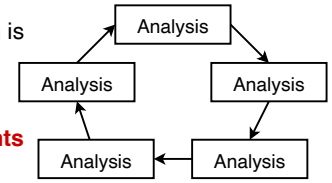
ALevel CS C15 Software Development (1)

Develop life cycle

Analysis:



1. The first step in solving a problem is to **investigate the issues** and the current system if there is one.
The problem needs to be defined clearly and precisely. A '**requirements specification**' is drawn up.
2. The next step is **planning a solution**. Sometimes there is more than one solution. You need to **decide which is the most appropriate**.
3. The third step is to **decide how to solve the problem**
   1. **bottom-up**: start with a small sub-problem and then build on this
   2. **top-down**: stepwise refinement using pseudocode, flowcharts or structure charts.

Design:
Plan your algorithm by drawing a flowchart or writing pseudocode.

Coding:
You implement your algorithm by converting your pseudocode into program code. When you start writing programs you might find it takes several attempts before the program compiles.

Testing:
Only thorough testing can ensure the program really works under all circumstances

Rapid Application Development (RAD) model

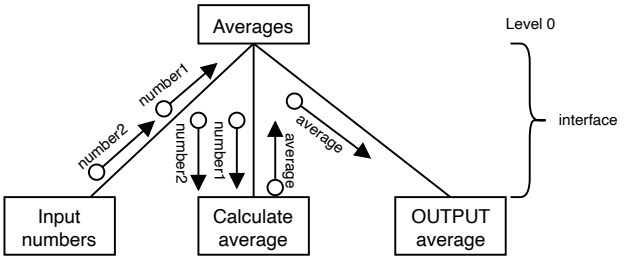
RAD is a software development methodology that uses minimal planning. Instead it uses prototyping. A prototype is a working model of part of the solution.
In the RAD model, the modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. There is no detailed preplanning. Changes are made during the development process.
The analysis, design, code and test phases are incorporated into a series of short, iterative development cycles.

Benefits:
1. Changing requirements can be accommodated.
2. Progress can be measured.
3. Productivity increases with fewer people in a short time.
4. Reduces development time.
5. Increases reusability of components.
6. Quick initial reviews occur.
7. Encourages customer feedback.
8. Integration from very beginning solves a lot of integration issues.

Drawbacks:
1. Only systems that can be modularised can be built using RAD.
2. Requires highly skilled developers/designers.
3. Suitable for systems that are component based and scalable.
4. Requires user involvement throughout the life cycle.
5. Suitable for projects requiring shorter development times.

Structure chart

top-level box is the name of the module
The arrow show how the parameters are passed between the modules. This parameter passing is known as the 'interface'.



Waterfall model

The arrows going down represent the fact that the results from one stage are input into the next stage. The arrows leading back up to an earlier stage reflect the fact that often more work is required at an earlier stage to complete the current stage.

Benefits:
1. Simple to understand as the stages are clearly defined.
2. Easy to manage due to the fixed stages in the model. Each stage has specific outcomes.
3. Stages are processed and completed one at a time.
4. Works well for smaller projects where requirements are very well understood.

Drawbacks:
1. No working software is produced until late during the life cycle.
2. Not a good model for complex and object-oriented projects.
3. Poor model for long and ongoing projects.
4. Cannot accommodate changing requirements.
5. It is difficult to measure progress within stages.
6. Integration is done at the very end, which doesn't allow identifying potential technical or business issues early.

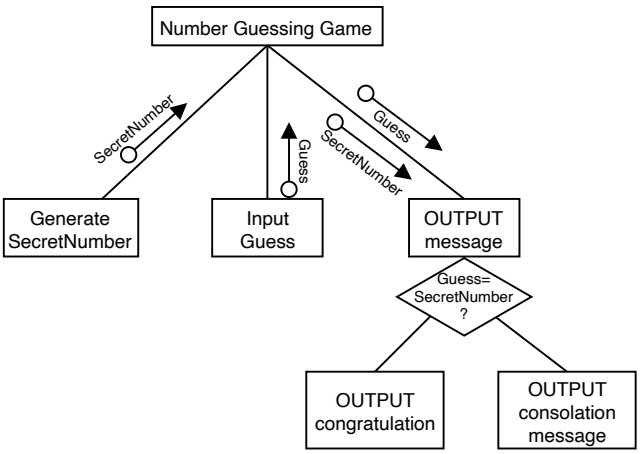
Iterative model

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development starts with the implementation of a small subset of the program requirements. Repeated (iterative) reviews to identify further requirements eventually result in the complete system.

Benefits:
1. There is a working model of the system at a very early stage of development, which makes it easier to find functional or design flaws. Finding issues at an early stage of development means corrective measures can be taken more quickly.
2. Some working functionality can be developed quickly and early in the life cycle.
3. Results are obtained early and periodically.
4. Parallel development can be planned.
5. Progress can be measured.
6. Less costly to change the scope/requirements.
7. Testing and debugging of a smaller subset of program is easy.
8. Risks are identified and resolved during iteration.
9. Easier to manage risk – high-risk part is done first.
10. With every increment, operational product is delivered.
11. Issues, challenges and risks identified from each increment can be utilised/applied to the next increment.
12. Better suited for large and mission-critical projects.
13. During the life cycle, software is produced early, which facilitates customer evaluation and feedback.

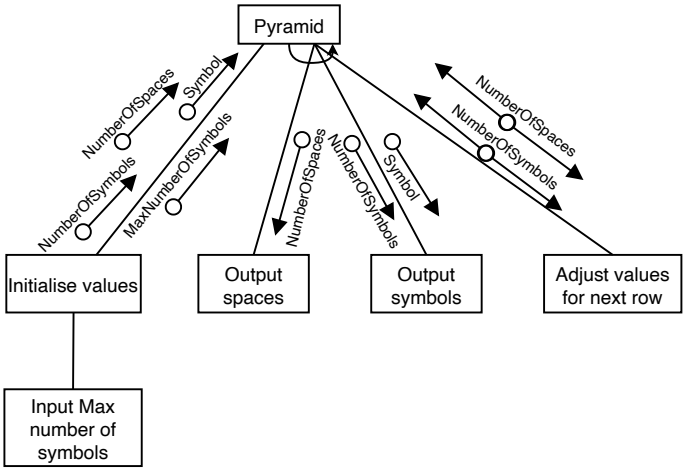
Drawbacks:
1. Only large software development projects can benefit because it is hard to break a small software system into further small serviceable modules.
2. More resources may be required.
3. Design issues might arise because not all requirements are gathered at the beginning of the entire life cycle.
4. Defining increments may require definition of the complete system.

selection



ALevel CS C15 Software Development (2)

repetition
An arrow with a solid round end ○→ show s that the value transferred is a flag (a boolean value)
A double-headed arrow ←○→ shows that the variable value is updated within the module.



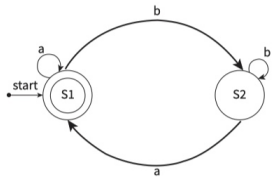
Pyramid pseudocode

```
MODULE Pyramid
   CALL SetValues(NumberOfSymbols, NumberOfSpaces, Symbol,
MaxNumberOfSymbols)
   REPEAT
      CALL OutputSpaces(NumberOfSpaces)
      CALL OutputSymbols(NumberOfSymbols, Symbol)
      CALL AdjustValuesForNextRow(NumberOfSpaces, NumberOfSymbols)
   UNTIL NumberOfSymbols > MaxNumberOfSymbols
ENDMODULE

PROCEDURE SetValues(NumberOfSymbols, NumberOfSpaces, Symbol,
MaxNumberOfSymbols)
   INPUT Symbol
   CALL InputMaxNumberOfSymbols
   NumberOfSpaces ← (MaxNumberOfSymbols – 1) / 2
   NumberOfSymbols ← 1
ENDPROCEDURE

PROCEDURE InputMaxNumberOfSymbols(MaxNumberOfSymbols)
   REPEAT
      INPUT MaxNumberOfSymbols
   UNTIL MaxNumberOfSymbols MOD 2 = 1
ENDPROCEDURE

PROCEDURE OutputSpaces(NumberOfSpaces)
   FOR Count ← 1 TO NumberOfSpaces
      OUTPUT Space // without moving to next line
   NEXT Count
ENDPROCEDURE

PROCEDURE OutputSymbols(NumberOfSymbols, Symbol)
   FOR Count← 1 TO NumberOfSymbols
      OUTPUT Symbol // without moving to next line
   NEXT Count
   OUTPUT Newline // move to the next line
ENDPROCEDURE

PROCEDURE AdjustValuesForNextRow(NumberOfSpaces, NumberOfSymbols)
   NumberOfSpaces ← NumberOfSpaces – 1
   NumberOfSymbols ← NumberOfSymbols + 2
ENDPROCEDURE
```

Test strategy:
1. flow of control: does the user get appropriate choices and does the chosen option go to the correct module?
2. validation of input: has all data been entered into the system correctly?
3. do loops and decisions perform correctly?
4. is data saved into the correct files?
5. does the system produce the correct results?

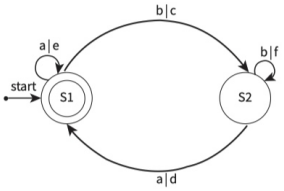
Finite state machine (FSM): a machine that consists of a fixed set of possible states with a set of inputs that change the state and a set of possible outputs
State-transition table: a table that gives information about the states of an FSM
State-transition diagram: a diagram that describes the behaviour of an FSM

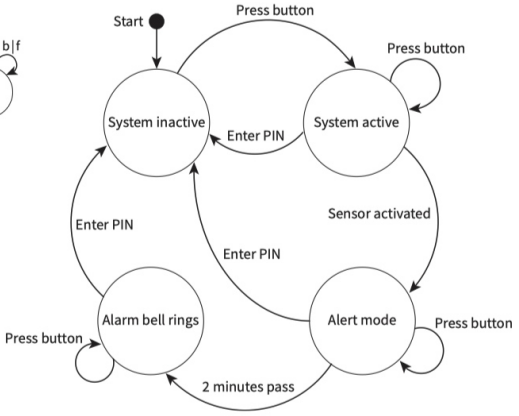
state-transition diagram



		current state	
		S1	S1
input	a	S1	S1
	b	S2	S2

state-transition diagram with outputs





Current state	Event	Next state
System inactive	Press start button	System active
System active	Enter PIN	System inactive
System active	Activate sensor	Alter mode
System active	Press start button	System active
Alert mode	Enter PIN	System inactive
Alert mode	2 minutes pass	Alarm bell ringing
Alert mode	Press start button	Alert mode
Alarm bell ringing	Enter PIN	System inactive
Alarm bell ringing	press start button	Alarm bell ringing

Syntax error: an error in which a program statement does not follow the rules of the language
Logic error: an error in the logic of the solution that causes it not to behave as intended
Run-time error: an error that causes program execution to crash or freeze
Why errors occur and how to find them:
1. the programmer has made a coding mistake
2. the requirement specification was not drawn up correctly
3. the soft ware designer has made a design error
4. the user interface is poorly designed, and the user makes mistakes
5. computer hardware experiences failure.
Test data: carefully chosen values that will test a program
Black-box testing: comparing expected results with actual results when a program is run
White-box testing: testing every path through the program code
Dry-run (walk through): the process of checking the execution of an algorithm or program by recording variable values in a trace table
Trace table: a table with a column for each variable that records their changing values
Integration testing: individually tested modules are joined into one program and tested to ensure the modules interact correctly
Alpha testing: testing of software in-house by dedicated testers
Acceptance testing: testing of software by customers before sign-off
Beta testing: testing of software by a limited number of chosen users before general release