

edgeangeles.com

All-access to the the twisted mind of Edge

Computer Program Development

SOFTWARE DEVELOPMENT CYCLE

by

David Lubar

Software doesn't just appear on the shelves by magic. That program shrink-wrapped inside the box along with the indecipherable manual and twelve-paragraph disclaimer notice actually came to you by way of an elaborate path through the most rigid quality control methods on the planet. Here, shared for the first time with the general public, are the inside details of the program development cycle.

1. Programmer produces code he believes is bug-free.
2. Product is tested. Twenty bugs are found.
3. Programmer fixes ten of the bugs and explains to the testing department that the other ten aren't really bugs.
4. Testing department finds that five of the fixes didn't work and discovers fifteen new bugs.
5. See 3.
6. See 4.
7. See 5.
8. See 6.
9. See 7.
10. See 8.
11. Due to marketing pressure and extremely pre-mature product

announcement based on over-optimistic programming schedule, the product is released.

12. Users find 137 new bugs.

13. Original programmer, having cashed his royalty check, is nowhere to be found.

14. Newly-assembled programming team fixes almost all of the 137 bugs, but introduces 456 new ones.

15. Original programmer sends underpaid testing department a postcard from Fiji. Entire testing department quits.

16. Company is bought in hostile takeover by competitor using profits from their latest release, which had 783 bugs.

17. New CEO is brought in by board of directors. He hires programmer to redo program from scratch.

18. Programmer produces code he believes is bug-free.

Copyright © 1996 by David Lubar

from <http://davidlubar.com/cycle.html>

Computer program development expanded outline

Program Development Cycle

1. Problem Identification and Analysis
 1. Output
 2. Given
 3. Input
 4. Restrictions
2. Logic Formulation
 1. Data representation
 1. Variables and constants
 2. Numeric
 1. Integer
 2. Real
 3. Literal
 1. Character
 2. String
 4. Logical
 5. Operators
 6. Expressions
 2. Algorithms
 1. Algorithm development

2. Algorithm representation
 1. Pseudocode
 2. Flowchart
3. Coding
 1. Program code and programming languages
 2. Abstraction
 1. Assembly code
 2. Machine code
4. Testing and Debugging
5. Storage and Maintenance




Pseudocode convention

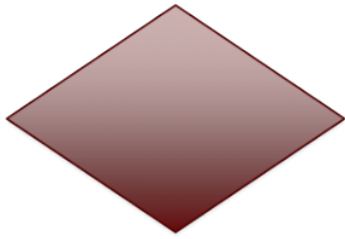
- Numbered pseudocode
 - uses numbers for direction of flow
 - employs the **goto** statement
- Labeled pseudocode
 - uses labels for flow of execution
 - also employs the **goto** statement
- Indented pseudocode
 - groups of common statements or blocks are indented at the same level

Use the \leftarrow symbol for variable assignment

Use the **goto** statement to control the flow of a program

Flowchart convention

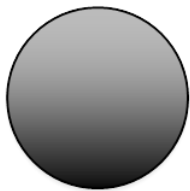
Symbol	Description
	<ul style="list-style-type: none"> • oval • marks the beginning or end of a process • appears exactly twice in any complete flowchart • contains either <ul style="list-style-type: none"> - START and STOP - BEGIN and END
	<ul style="list-style-type: none"> • parallelogram • either <ul style="list-style-type: none"> - accepts input - provides output
	<ul style="list-style-type: none"> • rectangle • performs <ul style="list-style-type: none"> - computations - assignments • may contain several assignments in one box



- rhombus/diamond
- only symbol among the elementary symbols that has two exits
- flow is determined by a dichotomous condition



- rectangle with double vertical edges
- associated with algorithms in a different flowchart



- circle
- used to organize flow lines on one page
- typically contains a letter
 - each letter may occur more than twice on a page
 - each letter has only one exit



- pentagon
- used to transfer flow from one page to another
- typically contains a page number
- contains a page number and a letter if more than one exit exist on a page

- Flow moves downward for clean diagrams
 - flow may move sideward from a decision symbol
 - connectors may be used to return to a previous section
- Each symbol should have a maximum of one flow entrance
 - junctions could be used; or
 - flow lines may just point towards other flow lines

Algorithm complexity

- Measures how complicated your algorithm is
- Takes into consideration resources
 - time (speed)
 - space (memory)