Suggested Readings in Computer Programming

A brief, informal, annotated bibliography and partial list of terms and works

Abelson, Harold, et al. 1996. *Structure and Interpretation of Computer Programs – 2nd Edition*. The MIT Press. 657 pp. ISBN: 978-0262510875. A more advanced textbook that was used for introductory computer science classes at MIT, using the Lisp programming language. MIT notwithstanding, I'm not sure this text is entirely suitable for beginning students. (There are also a JavaScript version, see Henz, Martin, below; and an interactive Lisp-base version, see Xuanji, below. If you're going to use a Lisp-based version, I recommend the interactive version.)

Access Control List (ACL) – A computer file or database listing users and their authorization level for access to files. See Wikipedia: https://en.wikipedia.org/wiki/Access-control_list.

Alexander, Christopher, 1964. *Notes on the Synthesis of Form*. Harvard University Press. 224 pgs. ISBN: 978-0674627512. Primarily about building architecture and design, its concepts apply to computer and software design, too.

Basic programming language – A simplified ("basic") programming language for beginners (the "B" stands for "Beginners'"). Computer scientists tend to deprecate basic because wasn't a very good language from a computer science point of view. But, as they all tend to do, the Basic language has evolved and "this isn't your father's Basic any more". The Visual Basic versions, for instance, are now modern, object-oriented, block-structured languages that can hold their own in many arenas. See Wikipedia: https://en.wikipedia.org/wiki/BASIC.

Bentley, Jon. 1999. *Programming Pearls 2nd Edition*. Addison-Wesley Professional (2nd edition, September 27, 1999) . ISBN: 978-0201657883. A little more advanced, but a very good book, primarily about algorithms.

Brooks, Frederick – Fred Brooks is a professor of computer science and was the project manager for development of IBM's 360 computer and for its OS/360 operating system. Among many other things, he wrote *The Mythical Man-Month* and "No silver bullet: Essence and accident in software engineering." See Wikipedia: https://en.wikipedia.org/wiki/Fred_Brooks

____ Brooks 1995. *Mythical Man-Month, The: Essays on Software Engineering, Anniversary Edition Anniversary Edition* Addison-Wesley Professional (Anniversary edition (August 2, 1995). ISBN: 978-0201835953. This is an excellent book, though it does not concentrate on programming but goes far beyond. I highly recommend it.

____ Brooks 1986. "No Silver Bullet—Essence and Accident in Software Engineering". Proceedings of the IFIP Tenth World Computing Conference: 1069–1076. Also available as "No Silver Bullet—Essence and Accident in Software Engineering". IEEE Computer. 20 (4), April 1987: 10–19. An excellent article, the (oversimplified) gist of which is that we've already picked all the "low hanging fruit", so software engineering is going to be hard from here on in.

Bug – see Software Bug, below. A software defect or error. Programs don't "catch" bugs like people catch diseases; rather, bugs are defects (unintentionally) "injected" by programmers.

Capability Maturity Model (CMM) – A model to rate an organization's level of excellence in software engineering (SE) management processes. Initiated by Watts Humphrey (see below) at Carnegie Mellon University's Software Engineering Institute. Sometimes incorrectly thought to be a model for doing software engineering, it is rather for processes in SE management. See Wikipedia: https://en.wikipedia.org/wiki/Capability_Maturity_Model

Code Complete – see McConnell, Steve, below. An excellent and comprehensive book about programming, primarily for practitioners as opposed to new students. Does not replace structured programming, but to be used *with* it. Includes object-oriented programming. Examples are in several programming languages.

"Danger, Will Robinson!" – A catch phrase warning from the Robot to 9-year-old Will Robinson in the campy 1960s TV series *Lost in Space*. See Wikipedia: https://en.wikipedia.org/wiki/Lost_in_Space#Catchphrases.

Debugging – corrective maintenance to fix errors. See also https://en.wikipedia.org/wiki/Debugging.

Dijkstra, Edsger W. – A consummate early computer scientist and software engineer. See Wikipedia: https://en.wikipedia.org/wiki/Edsger_W._Dijkstra.

____ Dijkstra 1975. "How do we tell truths that might hurt?", EWD498, 18 June 1975. A list of observations about programming and especially programming languages, not all favorable.

____ Dijkstra 1969. "Structured programming", EWD268, August 1969. An early, quite possibly the earliest, paper introducing structured programming.

EDVAC ("Electronic Discrete Variable Automatic Computer") – The first stored program electronic digital computer (in 1945). Based on the *von Neumann computer architecture*. See Wikipedia: https://en.wikipedia.org/wiki/EDVAC.

Elements of Programming Style – see Kernighan, Brian, below. Though rather dated (using Fortran and PL/1), a very good book that takes real programs as examples and rewrites them to be clearer, better, and more correct.

ENIAC ("Electronic Numerical Integrator and Computer" or "... and Calculator") – The first electronic digital computer (in 1945). Programmed via plugboards and connecting cables rather than via stored programs as in the von Neumann architecture and in all modern computers. See Wikipedia: https://en.wikipedia.org/wiki/ENIAC.

Factorial – The mathematical function *N!* that repeatedly multiplies 1 × 2 × 3 × 4 × ... up to *N*. See Wikipedia: https://en.wikipedia.org/wiki/Factorial.

Felleisen, Matthias, et al. 2018. *How to Design Programs, second edition: An Introduction to Programming and Computing*. (HTDP). The MIT Press. 792 pgs. ISBN: 978-0262534802. An excellent and comprehensive advanced introduction to how to design computer programs oriented toward computer science. It uses a Lisp-like prefix-notation "functional programming" language, which can be powerful but daunting at first. See https://htdp.org/

Flowchart – A diagramming technique that shows actions and their connections. There are several types of flowcharts, the most common of which is a procedural flowchart, showing the flow of control between the procedural blocks in a computer program. See Wikipedia: https://en.wikipedia.org/wiki/Flowchart.

Finite State Machine (FSM) – A particular way of organizing a program or hardware that operates by transitioning from one state to another, performing functions along the way. See Wikipedia: https://en.wikipedia.org/wiki/Finite-state_machine.

Fundamental Structure Theorem – A proven principle of programming that relates flowcharts to the logic structures used to represent and implement them. Specifically: programs (any that can be flowcharted), can be programmed using only three basic control structures: Sequence, If-Then-Else, and Do-While. see Linger, Richard, et al. 1979. *Structured programming – theory and practice*.

Gauss, Edward, 1982. "The 'Wolf Fence' algorithm for debugging", *Communications of the ACM*. 25,11, 01 November 1982. https://dl.acm.org/doi/10.1145/358690.358695. A method for troubleshooting computer program errors by repetitively narrowing down their possible location.

Glass, Robert. *Software Creativity 2.0*. developer.* Books. 484 pgs. ISBN: 978-0977213313. Programming and software development from a problem-solving point of view. Highly recommended, though probably not for beginners.

Henz, Martin, MIT Electrical Engineering and Computer Science. *Structure and Interpretation of Computer Programs: JavaScript Edition*. The MIT Press. 640 pgs. ISBN: 978-0262543231. https://sourceacademy.org/sicpjs/index. The JavaScript version of SICP. (There are also a Lisp-based version, see Abelson,

Harold, above; and an interactive Lisp-base version, see Xuanji, below. If you're going to use a Lisp-based version, I recommend the interactive version.)

Hopper, Grace – Rear Admiral Grace Murray Hopper, PhD, is a fascinating computer pioneer. "The second programmer on the first computer". Among other things, she was the inventor of the COBOL programming language. See Wikipedia: https://en.wikipedia.org/wiki/Grace_Hopper.

How to Design Programs (HTDP) – See Felleisen, above. An excellent and comprehensive advanced introduction to how to design computer programs oriented toward computer science. It uses a Lisp-like prefix-notation "functional programming" language, which can be powerful but daunting at first. See https://htdp.org/

How to Solve It: A New Aspect of Mathematical Method, by George Polya (1945) – A good, brief summary of the method, which is entirely consistent with Branscomb's "**Error! Reference source not found.**" (p. **Error! Bookmark not defined.**). See Wikipedia article https://en.wikipedia.org/wiki/How_to_Solve_It,

HTDP – see Felleisen, *How to Design Programs*, above.

Humphrey, Watts – Former vice president and manager of software development at IBM, and professor at Carnegie-Mellon University's Software Engineering Institute (SEI), where he initiated the software engineering management Capability Maturity Model (CMM; see above). Author of Intro to the *Personal Software Process*, the *Team Software Process*, and several other books on software engineering and software development. I recommend just about anything he's written. See Wikipedia: https://en.wikipedia.org/wiki/Watts_Humphrey.

____ Humphrey 1996. *Introduction to the Personal Software Process 1st Edition*. Addison-Wesley Professional (1st edition (January 1, 1996): 278 pp. ISBN: 978-0201548099. Essentially an earlier edition of *PSP: A Self-Improvement Process for Software Engineers*. Details a process for both developing software and especially estimating how long it will take to do so. A very valuable resource for professional programmers, most of whom are typically poor estimators.

____ Humphrey 2005. *PSP: A Self-improvement Process For Software Engineers 1st Edition*. Addison-Wesley Professional (1st edition (March 15, 2005): 345 pp. ISBN: 978-0321305497. Essentially a newer edition of *Introduction to the Personal Software Process*, oriented toward professional software engineers. Details a process for both developing software and especially estimating how long it will take to do so. A very valuable resource for professional programmers, most of whom are typically poor estimators.

Isaiah 28:10-13 – A prophet's ancient observation that sounds much like modern computer programs. See Bible Gateway. http://www.biblegateway.com/passage/?search=Isaiah28:10-13&version=KJV.

Kernighan, Brian, and Rob Pike. 1999. *The Practice of Programming*. Addison-Wesley. 288 pgs. ISBN: 978-0201615869. Another good book somewhat similar in scope, though shorter in volume, to *Code Complete*. Uses several languages.

Kernighan, Brian and P.J. Plauger. 1978. *The Elements of Programming Style, 2nd Edition 2nd Edition*. McGraw-Hill. ISBN: 978-0070342071. Though rather dated (using Fortran and PL/1), a very good book that takes real programs as examples and rewrites them to be clearer, better, and more correct.

Ledgard, Henry, 1975. *Programming Proverbs: Principles of Good Programming with Numerous Examples to Improve Programming Style and Proficiency*. Hayden Book Company. ISBN: 978-0810455221. As its subtitle says, it is principles of good programming practice. It does not replace structured programming, but to be used *with* it. A little dated (uses Algol and PL/1) but good practices, nevertheless.

Linger, Richard, Harlan Mills, and Bernard Witt. 1979. *Structured programming – theory and practice*. Addison-Wesley. ISBN 978-0-201-14461-1. An excellent book concentrating on the computer science basis of structured programming, though rather pricey. For a more accessible summary, see Mills, Harlan, "How to write correct programs and know it", below.

Lisp programming language – An unusual, non-traditional programming language used for linked list processing ("Lisp" comes from "List Processor"), for early artificial intelligence programming, teaching computer science, and the like. Lisp is based on *Polish notation* (see below). Both Lisp and prefix notation are powerful but can be daunting until we're used to them. See Wikipedia: https://en.wikipedia.org/wiki/Lisp_(programming_language).

Loop Invariant – A condition that remains (invariant) every time the test in a loop is executed. See Wikipedia: https://en.wikipedia.org/wiki/Loop_invariant.

McConnell, Steve. 2004. *Code Complete: A Practical Handbook of Software Construction, 2nd Edition*. Microsoft Press (2nd edition (July 7, 2004) . ISBN: 978-0735619678. An excellent and comprehensive book about programming, primarily for practitioners as opposed to new students. Does not replace structured programming, but to be used *with* it. Includes object-oriented programming. Examples are in several programming languages.

Mills, Harlan – A computer scientist critical to the early development of software engineering and structured programming. I recommend just about anything he's written. See Wikipedia: https://en.wikipedia.org/wiki/Harlan_Mills.

____ Mills 1975. "How to write correct programs and know it". *Proc. 1975 International Conference on Reliable Software*, Los Angeles, Apr. 21-23, 1975. IEEE Cat. No. 5CH0940-7CSR. ACM Digital Library: https://dl.acm.org/doi/10.1145/800027.808459. Also available at: https://ia801709.us.archive.org/23/items/how-to-write-correct-programs-and-know-

it/How%20to%20Write%20Correct%20Programs%20and%20Know%20It.pdf. A more accessible summary of, and predecessor to, Linger, Mills, and Witt's *Structured Programming*.

Nassi-Schneiderman Chart – A type of block diagram that in some ways is simpler than a flowchart and allows only valid structures of the Fundamental Structure Theorem. See Wikipedia: https://en.wikipedia.org/wiki/Nassi%E2%80%93Shneiderman_diagram.

"No Silver Bullet " see Brooks, Frederick, "No Silver Bullet – Essence and Accident in Software Engineering". An excellent article, the (oversimplified) gist of which is that we've already picked all the "low hanging fruit", so software engineering is going to be hard from here on in.

On Programming: A How-to Guide, by Karl Schank. An introductory guide to writing computer programs well, regardless of programming language, and knowing they're right. Amazon: https://www.amazon.com/Programming-How-introductory-regardless-programming/dp/8324203737

Polish notation – A mathematical notation in which the operator comes first, rather than between the operands. See Wikipedia: https://en.wikipedia.org/wiki/Polish_notation. See also Prefix Notation, below.

Personal Software Process, and *PSP* – See Humphrey, Watts, above. Details a process for developing software and estimating how long it will take to do so. A very valuable resource for professional programmers, most of whom are typically poor estimators.

Pragmatic Programmer – An excellent and comprehensive book about programming, primarily for practitioners as opposed to new students. Does not replace structured programming, but to be used *with* it. Highly recommended. For information, see Wikipedia: https://en.wikipedia.org/wiki/The_Pragmatic_Programmer. For the book citation, see Thomas, David, below.

Prefix notation – Polish notation or prefix notation puts the operation *before* the arguments. For example, 2 times 3 would be written `(* 2 3)` rather than the more familiar `2 × 3` or `2 * 3`, and 4 plus 5 plus 6 would be written `(+ 4 5 6)` rather than `4 + 5 + 6`. This is powerful but can be daunting until we're used to it. See Wikipedia: https://en.wikipedia.org/wiki/Polish_notation.

Programming Pearls – see Bentley, John, above. A little more advanced, but a very good book, primarily about algorithms.

Programming Proverbs – see Ledgard, Henry, above. As its subtitle says, its principles of good programming practice. It does not replace structured programming, but to be used *with* it. A little dated (uses Algol and PL/1) but good practices, nevertheless.

Recursion – As noted above, recursion is another form of repetition than loops. In recursion, a program (usually a subroutine) or invokes itself. See Wikipedia:

https://en.wikipedia.org/wiki/Recursion, particularly see https://en.wikipedia.org/wiki/Recursion#In_computer_science.

SICP – see *Structure and Interpretation of Computer Programs*, below. There are at least three versions to choose from.

Software bug – A software defect or error. The term comes from an incident in which a moth was trapped in an electromechanical relay in one of the very first electromagnetic digital computers around 1947, caused a malfunction, and resulted in the term "bug". See photo of the console log here: https://en.wikipedia.org/wiki/Grace_Hopper#/media/File:First_Computer_Bug,_1945.jpg. Programs don't "catch" bugs like people catch diseases; rather, bugs are defects (unintentionally) "injected" by programmers. See Wikipedia: https://en.wikipedia.org/wiki/Software_bug#History.

Software engineering (*SE*) – Applying engineering approaches and techniques to the development of software. Includes development processes, usually the entire development lifecycle from systems analysis through design, construction and programming, testing, installation / implementation, and maintenance.

Structure and Interpretation of Computer Programs (SICP) – There are several versions, including the three cited herein. See:

- Abelson, Harold for the Lisp-based version;
- Henz, Martin for JavaScript-based version; and
- Xuanji, for the online, interactive Lisp-based version.

For those interested in a more advanced textbook that was used for introductory computer science classes at MIT. If you're going to use the Lisp-based version, I recommend the Xuanji online, interactive version.

Structured programming – theory and practice – see Linger, Richard, above. An early explanation of structured programming and the Fundamental Structure Theorem.

Thomas, David and Andrew Hunt, 2019. *The Pragmatic Programmer: Your journey to mastery, 20th Anniversary Edition, 2nd Edition* Addison-Wesley Professional (2nd edition (July 30, 2019) 522 pp. An excellent and comprehensive book about programming, primarily for practitioners as opposed to new students. Does not replace structured programming, but to be used *with* it. I recommend pretty much anything they've written.

Yourdon, Edward. 1988. *Modern Structured Analysis First Edition*. Prentice Hall. 688 pgs. ISBN: 978-0135986240. An introduction to "structured" computer systems analysis and design, for those who are interested. In spite of the title, it's no longer very "modern".

Von Neumann computer architecture – The foundational "stored program" architecture of modern computers, considerably extended and expanded since its inception by John von Neumann in the EDVAC computer in 1945. See Wikipedia: https://en.wikipedia.org/wiki/Von_Neumann_architecture.

Wolf Fence debugging method– see Gauss, Edward, "The 'Wolf Fence' algorithm for debugging", above. A method for troubleshooting computer program errors by repetitively narrowing down their possible location. See also https://en.wikipedia.org/wiki/Debugging#Techniques *"Wolf fence" algorithm*.

Xuanji. *Structure and Interpretation of Computer Programs: Interactive Version*. (undated, work in progress). For those interested in a more advanced textbook that was used for introductory computer science classes at MIT. If you're going to use the Lisp-based version, I recommend this online, interactive version. See https://xuanji.appspot.com/isicp/.