Great Ideas in Computing

University of Toronto CSC196
Fall 2023

Week 1: September 11-September 15 (2023)

Course Organization

Course Instructor: Allan Borodin

- Email: bor@cs.toronto.edu

Teaching Assistant : Aniket Kali

I strongly encourage questions and discussions in class. In addition, we are also using piazza for questions and discussions.

Using piazza, you can also answer questions posed by others. You can pose or answer questions anonymously or using your name. The benefit of in class and piazza (over email) is that the entire class benefits from the discussion. Please sign up at "piazza.com uoft" if you wish to be part of the piazza discussions.

Course organization continued

The class meets M,W,F at 10 AM in MY 317

Usually I will be presenting on Mondays and Wednesdays and the tutorials will be on Fridays. You should not have a conflict with any of these three 50 minute classes.

There might be be some exceptions to the usual schedule if we have to accomodate the schedule of a guest presenter. Also for the week of September 25, the tutorial will be on Monday, September 25 with the class on Wednesday, the 27th and Friday the 29th.

Course web site: http://www.cs.toronto.edu/~bor/196f23
My slides will be on the web page. I will try to post slides "soon" after each class. Why not before? I will also post various documents on the web page.
There are also links to the previous versions of CSC196 (fall 2021, fall 2022, fall 2023) and links to versions of SCI199 on the web page.

I will mainly be using Quercus for Announcements.

COVID considerations

Lets hope we do not have to worry about COVID.

Unfortunately, COVID is still around but currently not a major health problem. Everyone has their own sense of how much to still be concerned about COVID. At moment, I am not that worried about the current status of COVID. But things can change so keep aware of any changes in Universoty policy. Please speak to me if you have any concerns.

While the mask mandate was been suspended as of 1 July 2022, the use of medical masks continues to be encouraged in high density indoor settings where physical distancing is not possible. We ask everyone to respect each other's decisions, comfort levels, and health needs.

Announcement: RSGs

Recognized Study Groups (RSGs) are voluntary, peer-led study groups of up to eight students enrolled in the same Arts and Science course. Last year, over 3,000 students participated in RSGs for courses spanning all disciplines and class sizes.

[RSG Poster]

RSGs are led by students enrolled in the course who receive training on group study strategies and academic integrity.

Preliminaries

Getting to know each other

- A little about myself
- Your plans at the University?

What is this course about?

- FAS Calendar about First Year Foundational Seminars
  https://www.artsci.utoronto.ca/future/academic-opportunities/first-year-opportunities/first-year-foundations-seminars
  One might say that all of these first year courses are an "Introduction to Critical Thinking".
- Seminars enable students to engage in discussions and develop strong written, oral and teamwork skills. Small classes help students build relationships with professors early in their academic career.

Brief theme of our CSC196 course: Great Ideas in Computing

What constitutes a "great idea"?

Brief theme of our CSC196 course: Great Ideas in Computing

What constitutes a "great idea"?

Impact, surprise, elegance. Of course, it is easier to agree on great ideas in retrospect rather than as ideas are being introduced.

And of course ideas rarely occur in a "vacuum"; usually there are similar ideas known and often the timing of when an idea becomes viable is very critical.

It is also the case that credit for an idea is not always completely fair to all those involved.

More thoughts on what constitutes a great idea

- "It is a breakthrough"; meaning it accomplishes something that was not possible before.
- "It allows for new (possibly unexpected) possibilities"
- "It provides an *optimal* solution to a problem"
- "I wish I could have thought of this"
- "It is the first thing I would want to describe to someone not in CS"'

Great ideas, good decisions, good plans

- "At the time, it was not considered a good idea" Can we call an idea a great idea if it never gets adopted?

Great ideas, good decisions, good plans

- "At the time, it was not considered a good idea" Can we call an idea a great idea if it never gets adopted?
- How should we characterize a good decision or a good plan?

Great ideas, good decisions, good plans

- "At the time, it was not considered a good idea" Can we call an idea a great idea if it never gets adopted?
- How should we characterize a good decision or a good plan?
  A point of view: perhaps one should not judge the quality of a decision by outcomes. The quality of a good decision may simply be whether or not, given all the information available at the time the decision was made, the decision was a good or the best decision one could make.
- Decisions can be irrevocable or can be modified. Plans usually do evolve over time.

Great ideas, good decisions, good plans

- "At the time, it was not considered a good idea" Can we call an idea a great idea if it never gets adopted?
- How should we characterize a good decision or a good plan?
  A point of view: perhaps one should not judge the quality of a decision by outcomes. The quality of a good decision may simply be whether or not, given all the information available at the time the decision was made, the decision was a good or the best decision one could make.
- Decisions can be irrevocable or can be modified. Plans usually do evolve over time.
- It is important to understand that in all ideas, decisions, plans, there is always some degree of **uncertainty** as to how the "world" will unfold.
- In research and development (and more generally in life) how long should we stick with our intuition or when do we see that our beliefs and plans are inconsistent with facts or "common wisdom"?

Great ideas may have negative consequences

There can be unintended, undesireable consequences for ideas we may come to accept as great ideas? Is it then still a great idea?

Great ideas may have negative consequences

There can be unintended, undesireable consequences for ideas we may come to accept as great ideas? Is it then still a great idea?

- If a technology becomes a standard, it can also become a barrier to innovation. For example, we will soon encounter our first great idea "The von Neumann" model. Some have argued that it has impeded progress on alternative computational architectures.
- Social networks allow for maintaining friendships. They also make possible the rapid spread of information and **mis-information**. It is claimed that social media companies deliberately fail to act on hate speech and political extremism while social media companies argue they are actively removing hate speech and violent discourse above and beyond reported cases.
- Moreover, some claim that these companies do not want to eliminate hate speech and that they often target information which in turn reinforces divisions in society. (See the links to article by Starbird et al, Axios article on monitoring of AI with regard to hate speech, and a text file with some links to the issue of hate speech and bias on social media platforms.)

Great ideas can have negative consequence continued

What are the ultimate dangers of AI?

I saw the following comment in one of the news feeds I look at:

Dozens of industry leaders and academics in the field of artificial intelligence have called for greater global attention to the possible threat of "extinction from AI." A statement, signed by leading industry officials like OpenAI CEO Sam Altman and Geoffrey Hinton — the so-called "godfather" of artificial intelligence — highlights wide-ranging concerns about the ultimate danger of unchecked AI. Experts say humanity is still a ways off from the prospect of science-fiction-like AI overlords, but the flood of hype and investment into the AI industry has led to calls for regulation now before any major mishaps occur. The growing AI arms race has already generated more immediate concerns. Lawmakers, advocacy groups and tech insiders have raised alarms about the potential for AI-powered language models like ChatGPT to spread

Predictions and the nature of research

Niels Bohr, Mark Twain, Yogi Berra: "Predicting the future is hard because it hasn't happened yet" and "it's tough to make predictions, especially about the future" Who knows who said it first?

- Predictions about computing and what and can and can't be done within some predicted time frame are often wrong. I posted a link to the 1955 Dartmouth summer project on artificial intelligence AI (where the term seems to have first appeared). Turing's 1950 article provided what is now called the Turing Test as to "whether or not it is possible for a machine to show intelligent behaviour". The Dartmouth project suggests that the indicated challenges for AI could be done over the summer.
- I posted a link to an article by John Backus about the view (that he proved wrong) that source level languages could never be nearly as efficient as machine code.
- There is also an article giving what one individual calls the "7 Worst Tech Predictions of All Time." But note that the quote attributed to T.J. Watson may never have happened.

Grading scheme, syllabus and possible topics

- The grading scheme will be based on 4 assignments (15% each), two quizzes (10% each), and class participation (20%). Students are expected to attend all classes regularly and participate actively. There will be no final exam.
- Assignments will be submitted on Markus. Login at https://markus.teach.cs.toronto.edu/2023-09 and select CSC196. Once you login, select the course CSC196
  Tentative dates appear on the next slide and I also posted tentative dates in the syllabus.
- It is difficult but not impossible to fail this course. Mainly do NOT plagarize. If you have any questons about plagarism, ask me.
- The syllabus (listed on the course web page) contains other organizational information.
- There is also an ambitious list of possible topics on the web page.
  What have we missed?
  When I ask a question in red, thats a strong invitation for YOU to join the conversation but don't wait for an invitation to speak up.

Relevant dates

- A0 September 19 11PM. This is worth 2% as part of the participation grade
- A1 October 6 9AM
- Q1 October 20
- A2 October 27 9AM
- *Note:* November 6 is the last day to drop a Fall (F) course. But if you think you are going to withdraw please do so as soon as possible so as to make room for someone on the wait list.
- Reading week is November 6-10.
- A3 November 18 9AM
- Q2 November 24
- A4 December 1 9AM
- Fall classes end December 6. We are allowed to schedule a possible makeup date on December 7 to compensate for the missed class on Thanksgiving.

Possible Topics

- What is a great idea?
- What responsibilties do computer professionals have for the impact (and possible misuse) of the technology?
- What is a computer? The von Neumann architecture. Digital vs analogue. What were the alternatives? What else is possible (parallel, quantum)?
- The genius of Alan Turing; A mathematical definition of computable function. Interpreters. Non computable functions.
- How did computers and computing become a commodity? The amazing advances in software and algorithms came along with advances in hardware (cost, speed, memory size, physical size, power) and communication (cost, capacity and speed) . Demand for "killer applications" such as word processors, email, search engines, navigation systems, games).
- The internet; packet routing. TCP/IP.

More possible topics

- Fortran, the first commericial source level language (with an efficient compiler). First compiler often attributed to Grace Hopper. John Backus vs the prevailing view that compiled code would be too slow compared to machine code.
- How search engines work and what they do well and what (if anythng) they dont do well.
- The semantic web.
- A local great idea: NP completeness. What is and what is not *efficiently* computable.
- Complexity based cryptography; public key cryptography; digital signatures. Captchas.
- Another local great idea: deep neural networks and the success of machine learning (ML). Large language models. What (if anything) is the limitation of ML?
- HCI (human computer interaction). The mouse. Menus, click, paste and drag. Visualization.

And some more possible topics

- Information theory: the genius of Claude Shannon. Error correcting codes; compression.
- Social networks and the spread of information (and mis-information, conspiracies, etc). Targetting information to different communities. (What is a social network community?)
- Open Source. Wikipedia. Blogs
- Relational data bases.
- Linear programming; dynamic programming and combinatorial optimization. How far can one go with conceptually simple algorithms? Dijkstra's algorithm and navigation systems.
- Distributed System primitives : mutual exclusion, consensus
- Differential privacy; extracting useful statistical information without sacrificing individual information.
- Algorithmic mechanism design; automated auctions. Algorithmic social choice; (e.g., voting, fair allocation).

Great ideas are not isolated

To appreicate what we (possibly) cannot do efficiently, we should know what we can do efficiently by known methods such as linear programming.

Many computational problems now have improved solutions using machine learning techniques (ML). For example, search engines now use ML to improve the responses that you (personally) receive to a query.

Of course we will only be able to discuss a small subset of these ideas and yet

Great ideas are not isolated

To appreicate what we (possibly) cannot do efficiently, we should know what we can do efficiently by known methods such as linear programming.

Many computational problems now have improved solutions using machine learning techniques (ML). For example, search engines now use ML to improve the responses that you (personally) receive to a query.

Of course we will only be able to discuss a small subset of these ideas and yet

What great ideas have we might mised?

Great ideas are not isolated

To appreicate what we (possibly) cannot do efficiently, we should know what we can do efficiently by known methods such as linear programming.

Many computational problems now have improved solutions using machine learning techniques (ML). For example, search engines now use ML to improve the responses that you (personally) receive to a query.

Of course we will only be able to discuss a small subset of these ideas and yet

What great ideas have we might mised?

In Assignment A0, I asked you to rank your top 3 choices for a topic to be discussed. In addtion to any of the topics mentioned, feel free to add any topic not mentioned as part of your top three.

Given that different individuals have different preferences, how do we form a consensus? Forming consensus and making "fair" decisions are central topics in computational social choice.

End of Monday, September 11 class

We ended the class by mentioning some possible topics.

At the end of the class, one student suggested the following topic: What are the current *fundamental challenges* in computing/computer science?

I forgot to mention that I encourage students to sign up for piazza. See slide 2.

What would be good times for office hours?

Slides and guest speakers

NOTE: My slides will only be an outline of our discussions. **And we will often "wander" (i.e., take tangents) as we discuss ideas.** These wanderings will most likely not be in the slides.

I have lined up some guest speakers to lead discussions on some recent "great ideas". Here is the list of guest presentations.

- Colin Raffel (ML, Large Languagve Models) October 2
- Kyros Kualakos (computational vision) November 20
- Akshay Srinivasan (cryptography) TBA

Our first great idea

Our first topic/great idea for discussion: the von Neumann architecture

- By the mid 40s the first computers were being built. (I am not going to talk say about Babbage and the recent implementation of Babbages 1850 machine.)
- The earliest computers had fixed programs and were not general purpose machines.
- The stored program computer. The conceptual idea is associated with von Neumann who first described the model in a paper "First Draft of a Report on the EDVAC" dated June 30, 1945.
- It is clear that the stored program idea is present in Turing's 1936 paper which we will discuss later. But the Turing model was not meant to be a model of an actual computer.

The von Neumann architecture and digitilization

- The basic organization consists of 4 units: memory, I/O, ALU, control. (Some would say that a "bus" to cary signals between units is a fifth component but most say 4 units.) The memory is organized into a list or array of "words", each with its own address.
- Each word w_i (with say address i) is some fixed length string of bits bits ; i.e., $w = b_{n-1}b_{n-2}, \ldots, b_0$ which as an integer represents $\sum_{j=0}^{n-1} b_j 2^j$. In order to represent negative numbers, we could reserve the high order bit as a sign bit. Thus $b_{n-1}b_{n-2}\ldots b_0$ could represent $(-1)^{b_{n-1}} \sum_{j=0}^{n-2} b_j 2^j$. That is, $b_{n-1} = 1$ specifies a negative number. (The choice of 2 is not essential mathematically.). It is also possible to think of a word as a string of "bytes" in which each byte can also be addressable but we will ignore that. Why not unary, decimal, or some other representation?

The von Neumann architecture and digitilization

- The basic organization consists of 4 units: memory, I/O, ALU, control. (Some would say that a "bus" to cary signals between units is a fifth component but most say 4 units.) The memory is organized into a list or array of "words", each with its own address.
- Each word w_i (with say address i) is some fixed length string of bits bits ; i.e., $w = b_{n-1}b_{n-2}, \ldots, b_0$ which as an integer represents $\sum_{j=0}^{n-1} b_j 2^j$. In order to represent negative numbers, we could reserve the high order bit as a sign bit. Thus $b_{n-1}b_{n-2}\ldots b_0$ could represent $(-1)^{b_{n-1}} \sum_{j=0}^{n-2} b_j 2^j$. That is, $b_{n-1} = 1$ specifies a negative number. (The choice of 2 is not essential mathematically.). It is also possible to think of a word as a string of "bytes" in which each byte can also be addressable but we will ignore that. Why not unary, decimal, or some other representation?
- But before we go any further we need to step back and talk about *digitalization and encoding*. That is, what are we storing in these words?

The von Neumann architecture and digitilization

- The basic organization consists of 4 units: memory, I/O, ALU, control. (Some would say that a "bus" to cary signals between units is a fifth component but most say 4 units.) The memory is organized into a list or array of "words", each with its own address.
- Each word w_i (with say address i) is some fixed length string of bits bits ; i.e., $w = b_{n-1}b_{n-2}, \ldots, b_0$ which as an integer represents $\sum_{j=0}^{n-1} b_j 2^j$. In order to represent negative numbers, we could reserve the high order bit as a sign bit. Thus $b_{n-1}b_{n-2}\ldots b_0$ could represent $(-1)^{b_{n-1}} \sum_{j=0}^{n-2} b_j 2^j$. That is, $b_{n-1} = 1$ specifies a negative number. (The choice of 2 is not essential mathematically.). It is also possible to think of a word as a string of "bytes" in which each byte can also be addressable but we will ignore that. Why not unary, decimal, or some other representation?
- But before we go any further we need to step back and talk about *digitalization and encoding*. That is, what are we storing in these words? Data and instructions (as part of a program)

Digital computing vs the continuous real world

- The concept of digitalization is one of the most profound concepts that underlies science today.

Digital computing vs the continuous real world

- The concept of digitalization is one of the most profound concepts that underlies science today.
- In the "real world", space and time are continuous A typical computing application might wish to study the trajectory or movement of an object (person, ball, missle, etc). An object is moving through continuous 3 dimensional space in continuous time according to some (hopefully) known laws of motion.
- In contrast, digital computers use discrete finitely represented data and instructions.

Digital computing vs the continuous real world

- The concept of digitalization is one of the most profound concepts that underlies science today.
- In the "real world", space and time are continuous A typical computing application might wish to study the trajectory or movement of an object (person, ball, missle, etc). An object is moving through continuous 3 dimensional space in continuous time according to some (hopefully) known laws of motion.
- In contrast, digital computers use discrete finitely represented data and instructions.

  However, a real value x can be approximated by a rational number. Every integer has a finite representation and every rational can be represented by a pair of integers and hence and hence has a finite reperesentation.
- We will see how we can *approximate* any real number (within some reasonably large range) using *floating point representation*.

Digital computing vs the real world continued

- Furthermore there are some stochastic (i.e., probabilistic) aspects (such as wind and air polution in the trajectory example) in real world processes.
- In contrast, digital computers are deterministic; that is, each "state" of the computer is precisely determined from the previous state.

Digital computing vs the real world continued

- Furthermore there are some stochastic (i.e., probabilistic) aspects (such as wind and air polution in the trajectory example) in real world processes.
- In contrast, digital computers are deterministic; that is, each "state" of the computer is precisely determined from the previous state.

  However, we can simulate stochastic events by drawing random variables from some *(pseudo) random* distribution.
- Stochastic processes can then be simulated by a discrete deterministic machine. Of course, the conclusions to be made from simulations relative to a specific outcome in the real world will depend on how well we have modelled the system and how unpredictable is the system.
- The early debate: relative benefits of digital computation vs that of analog computers. (In the past, thermostats and other control devices were essentially simple analog computers.) Why did digitlilization win the debate?

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is "addressable". We conceptually think of each word being accessed (read or written) at some unit (of time) cost.
Is this how your lap top memory is organized?

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is "addressable". We conceptually think of each word being accessed (read or written) at some unit (of time) cost.
Is this how your lap top memory is organized?

Take a look at the specifications of your lap top and you will almost surely see different types of memory. This is called the memory hierarchy. Lets just consider a two level hierarchy, where the fast memory might be called the cache and a slower memory which we can just call the "main memory". (There can be more than two levels in the memory hierarchy including external memory which still might be addressable.)

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is "addressable". We conceptually think of each word being accessed (read or written) at some unit (of time) cost.
Is this how your lap top memory is organized?

Take a look at the specifications of your lap top and you will almost surely see different types of memory. This is called the memory hierarchy. Lets just consider a two level hierarchy, where the fast memory might be called the cache and a slower memory which we can just call the "main memory". (There can be more than two levels in the memory hierarchy including external memory which still might be addressable.)

The cache is a more costly relatively small fast memory and main memory is considerably larger and slower memory in terms of accessing time. So, in fact, different accesses can take different amounts of time. However ....

The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting "locality of reference" so that it is likey that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a page) at the same time.

The following Least Recently Used (LRU) algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting "locality of reference" so that it is likey that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a page) at the same time.

The following Least Recently Used (LRU) algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

Why not just have fast memory?

The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting "locality of reference" so that it is likey that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a page) at the same time.

The following Least Recently Used (LRU) algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

Why not just have fast memory?

In computing (and life) there are almost inevitably tradeoffs to be made.

The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting "locality of reference" so that it is likey that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a page) at the same time.

The following Least Recently Used (LRU) algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

Why not just have fast memory?

In computing (and life) there are almost inevitably tradeoffs to be made.

The important thing is that you (the algorithm designer) usually do not need to know how this memory hierarchy is managed; that is, when data is brought into the cache and when it is ejected from the cache. This is referred to as caching or sometimes as paging.

Memory management also make it possible to have several processes sharing the main memory.

A little more about memory management

Another aspect of memory management is allocating memory to arrays and other data structures. For example, when we declare (in a programming language) that A is an array of some fixed size, for efficiency we want to allocate consecutive blocks of memory words. (We will soon discuss the idea of data types and data structures.)

A False Economy

An algorithm designer may think they know their application and can better optimize storage management for their application.

Why is this a false economy?

This is similar to the false economy of an algorithm that wants to write its own machine code.

Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.
Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!
It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.
Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!
It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

Why is the von Neumann model such a great idea?

Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.
Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!
It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

Why is the von Neumann model such a great idea?
Are we stuck in a "von Neumann tarpit?"

The benefits of a well agreed upon abstract model of computation

One of the main reasons to consider the von Neumann model a great idea is that by being a well agreed upon model, coordination amongst different people is minimized. That is,

- A computer architect doesn't need to know which programming languages will be run on their specific architecture. The abstract von Neumann model doesn't specify the instruction set, the memory management, how interupts are handled, etc. This is what the architects do.
- A compiler writer for a programming language $\mathcal{L}$ doesn't have to know what algorithms will be implemented using the language $\mathcal{L}$.
- Without complete knowledge of the architecture, and the compiler, the algorithm designer can make a rough approximation for the memory and time requirements of their algorithm.

Progress in parallel computation had been relatively slow but there is now some common approaches (e.g., MapReduce for large scale parallel computation).

Dataflow architecture

Direct from Wikipedia:

Dataflow architecture is a computer architecture that directly contrasts the traditional von Neumann architecture or control flow architecture. Dataflow architectures do not have a program counter (in concept): the executability and execution of instructions is solely determined based on the availability of input arguments to the instructions,[1] so that the order of instruction execution is unpredictable, i.e. behavior is *nondeterministic*. (My emphasis)
Although no commercially successful general-purpose computer hardware has used a dataflow architecture, it has been successfully implemented in specialized hardware such as in digital signal processing, network routing, graphics processing, telemetry, and more recently in data warehousing.[citation needed] It is also very relevant in many software architectures today including database engine designs and parallel computing frameworks.[citation needed]

More on data flow architecture

From J. Paul Morrison's Flow-Based Programming text

The von Neumann machine is perfectly adapted to the kind of mathematical or algorithmic needs for which it was developed: tide tables, ballistics calculations, etc., but business applications are rather different in nature. . . .
Business programming works with data and concentrates on how this data is transformed, combined, and separated. . . . Broadly speaking, whereas the conventional approaches to programming (referred to as "control flow") start with process and view data as secondary, business applications are usually designed starting with data and viewing processes as secondary—processes are just the way data is created, manipulated, and destroyed. We often call this approach "data flow." (21)