Computer programming

From Wikipedia, the free encyclopedia

Jump to: navigation, search

Programming" redirects here. For other uses, see Programming (disambiguation).

Software development process

Activities and steps
Requirements · Specification Architecture · Design **Implementation** · Testing Deployment · Maintenance
Models
Agile · Cleanroom · DSDM Iterative · RAD · RUP · Spiral Waterfall · XP · Scrum · Lean V-Model · FDD
Supporting disciplines
Configuration management Documentation Quality assurance (SQA) Project management User experience design
Tools
Compiler · Debugger · Profiler GUI designer Integrated development environment

Computer programming (often shortened to **programming** or **coding**) is the process of writing, testing, debugging/troubleshooting, and maintaining the source code of computer programs. This source code is written in a programming language. The code may be a modification of an existing source or something completely new. The purpose of programming is to create a program that exhibits a certain desired behaviour (customization). The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic.

Contents

Overview

Within software engineering, programming (the *implementation*) is regarded as one phase in a software development process.

There is an ongoing debate on the extent to which the writing of programs is an art, a craft or an engineering discipline.[1] Good programming is generally considered to be the measured application of all three, with the goal of producing an efficient and evolvable software solution (the criteria for "efficient" and "evolvable" vary considerably). The discipline differs from many other technical professions in that programmers generally do not need to be licensed or pass any standardized (or governmentally regulated) certification tests in order to call themselves "programmers" or even "software engineers." However, representing oneself as a "Professional Software Engineer" without a license from an accredited institution is illegal in many parts of the world.[*citation needed*]

Another ongoing debate is the extent to which the programming language used in writing computer programs affects the form that the final program takes. This debate is analogous to that surrounding the Sapir-Whorf hypothesis [2] in linguistics, that postulates that a particular language's nature influences the habitual thought of its speakers. Different language patterns yield different patterns of thought. This idea challenges the possibility of representing the world perfectly with language, because it acknowledges that the mechanisms of any language condition the thoughts of its speaker community.

Said another way, programming is the craft of transforming requirements into something that a computer can execute.

History of programming

See also: History of programming languages

Wired plug board for an IBM 402 Accounting Machine.

The concept of devices that operate following a pre-defined set of instructions traces back to Greek Mythology, notably Hephaestus and his mechanical servants[3]. The Antikythera mechanism was a calculator utilizing gears of various sizes and configuration to determine its operation. The earliest known programmable machines (machines whose behavior can be controlled and predicted with a set of instructions) were a Muslim Scientist Al-Jazari's programmable Automata in 1206.[4] One of Al-Jazari's robots was originally a boat with four automatic musicians that floated on a lake to entertain guests at royal drinking parties. Programming this mechanism's behavior meant placing pegs and cams into a wooden drum at specific locations. These would then bump into little levers

that operate a percussion instrument. The output of this device was a small drummer playing various rhythms and drum patterns.[5][6] Another sophisticated programmable machine by Al-Jazari was the castle clock, notable for its concept of variables which the operator could manipulate as necessary (i.e. the length of day and night). The Jacquard Loom, which Joseph Marie Jacquard developed in 1801, uses a series of pasteboard cards with holes punched in them. The hole pattern represented the pattern that the loom had to follow in weaving cloth. The loom could produce entirely different weaves using different sets of cards. Charles Babbage adopted the use of punched cards around 1830 to control his Analytical Engine. The synthesis of numerical calculation, predetermined operation and output, along with a way to organize and input instructions in a manner relatively easy for humans to conceive and produce, led to the modern development of computer programming. Development of computer programming accelerated through the Industrial Revolution.

In the late 1880s Herman Hollerith invented the recording of data on a medium that could then be read by a machine. Prior uses of machine readable media, above, had been for control, not data. "After some initial trials with paper tape, he settled on punched cards..."[7] To process these punched cards, first known as "Hollerith cards" he invented the tabulator, and the key punch machines. These three inventions were the foundation of the modern information processing industry. In 1896 he founded the *Tabulating Machine Company* (which later became the core of IBM). The addition of a control panel to his 1906 Type I Tabulator allowed it to do different jobs without having to be physically rebuilt. By the late 1940s there were a variety of plug-board programmable machines, called unit record equipment, to perform data processing tasks (card reading). Early computer programmers used plug-boards for the variety of complex calculations requested of the newly invented machines.

Data and instructions could be stored on external punch cards, which were kept in order and arranged in program decks.

The invention of the Von Neumann architecture allowed computer programs to be stored in computer memory. Early programs had to be painstakingly crafted using the instructions of the particular machine, often in binary notation. Every model of computer would be likely to need different instructions to do the same task. Later assembly languages were developed that let the programmer specify each instruction in a text format, entering abbreviations for each operation code instead of a number and specifying addresses in symbolic form (e.g. ADD X, TOTAL). In 1954 Fortran was invented, being the first high level programming language to have a functional implementation.[8][9] It allowed programmers to specify calculations by entering a formula directly (e.g. Y = X*2 + 5*X + 9). The program text, or *source*, is converted into machine instructions using a special program called a compiler. Many other languages were developed, including some for commercial programming, such as COBOL. Programs were mostly still entered using punch cards or paper tape. (See computer programming in the punch card era). By the late 1960s, data storage devices and computer terminals became inexpensive enough so programs could be created by typing directly into the computers. Text editors were developed that allowed changes and

corrections to be made much more easily than with punch cards.

As time has progressed, computers have made giant leaps in the area of processing power. This has brought about newer programming languages that are more abstracted from the underlying hardware. Although these high-level languages usually incur greater overhead, the increase in speed of modern computers has made the use of these languages much more practical than in the past. These increasingly abstracted languages typically are easier to learn and allow the programmer to develop applications much more efficiently and with less code. However, high-level languages are still impractical for many programs, such as those where low-level hardware control is necessary or where processing speed is at a premium.

Throughout the second half of the twentieth century, programming was an attractive career in most developed countries. Some forms of programming have been increasingly subject to offshore outsourcing (importing software and services from other countries, usually at a lower wage), making programming career decisions in developed countries more complicated, while increasing economic opportunities in less developed areas. It is unclear how far this trend will continue and how deeply it will impact programmer wages and opportunities.

Modern programming

Quality requirements

Whatever the approach to software development may be, the final program must satisfy some fundamental properties. The following five properties are among the most relevant:

- **Efficiency**/**performance**: the amount of system resources a program consumes (processor time, memory space, slow devices such as disks, network bandwidth and to some extent even user interaction): the less, the better. This also includes correct disposal of some resources, such as cleaning up temporary files and lack of memory leaks.
- **Reliability**: how often the results of a program are correct. This depends on conceptual correctness of algorithms, and minimization of programming mistakes, such as mistakes in resource management (e.g. buffer overflows and race conditions) and logic errors (such as division by zero).
- **Robustness**: how well a program anticipates problems not due to programmer error. This includes situations such as incorrect, inappropriate or corrupt data, unavailability of needed resources such as memory, operating system services and network connections, and user error.
- **Usability**: the ergonomics of a program: the ease with which a person can use the program for its intended purpose, or in some cases even unanticipated purposes. Such issues can make or break its success even regardless of other issues. This involves a wide range of textual, graphical and sometimes hardware elements that improve the clarity, intuitiveness, cohesiveness and completeness of a program's user interface.
- **Portability**: the range of computer hardware and operating system platforms on which the source code of a program can be compiled/interpreted and run. This depends on differences in the programming facilities provided by the different platforms, including hardware and operating system resources, expected behaviour of the hardware and operating system, and availability of platform specific compilers (and sometimes libraries) for the language of the source code.

Algorithmic complexity

The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of

problem. For this purpose, algorithms are classified into *orders* using so-called Big O notation, *O(n)*, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances.

Methodologies

The first step in most formal software development projects is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). There exist a lot of differing approaches for each of those tasks. One approach popular for requirements analysis is Use Case analysis.

Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA). The Unified Modeling Language (UML) is a notation used for both OOAD and MDA.

A similar technique used for database design is Entity-Relationship Modeling (ER Modeling).

Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.

Measuring language usage

It is very difficult to determine what are the most popular of modern programming languages. Some languages are very popular for particular kinds of applications (e.g., COBOL is still strong in the corporate data center, often on large mainframes, FORTRAN in engineering applications, scripting languages in web development, and C in embedded applications), while some languages are regularly used to write many different kinds of applications.

Methods of measuring language popularity include: counting the number of job advertisements that mention the language[10], the number of books teaching the language that are sold (this overestimates the importance of newer languages), and estimates of the number of existing lines of code written in the language (this underestimates the number of users of business languages such as COBOL).

Debugging

A bug which was debugged in 1947.

Debugging is a very important task in the software development process, because an incorrect program can have significant consequences for its users. Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking

as other languages. Use of a static analysis tool can help detect some possible problems.

Debugging is often done with IDEs like Visual Studio, NetBeans, and Eclipse. Standalone debuggers like gdb are also used, and these often provide less of a visual environment, usually using a command line.

Programming languages

Main articles: Programming language and List of programming languages

Different programming languages support different styles of programming (called *programming paradigms*). The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Ideally, the programming language best suited for the task at hand will be selected. Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute.

Allen Downey, in his book *How To Think Like A Computer Scientist*, writes:

> The details look different in different languages, but a few basic instructions appear in just about every language:
>
> - **input**: Get data from the keyboard, a file, or some other device.
> - **output**: Display data on the screen or send data to a file or other device.
> - **arithmetic**: Perform basic arithmetical operations like addition and multiplication.
> - **conditional execution**: Check for certain conditions and execute the appropriate sequence of statements.
> - **repetition**: Perform some action repeatedly, usually with some variation.

Many computer languages provide a mechanism to call functions provided by libraries. Provided the functions in a library follow the appropriate runtime conventions (eg, method of passing arguments), then these functions may be written in any other language.

Programmers

Main article: Programmer
See also: Software developer and Software engineer

Computer programmers are those who write computer software. Their jobs usually involve:

- Coding
- Compilation
- Documentation
- Integration
- Maintenance
- Requirements analysis
- Software architecture
- Software testing
- Specification
- Debugging

References

1. ^ Paul Graham (2003). *Hackers and Painters*. http://www.paulgraham.com/hp.html. Retrieved on 2006-08-22.

2. ^ Kenneth E. Iverson, the originator of the APL programming language, believed that the Sapir–Whorf hypothesis applied to computer languages (without actually mentioning the hypothesis by name). His Turing award lecture, "Notation as a tool of thought", was devoted to this theme, arguing that more powerful notations aided thinking about computer algorithms. Iverson K.E.,"Notation as a tool of thought", *Communications of the ACM*, 23: 444-465 (August 1980).
3. ^ New World Encyclopedia Online Edition New World Encyclopedia
4. ^ Al-Jazari - the Mechanical Genius, MuslimHeritage.com
5. ^ A 13th Century Programmable Robot, University of Sheffield
6. ^ Fowler, Charles B. (October 1967), "The Museum of Music: A History of Mechanical Instruments", *Music Educators Journal* **54** (2): 45–49, doi:10.2307/3391092
7. ^ Columbia University Computing History - Herman Hollerith
8. ^ [1]
9. ^ [2]
10. ^ Survey of Job advertisements mentioning a given language>

Further reading

- Weinberg, Gerald M., *The Psychology of Computer Programming*, New York: Van Nostrand Reinhold, 1971

See also

Main article: Outline of computer programming

- ACCU (organisation)
- Association for Computing Machinery
- Computer programming in the punch card era
- Hello world program
- List of basic computer programming topics
- List of computer programming topics
- Programming paradigms
- Software engineering
- *The Art of Computer Programming*

[edit] External links

Wikibooks has a book on the topic of ***Computer programming***

Wikibooks has a book on the topic of ***Windows Programming***

- Programming Wikia
- Programming Wiki
- How to Think Like a Computer Scientist - by Jeffrey Elkner, Allen B. Downey and Chris Meyers

Major fields of computer science

Mathematical foundations	Mathematical logic · Set theory · Number theory · Graph theory · Type theory · Category theory · Numerical analysis · Information theory

Theory of computation	Automata theory · Computability theory · Computational complexity theory · Quantum computing theory
Algorithms and data structures	Analysis of algorithms · Algorithm design · Computational geometry
Programming languages and Compilers	Parsers · Interpreters · Procedural programming · Object-oriented programming · Functional programming · Logic programming · Programming paradigms
Concurrent, Parallel, and Distributed systems	Multiprocessing · Grid computing · Concurrency control
Software engineering	Requirements analysis · Software design · **Computer programming** · Formal methods · Software testing · Software development process
System architecture	Computer architecture · Computer organization · Operating systems
Telecommunication & Networking	Computer audio · Routing · Network topology · Cryptography
Databases	Data mining · Relational databases · SQL • OLAP
Artificial intelligence	Automated reasoning · Computational linguistics · Computer vision · Evolutionary computation · Machine learning · Natural language processing · Robotics
Computer graphics	Visualization · Image processing
Human computer interaction	Computer accessibility · User interfaces · Wearable computing · Ubiquitous computing · Virtual reality
Scientific computing	Artificial life · Bioinformatics · Cognitive Science · Computational chemistry · Computational neuroscience · Computational physics · Numerical algorithms · Symbolic mathematics

NOTE: Computer science can also be split up into different topics or fields according to the ACM Computing Classification System.

Software engineering

Fields	Requirements analysis • Software design • **Computer programming** • Formal methods • Software testing • Software deployment • Software maintenance
Concepts	Data modeling • Enterprise architecture • Functional specification • Modeling language • Programming paradigm • Software • Software architecture • Software development methodology • Software development process • Software quality • Software quality assurance • Structured analysis
Orientations	Agile • Aspect-oriented • Object orientation • Ontology • Service orientation • SDLC
Models	*Development models*: Agile • Iterative model • RUP • Scrum • Spiral model • Waterfall model • XP • V-Model *Other models*: Automotive SPICE • CMMI • Data model • Function model • IDEF • Information model • Metamodeling • Object model • Systems model • View model • UML
Software	Kent Beck • Grady Booch • Fred Brooks • Barry Boehm • Ward Cunningham • Ole-

engineers	Johan Dahl • Tom DeMarco • Edsger W. Dijkstra • Martin Fowler • C. A. R. Hoare • Watts Humphrey • Michael A. Jackson • Ivar Jacobson • Craig Larman • James Martin • Bertrand Meyer • David Parnas • Winston W. Royce • James Rumbaugh • Niklaus Wirth • Edward Yourdon
Related fields	Computer science • Computer engineering • Enterprise engineering • History • Management • Mathematics • Project management • Quality management • Software ergonomics • Systems engineering

Categories: Software development process | Computer programming