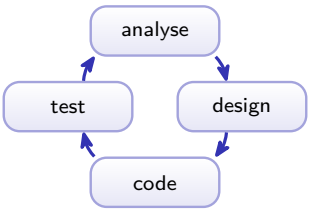
Software Development

AS Computer Science ▪ Topic 12

A-Level Computer Science

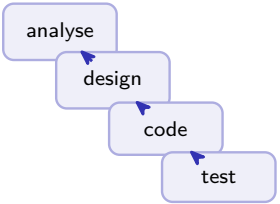


What we will cover

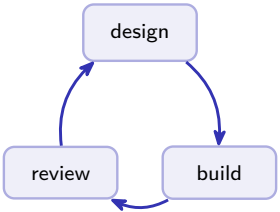
1. Development life cycle
2. Design tools
3. Errors
4. Testing
5. Maintenance
6. Recap

1 Life cycle

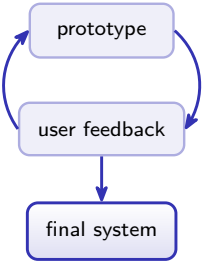
Development life-cycle models



waterfall: linear



iterative: repeated passes



RAD: quick prototypes

Waterfall 瀑布模型: stable needs. **Iterative** 迭代模型/**RAD** 快速应用开发/**Agile** 敏捷: needs discovered or changing.

Program development life cycle

Tabel Simbol-simbol *flowchart* program

No.	Simbol	Keterangan
1		**Kotak Terminal (*Terminal Symbol/Terminator*)** ↘ simbol dimulainya (Start/Begin) dan diakhirinya suatu program (Finish/End).
2		**Kotak Proses (*Processing Symbol*)** ↘ tempat untuk menuliskan perintah/operasi aritmatika.
3		**Kotak Input dan Output (*Input/Output Symbol*)** ↘ tempat untuk memberikan masukan (*input*) atau menampilkan keluaran (*output*) dari suatu operasi.
4		**Kotak Keputusan (*Decision Symbol*)** ↘ tempat pertanyaan/pengujian kondisi, berisikan operasi perbandingan logika, dengan dua nilai keluaran (YA/BENAR/*TRUE* dan TIDAK/SALAH/*FALSE*).
5		**Lingkaran Penyambung (*Connector Symbol*)** ↘ digunakan jika diagram alir belum selesai dan akan berlanjut ke sampingnya pada halaman yang sama.
6		***Off-page Connector Symbol*** ↘ digunakan jika diagram alir belum selesai dan akan berlanjut ke halaman yang berbeda.
7		***Looping Symbol/Preparation Symbol*** ↘ digunakan untuk memberikan nilai awal pada suatu variabel/*counter* dan menggambarkan proses yang perulangan.
8		**Arah Proses (*Direction*)** ↘ untuk menunjukkan arah aliran proses program.

A flowchart plans a program's logic during the design stage of the cycle

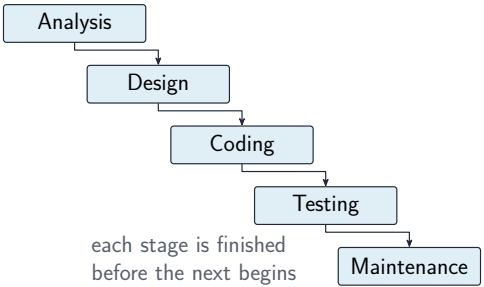
Why there are different ones

No single life cycle fits every project, so several **development life cycles** exist.

> The choice depends on the size and complexity, how clear the **requirements** 需求 are at the start, how much change is expected, the risk level, the team, and the deadline.

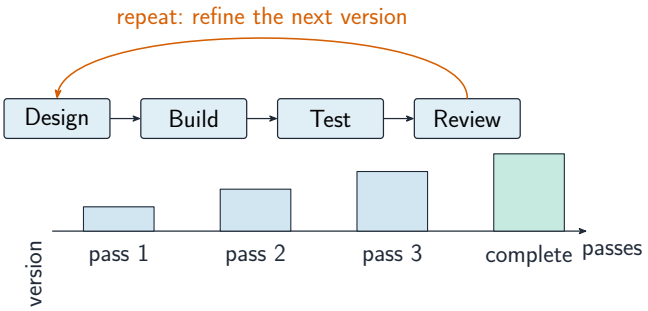
The waterfall model

A linear sequence, each stage finished before the next. Clear and well-documented, good for stable requirements – but poor at mid-project change, and the customer sees nothing working until the end.



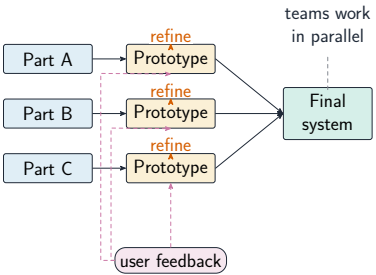
The iterative model

Repeated passes, each producing a partial version that is reviewed and refined. Catches problems earlier and suits requirements discovered over time – but is harder to estimate.

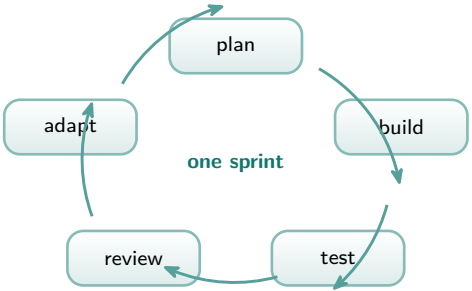


Rapid application development

Heavy use of a **prototype** 原型 and user feedback. Very fast first delivery and good for changing requirements – but it depends on users being available, and suits smaller systems.



Agile: short sprints, constant contact

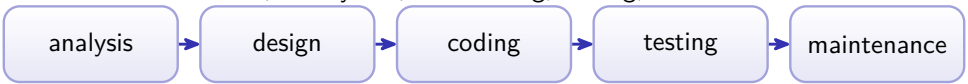


Agile 敏捷 runs short iterations – **sprints** – with constant collaboration and testing throughout.

Flexible and adaptive, but it needs a committed customer and a skilled team. No model is best: choose on size, how clear the **requirements** 需求 are, how much change is expected, the risk, the team and the deadline.

The standard stages

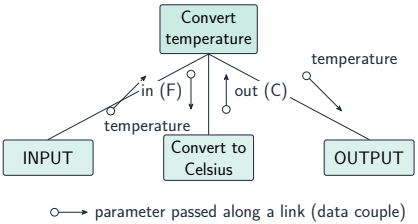
Analysis decides *what* is needed; **design** decides *how* – data structures, algorithms, modules, interface, file layouts; then coding, testing, maintenance.



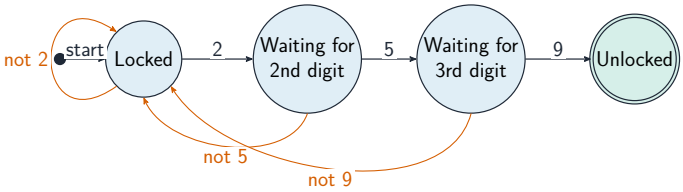
> analysis = what; design = how; then code, test and maintain.

2 Design tools

Design tools



A **structure chart** 结构图 shows the **hierarchical decomposition** 分解 of a program into modules (**subroutines** 子程序) and the **parameters** 参数 passed between them. A design-stage tool – you can read the procedure **signatures** 签名 straight off it.



State-transition diagram

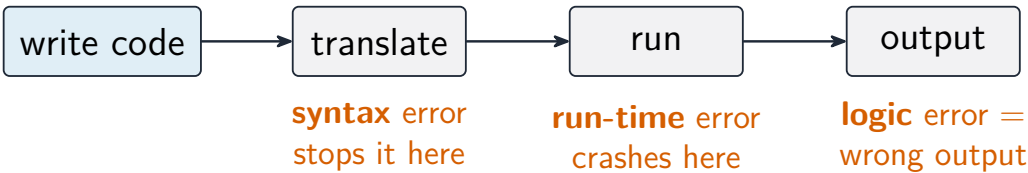
A **state-transition diagram** 状态转换图 shows the **states** 状态 a system can be in and the **events** that move it between them – good for vending machines, traffic lights.

> **State-transition diagrams** are used to document the behaviour of an algorithm or system.

3 Errors and testing

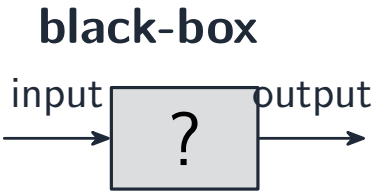
Three kinds of error

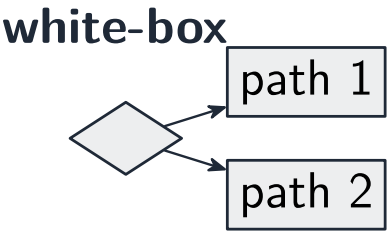
- a **syntax error** 语法错误 breaks the language's grammar – caught at translation, and the program will not run until it is fixed
- a **run-time error** 运行时错误 happens while running (divide by zero, file not found, index out of range): it crashes or raises an exception
- a **logic error** 逻辑错误 runs perfectly and gives the wrong output – find it with a **dry run** 手工跟踪 or a trace table



Testing methods

- **black-box testing** 黑盒测试 – from the spec, ignoring the code
- **white-box testing** 白盒测试 – every code path
- a **unit test** 单元测试 checks one routine, with a **stub** 桩程序 standing in for anything it calls; **integration testing** 集成测试 then checks them together
- **alpha** α 测试 (in-house), **beta** β 测试 (real users), **acceptance testing** 验收测试 (the customer signs it off)





Test strategy vs test plan

test strategy 测试策略

the high-level approach:
which kinds of testing,
who runs them, and when

test plan 测试计划

the detailed list of tests:
input data, expected output,
actual output

The plan's "expected vs actual" columns are exactly how logic errors get caught.

Choosing test data, and the stages of the life cycle

normal data	typical values the program should accept
boundary data 边界数据	the values at the edge of the valid range, and just outside it
extreme data 极端数据	the largest and smallest valid values
abnormal data	values that should be rejected – wrong type, out of range, empty
a stub 桩	a placeholder for a module that does not exist yet, so the structure can be tested top-down
a walkthrough 走查	reading the code aloud against a trace table, before running it at all

A life cycle manages complexity, coordinates teams, tracks progress with milestones, and makes the work reviewable. Naming why it exists is the essay question.

Choosing test data

Four kinds of test data for a 0–100 field:

normal (50, 75) inside; extremes (0, 100) the accepted edges; abnormal (-1, 101, -10, 200) rejected

normal 正常数据

50, 75
well inside

extreme 极端数据

0, 100
edges, still accepted

abnormal 异常数据

-10, 200
must be rejected

boundary 边界数据

$0/-1$, $100/101$
both sides of the line

Worked example: catch the logic error

"Award a pass for marks of 50 or above":

```
INPUT Mark
IF Mark > 50 THEN
  OUTPUT ``Pass''
ELSE
  OUTPUT ``Fail''
ENDIF
```

the bug: > should be >=

Mark	expected	actual	
75	Pass	Pass	✓
30	Fail	Fail	✓
50	Pass	**Fail**	**X**

Only the boundary value 50 exposes it – the normal values both pass.

A logic error runs without crashing – compare expected vs actual output, and always test the exact boundary.

4 Maintenance

Maintenance and amending code

perfective
improve features or speed

adaptive
keep it working in a changed environment

corrective
fix bugs found in use

- **perfective maintenance** 完善性维护 – improve it
- **adaptive maintenance** 适应性维护 – a new environment
- **corrective maintenance** 纠正性维护 – fix bugs

Amending a program: read it, make the smallest change, update callers, then

> **regression test** 回归测试: check the new behaviour *and* that you broke nothing old.

The three kinds of maintenance, and two design tools

corrective	fixing faults found after release
adaptive	changing the program to suit a new environment – a new OS, a new law, a new data format
perfective 完善性维护	improving performance or features even though it works
which dominates	maintenance is most of a program's total cost, and perfective is the largest share of it
a structure chart 结构图	shows the **hierarchical decomposition** 分解 of a program into modules, with the data passed between them
a state-transition diagram	makes missing transitions easy to spot – "what if a second coin is inserted while awaiting selection?"

That question about the second coin is the point of the diagram. It surfaces the cases nobody thought of, which is where most real faults live.

Worked example – four kinds of test data

A field accepts an exam mark from 0 to 100. Give test data of each kind, with its expected result.

- **Normal**: 50 – accepted, a typical value inside the range.
- **Abnormal**: −10, 200, äbč – rejected, out of range or the wrong type.
- **Extreme**: 0 and 100 – the largest and smallest values still *accepted*.
- **Boundary**: the pairs either side of each edge – −1 rejected beside 0 accepted.

Every value must carry its **expected result**, or the test plan proves nothing. An extreme value sits inside and is accepted; a boundary pair sits either side of the edge.

Exam tips

- Compare development models (waterfall, iterative, RAD) and know the stages of the **program development life cycle**.
- Distinguish **syntax, logic and run-time** errors and how each is found.
- Choose **test data** of three kinds – normal, boundary and erroneous – and give an example of each for the stated range.
- Distinguish the types of **maintenance** (corrective, adaptive, perfective).

5 Recap

Topic 12 – key takeaways

1 Life cycle

waterfall vs iterative/RAD/agile

2 Errors

syntax, run-time, logic

3 Testing

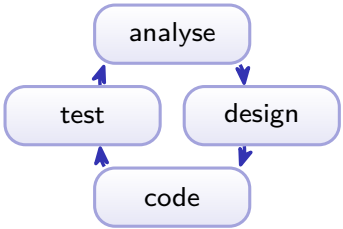
black/white box; normal, extreme, abnormal data

4 Maintenance

perfective, adaptive, corrective; regression test

Check yourself

1. Which error gives wrong output but does not crash?
2. Which testing uses only the specification?
3. Test data 0 and 100 for a 0–100 field – which kind?
4. What is regression testing?



Answers **1.** a logic error **2.** black-box **3.** extreme **4.** re-test old features still work