DESIRABLE PROGRAM CHARACTERISTICS

Integrity. This refers to the accuracy of the calculations. It should be clear that all other program enhancements will be meaningless if the calculations are not carried out correctly. Thus, the integrity of the calculations is an absolute necessity in any computer program.

For example, our program must *compute a (i) spanning tree of (ii) minimal cost.*

Clarity refers to the overall readability of the program, with particular emphasis on its underlying logic. If a program is clearly written, it should be possible for another programmer to follow the program logic without undue effort.

One of the objectives in the design of C is the development of clear, readable program through an orderly and disciplined approach to programming.

Simplicity. The clarity and accuracy of a program are usually enhanced by keeping things as simple as possible, consistent with the overall program objective. In fact, it may be desirable to sacrifice a certain amount of computational efficiency in order to maintain a relatively simple, straightforward program structure.

Efficiency is concerned with execution speed and efficient memory utilization. These are generally important goals, though they should not be obtained at the expense of clarity or simplicity.

For example, we should use the greedy algorithm for finding the minimal cost spanning tree, rather than, say, a brute force search algorithm.

Modularity. Many programs can be broken down into a series of identifiable subtasks. It is good programming practice to implement each of these subtasks as a separate program module. In C such modules are written as functions. The use of a modular programming structure enhances the accuracy and clarity of a program, and it facilitates future program alterations.

For example, we have the function `main` *which reads in the input, and calls* `sort_by_weight` *and* `spanning_tree`*.*

Generality. Usually we want a program to be as general as possible, within reasonable limits.

For example, we want to input `N`*, and assign an array of appropriate size, rather than define* `N` *to be a particular value.*

DEVELOPING PROGRAMS

4 basic activities of any software development effort:

1. establishing the requirements
2. **creating a design**
3. implementing the code
4. testing the implementation

These activities overlap and interact.
(From *Java Software Solutions* by Lewis and Loftus.)

Requirements specify what a program must accomplish—they indicate the tasks that a program should perform, but not how to perform them.

An initial set of requirements are often provided by the person or group who want the software developed. However, these initial requirements are often incomplete or ambiguous.

The software developer must work to refine the requirements until all key decisions have been addressed.

For example, what type of "malformed" inputs to test for?

Design specifies how a program will accomplish its requirements.

One of the most fundamental design issues is defining the algorithm to be used in the program. An algorithm is often described using pseudocode.

For example, Kruskal's algorithm in our design for the program computing minimum cost spanning trees.

Implementation is the process of writing source code: it is the translation of design into a particular programming language (C in this course).

Too many programmers focus on implementation exclusively when it actually should be the <u>least creative</u> of all activities.

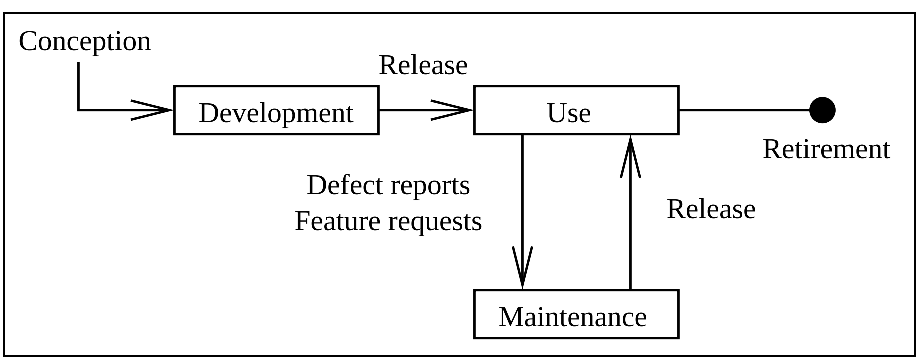
Testing a program involves running it multiple times with various inputs and observing the results. The goal of testing is to find errors.

SOFTWARE LIFE CYCLE

1. development

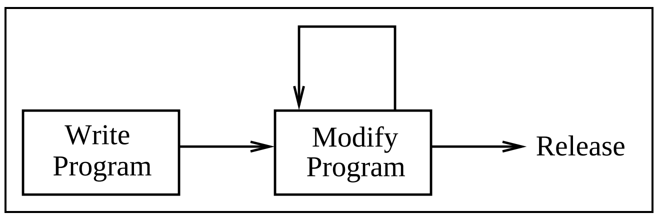
2. use

3. maintenance



DEVELOPMENT PROCESS MODELS

The build-and-fix approach.

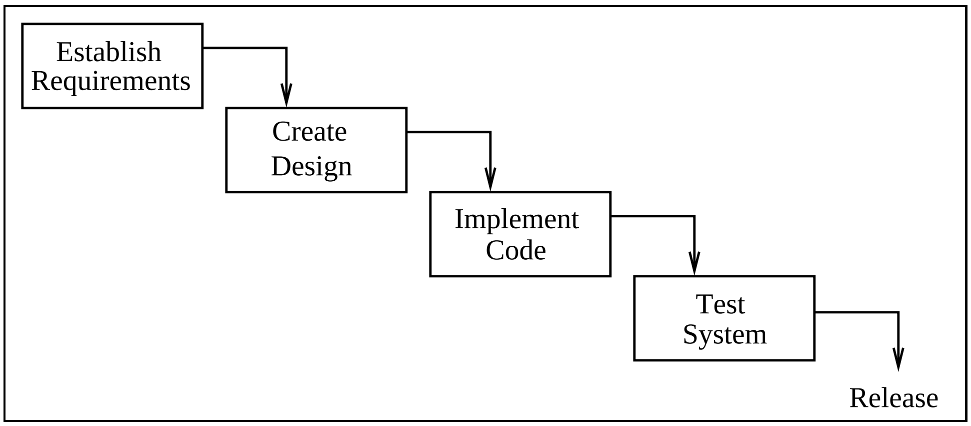


Programmer creates an initial version of the program, then continually modifies it until it has reached some level of acceptance.

Programmer is simply reacting to problems, as opposed to creating something worthy in the first place (i.e., *reactive* rather than *proactive* thinking).

Bad, *ad hoc*, reckless .

The (unrealistic) waterfall model.



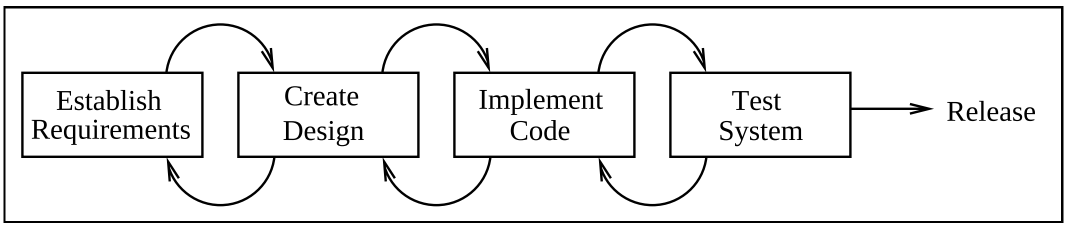
First offered in the early 1970s.

Linear: it does not allow for revisiting earlier stages.

Tremendous effort must go into each stage to ensure its completeness: unrealistic.

Does not take into account that the developmental activities are overlapping.

An iterative process.

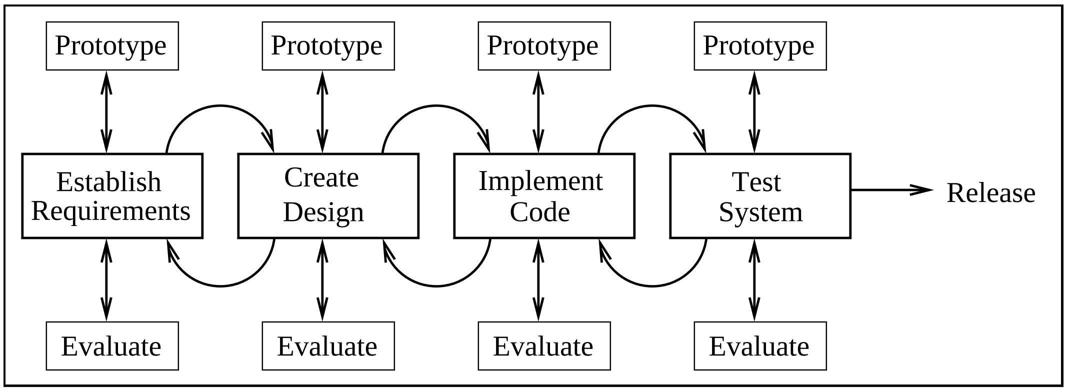


Earlier stages can be revisited (formally): waterfall model with backtracking.

Danger: rely on backtracking too much!

Backtracking should be used primarily to correct problems uncovered in later stages.

Iterative process with Prototyping.

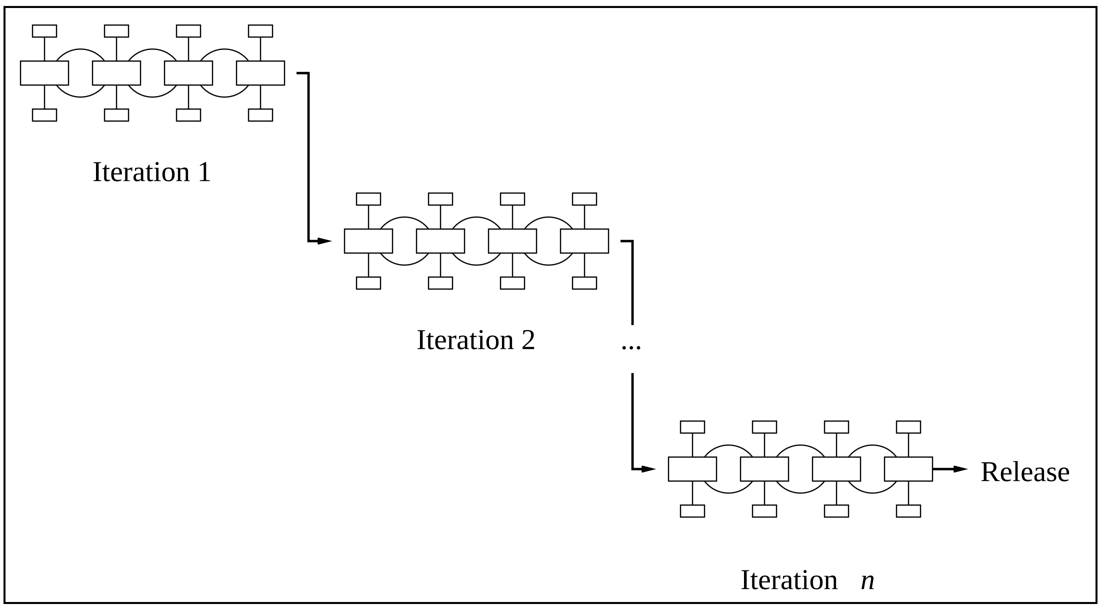


A prototype is a program that is created to explore a particular concept.

Sometimes a programmer doesn't know how to accomplish a particular task, or weather certain requirements are feasible, or the user interface acceptable to the client. Prototypes can be used to explore all these issues.

Before moving to the next step, the results of the current step must be evaluated. An evaluation of the implementation is done with a **code walkthrough** (a meeting to read and critique the code).

Iteration of the entire process.



The entire model can be used to develop a program iteratively.

One iteration of the model is used to build a specific part of a program.