayhoff later developed the one-letter amino acid code that is still in use today. This one-letter code was first used in Dayhoff and Eck's 1965 *Atlas of Protein Sequence and Structure*, the first ever biological sequence database.

Heng Li



Research Interests: Genomics, population genetics, phylogenetics and network dynamics.

Homepage: http://liheng.org/

Contact: me@liheng.org

Steven Salzberg

Research Interests: Computational biology, genomics, DNA sequence analysis, metagenomics, sequencing technology

Lab Website: Salzberg Lab

Publications: From Google Scholar / From Pub Med

Contact: salzberg@jhu.edu
Phone: 410-516-8246

Lior Pachter

Research Interests: Computational and experimental methods for genomics.

Homepage: https://pachterlab.github.io/biography.html

Publications: Lior Pachter - Google Scholar

Phone: 626-395-8023

Aaron Quinlan

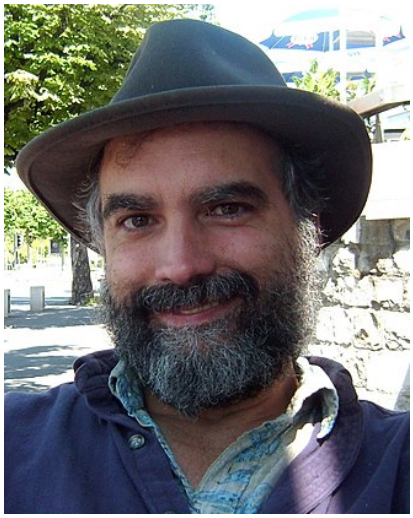
Research Interests: Genomics, Computational Biology, Bioinformatics, Human Genetics, Cancer Genomics, Rare Disease Genetics, Ovarian Cancer.

Lab Website: http://quinlanlab.org/

Publications: Aaron Quinlan | School of Medicine

Contact: aquinlan@genetics.utah.edu

William James Kent



Research Interests: Genomics, Computational Biology, pairwise sequence alignment algorithm.

Homepage: Jim Kent's Web Page

Publications: Jim Kent - Publications

Contact: kent@soe.ucsc.edu

Ewan Birney

Research Interests: Genomic biology, ENCODE project, assembly algorithms, statistical methods to analyse genomic information, and compression of sequence information.

Homepage: Ewan's Blog

Publications: Ewan Birney - Google Scholar

Benjamin Langmead

Research Interests: DNA sequencers, Computational Biology

Lab Website: Langmead Lab @ JHU

Publications: Ben Langmead - Google Scholar

Contact: langmea@cs.jhu.edu
Phone: 410-516-2033
YouTube: Ben Langmead - YouTube

Cole Trapnell

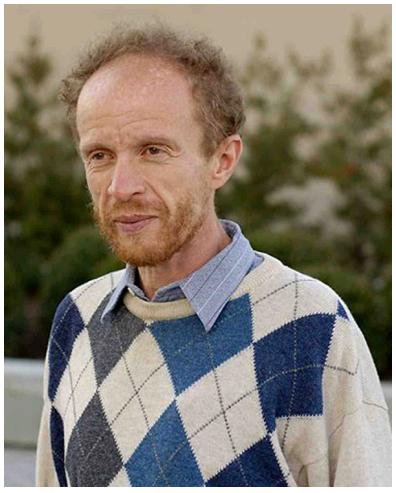
Research Interests: Gene regulation and cell-cell communication in development and disease.

Homepage: Cole Trapnell

Publications: Cole Trapnell - Google Scholar

Contact: coletrap@uw.edu

Eugene Koonin



Research Interests: Emergence and evolution of genomic and organisational complexity in biological systems, comparative analysis of sequenced genomes and automatic methods for genome-scale annotation of gene functions.

Homepage: Koonin's Group

Publications: Koonin's Group Publications</title>

Contact: koonin@ncbi.nlm.nih.gov
Phone: 301-435-5913

Ten institutions that pioneered and fostered computation in biology:

Institutions	Country
Birkbeck College, University of London	UK
Boston University	USA
European Molecular Biology Laboratory (EMBL)	DE and EMBL states
Institute of Protein Research, Academy of Sciences, Puschino	Former USSR
Laboratory of Molecular Biology (LMB), MRC Cambridge	UK
Los Alamos National Laboratory (LANL)	USA
National Biomedical Research Foundation (NBRF), Georgetown U	USA
Stanford University	USA
University of California San Francisco (UCSF)	USA
University College, University of London (UCL)	UK

Most prominent societies in bioinformatics:

Society	Description
International Society for Computational Biology (ISCB)	This is a global society dedicated to advancing the field of computational biology and bioinformatics. They offer conferences, workshops, and training programs, as well as a variety of membership benefits.
American Medical Informatics Association (AMIA)	This is a professional organization dedicated to advancing the use of informatics in healthcare. They have a Bioinformatics Translational Informatics Working Group that focuses on the use of bioinformatics to advance personalized medicine.
The Association for Computing Machinery (ACM)	This is a professional organization for computing professionals. They have a Bioinformatics and Computational Biology Special Interest Group (SIG) that provides a forum for researchers and practitioners to exchange ideas and advance the field.
The International Society for Clinical Biostatistics (ISCB)	This is a society for biostatisticians that provides opportunities for continuing education and professional development. They have a Bioinformatics Interest Group that focuses on the application of computational methods to biological data.

The Society for Industrial and Applied Mathematics (SIAM)	This is a professional society dedicated to promoting mathematics and its applications. They have a Life Sciences activity group that includes bioinformatics and computational biology.

Most important journals and publications

Publication	Comments
Zuckerkandl and Pauling, 1965b	First use of molecular sequences for evolutionary studies
Fitch and Margoliash, 1967	Use of molecular sequences to build trees
Needleman and Wunsch, 1970	First implementation of dynamic programming for protein sequence comparison
Lee and Richards, 1971	Calculation of accessibility on protein structures
Chou and Fasman, 1974	First secondary structure prediction method
Tanaka and Scheraga, 1975	Simulation of protein folding
Dayhoff, 1978	First collection of protein sequences
Hagler and Honig, 1978	One of the first explicit attempts to simulate protein folding
Doolittle, 1981	Seminal paper examining divergence and convergence in protein evolution
Felsenstein, 1981	One of the first statistical treatments of evolutionary tree construction
Richardson, 1981a	The most comprehensive description of protein structure to that date
Kabsch and Sander, 1984	Discovery with profound implications for model building by homology and structure prediction
Novotny *et al.*, 1984	The inability of distinguishing correct from incorrect structures threw back structure prediction approaches for a long while
Chothia and Lesk, 1986	Examination of divergence between sequence and structure
Doolittle, 1986	Influential book on sequence analysis
Feng and Doolittle, 1987	The first approach for an efficient multiple sequence alignment procedure, later implemented in CLUSTAL
Lathrop *et al.*, 1987	One of the first applications of Artificial Intelligence in protein structure analysis and prediction
Ponder and Richards, 1987	The very first threading approach, using sequence enumeration
Altschul *et al.*, 1990	The implementation of a sequence matching algorithm based on Karlin's statistical work
Bowie *et al.*, 1991	The first implementation of protein structure prediction using threading

Journals:

No.	Title	h5-index
1.	Bioinformatics	148
2.	Briefings in Bioinformatics	102
3.	PLOS Computational Biology	92
4.	BMC Bioinformatics	80
5.	GigaScience	70
6.	IEEE/ACM Transactions on Computational Biology and Bioinformatics	56

7.	Journal of Cheminformatics	53
8.	Journal of Theoretical Biology	53
9.	Genomics, Proteomics & Bioinformatics	49
10.	Database	43
11.	Mathematical Biosciences and Engineering	43
12.	Computational Biology and Chemistry	39
13.	Journal of Mathematical Biology	35
14.	Mathematical Biosciences	34
15.	IEEE International Conference on Bioinformatics and Biomedicine	33
16.	BMC Systems Biology	33
17.	Current Opinion in Systems Biology	30
18.	Current Bioinformatics	29
19.	NAR Genomics and Bioinformatics	29
20.	Molecular Informatics	29

Connection to other areas

Bioinformatics is a discipline that is dependent on other areas of both Biology and Computer Science. It is heavily influenced by the developments from the following Computer Science areas:

- Programming languages: As new programming languages emerge, the software for bioinformatics is following the trend. From assembly language, to Python, Perl and R, the developers of bioinformatics tools are constantly looking towards new developments in the programming languages field.

- Algorithms and Data Structures: In order to develop algorithms to analyse biological data, it is essential to use data structures such as graphs, arrays, hash tables, etc.

- Computer architecture: Computer architecture is responsible for the fast computational power that we have today. Bioinformatics relies on the advances in this field to be able to analyse more and more data in a shorter time.

- Databases and information retrieval: Databases are crucial for organising biological data, the main resources of bioinformatics software.

- Artificial intelligence and robotics: Artificial Intelligence and Machine Learning have had a great contribution to Bioinformatics, helping scientists to shape predictive models for disease diagnosis and predicting the structure of a protein from its amino acid sequence.

- Graphics: Visualising bioinformatics data allows researchers to gain insight into their data that is not obvious from simple descriptive statistics. Graphic representations are used in a variety of bioinformatics tools, such as CN3D, PyMOL and RasMol.

Bioinformatics at UVT

The West University of Timisoara offers a Master Degree in Bioinformatics:
Bioinformatics - Faculty of Mathematics and Computer Science

The future of bioinformatics

The late 20th century witnessed the emergence of computers in biology. Their use, along with continuously improving laboratory technology, has permitted increasingly complex research endeavors. Whereas sequencing a single protein or gene could have been the subject of a doctoral thesis up to the early 1990s, a PhD student may now analyze the collective genome of many microbial communities during his/her graduate studies. Whereas determining the primary structure of a protein was complex back then, one can now identify the whole proteome of a sample. Biology has now embraced a holistic approach, but within distinct macromolecular classes (e.g. genomics, proteomics and glycomics) with little crosstalk between each subdiscipline.

One may anticipate the next leap: instead of independently investigating whole genomes, whole transcriptomes or whole metabolomes, whole living organisms and their environments will be computationally modeled, with all molecular categories taken into account simultaneously. In fact, this feat has already been achieved in a whole cell model of Mycoplasma genitalium, in which all its genes, their products and their known metabolic interactions have been reconstructed in silico. Perhaps we will soon witness an in silico model of a whole pluricellular organism. Even though this might seem unfeasible to model millions to trillions of cells, one must keep in mind that we are now achieving research exploits that would have been deemed as computationally or technically impossible even 10 years ago.

I want a career in bioinformatics!

The role of a bioinformatics scientist is to develop and apply computational tools and algorithms to analyze, manage, and interpret large biological datasets.

The International Society for Computational Biology published guidelines and recommendations of core competencies that a bioinformatician should have in her/his curriculum, based on three user categories (bioinformatics user, bioinformatics scientist and bioinformatics engineer). All three user categories contain core competencies such as: *'[using] current techniques, skills, and tools necessary for computational biology practice', '[applying] statistical research methods in the contexts of molecular biology, genomics, medical, and population genetics research' and 'knowledge of general biology, in-depth knowledge of at least one area of biology, and understanding of biological data generation technologies'*.

Additional competencies were defined for the remaining two categories, such as: *'[analysing] a problem and identify and define the computing requirements appropriate to its solution' for the bioinformatics scientist, and '[applying] mathematical foundations, algorithmic principles, and computer science theory in the modelling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices' for the bioinformatics engineer*'.

The general path to becoming a bioinformatician seems to look like this:

- Obtain a Bachelor's Degree: To become a bioinformatics scientist, you should begin by obtaining a Bachelor's Degree in Biology, Computational Biology, Bioinformatics, or a related field, such as biochemistry, genetics, molecular biology or computer science.
- Acquire Programming Skills: Programming skills are essential for bioinformatics scientists. You should learn programming languages such as Python, Perl, and R, which are commonly used in bioinformatics. Bioinformatics involves working with large datasets, and it is important to know how to manage data using databases, such as SQL.
- Obtain a Graduate Degree: To become a bioinformatics scientist, you will need to obtain a Master's or Ph.D. Degree in Bioinformatics, Computational Biology, or a related field.
- Gain Practical Experience: You should look for opportunities to work in research laboratories, internships, or participate in projects that use bioinformatics tools and techniques.
- Network: Networking with other bioinformatics professionals can help you to learn about job opportunities, stay up-to-date with the latest research, and collaborate on projects.

References:

7. Christos A. Ouzounis and Alfonso Valencia (2003) *Early bioinformatics: the birth of a discipline — a personal view*.
8. Jeff Gauthier, Antony T Vincent, Steve J Charette, Nicolas Derome (2019), *A brief history of bioinformatics*.
9. Sanju Tamang (2023) *Bioinformatics Databases, Software, and Tools with Uses*.
10. Martti T Tammi (2023) *History and major milestones in bioinformatics*.
11. https://omicstutorials.com/current-challenges-inclusive-solutions-and-future-horizons-in-bioinformatics/
12. Ivan Merelli, Horacio Pérez-Sánchez, Sandra Gesing, and Daniele D'Agostino (2014) *Managing, Analysing, and Integrating Big Data in Medical Bioinformatics: Open Problems and Future Perspectives*
13. Agustin Calatroni, Jeremy J. Wildfire (2017) *Graphic depiction of bioinformatics data*