Software development

The program development life cycle

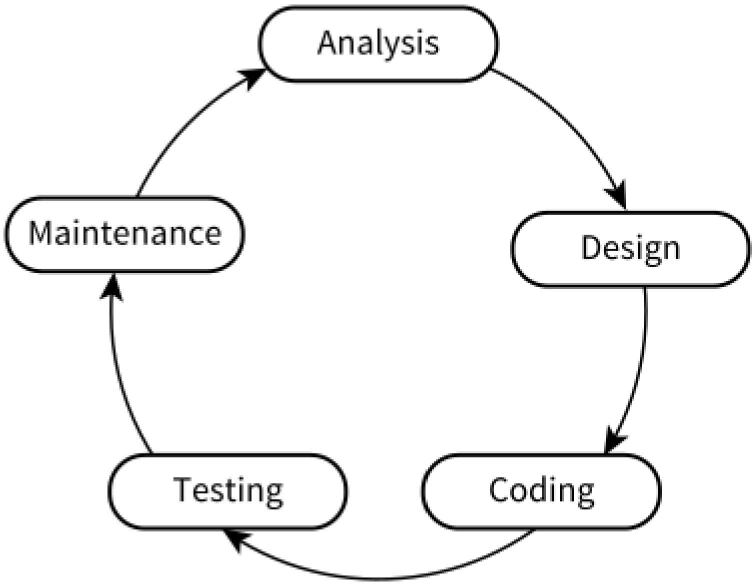
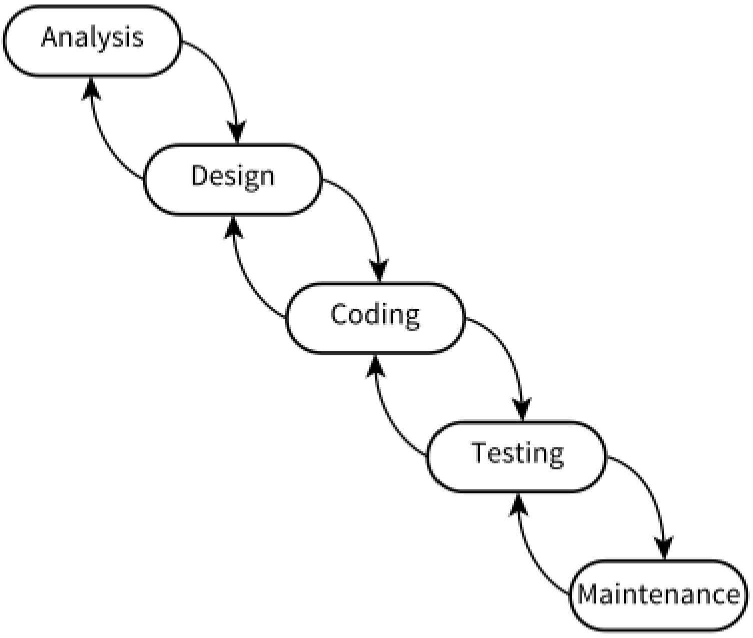


Figure 15.01 The program development life cycle

Waterfall model

> Goes from one stage to another after completion



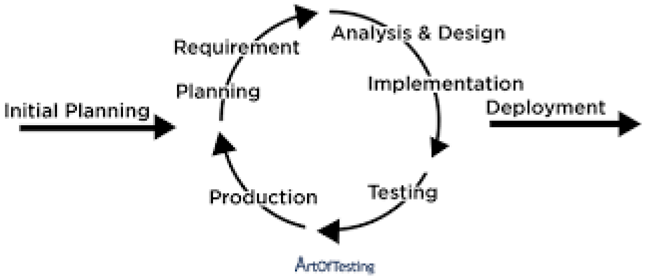
- Benefits of using waterfall model:
    - Simple to understand (clearly defined steps)
    - Easy to manage (fixed nature, specific outcome)
    - Stages are processed and completed one at a time

- Works well for smaller projects

- Drawbacks of waterfall model:
  - No working software until late during the life cycle
  - Not a good model for complex and Object oriented projects
  - cannot accomodate change
  - poor for long and ongoing projects
  - difficult to measure progress within stages
  - integration at end, so doesn't allow identifying potential issues early

Iterative model

> Starts with few requirements only and iteratively identifies and implements more.



- Benefits:
  - Working model at early stage
  - Results are obtained early and periodically
  - Risks are easy to identify and manage
  - Better suited for large projeects
  - Testing and debugging small subset is easy
  - Parallel development is possible

- Drawbacks:
  - Defining increment may require definition of the complete system
  - Hard to break down
  - More resources
  - Design issues due to lack of all requirements

RAD - Rapid Application Development

> Minimal planning, prototyping and integration

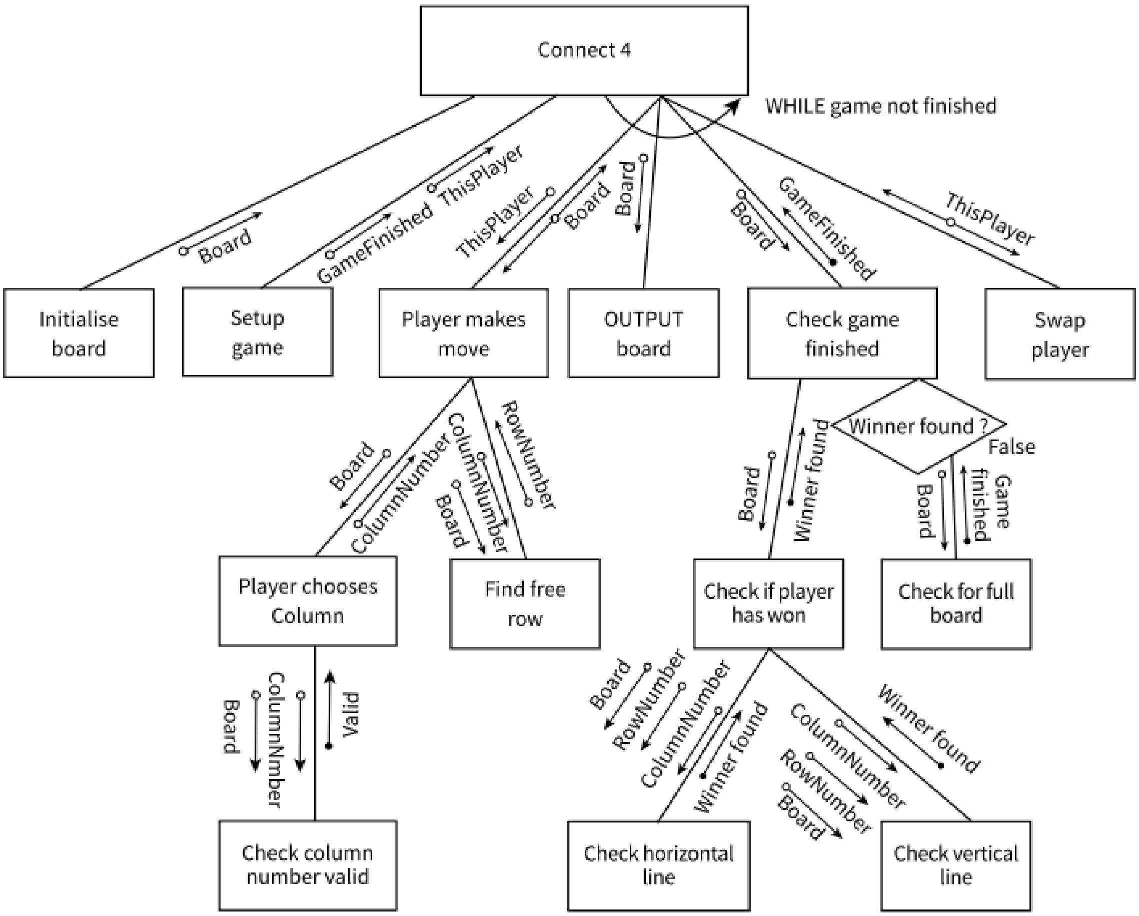
- Benefits:
  - Increased reusability
  - Quick initial reviews
  - Encourages customer feedback
  - Changing requirement can be accomodated
  - Increased productivity

- Drawbacks:
  - Only for system that can be modularized
  - Requires skilled team
  - Requires user involvement

Structure Charts

> Based on top-down approach i.e. stepwise refinement

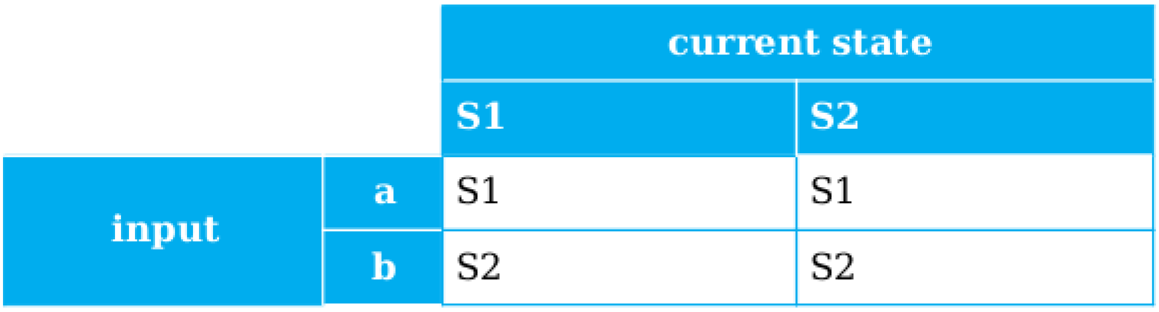
- Symbols:
  - Rectangle: Modules
  - Downward arrow: function call
    - text on downward arrow: parameters
  - Upward arrow: return value
  - Diamond: Decision
  - Arrow with solid round end: Boolean value/Flag
  - Double-headed arrow: variable updated within module



State-transition diagram

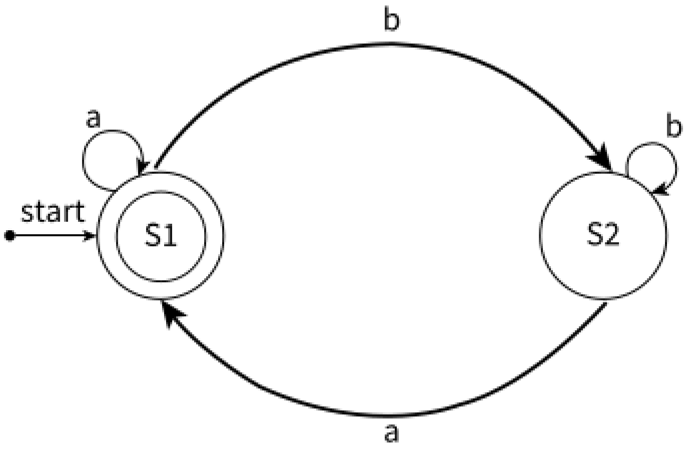
> FSM : Finite State Machine. An input to FSM causes state transition.

- State-transition-table

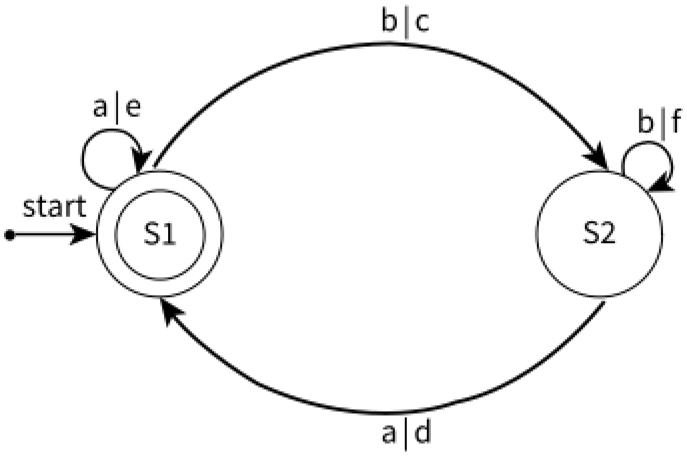


		current state	
		S1	S2
input	a	S1	S1
	b	S2	S2

- State-transition-diagram without output



- State-transition-diagram with output



- double circled state is halting/final/acceptting state
- FSM with outputs: Mealy Machine

Error and types

> Flaw in program that results in unintended behavior

- Syntax errors: Mistakes in code that violate rules of the programming language
- Runtime erorrs: Errors during execution. Eg. divide by 0, memory full etc.
- Logical error: Flaws in algorithm

> Compiler only finds syntax erorrs.

Testing methods

Stub testing

- Module with just headers and some output statement for acknowledgement => Stub
- Testing the interface of modules without actually implementing using stub => Stub Testing

Black box and white box testing

- Black box testing: Testing without seeing the program code. Testing by running the program and seeing output.

- White-box testing: Checking every path through the code.

- Dry running: Walking through the algorithm and creating trace table. Helps in finding errors in algorithm.

Others

- Unit testing: Testing one module to see if it works properly
- Integration testing: Testting if modules work together as a program.
- Alpha testing: Testing by development company.
- Acceptance testing: Testing by customer.
- Beta testing: Testing by a limited audience of potential users.

Test strategy, test plans and test data

- Types of test data:
  - Normal
  - Abnormal
  - Boundary/extreme/edge

Types of maintenance:

- Corrective maintenance: Solving erorrs in programs
- Adaptive maintenance: Changing functions, adding features etc.
- Perfective maintenance: Improving efficiecy./Optimizing.
