

Topic 12: Software Development

How professional teams manage code, automate testing and keep complexity under control.

Optional stretch material. This goes beyond the 9618 syllabus, so you will not be asked about it directly in the exam. Its purpose is to deepen ideas you already know. **Builds on:** 12.1 Program development life cycle, 12.3 Program testing and maintenance. Try the challenge questions on the last pages before you read the answers.

Version control and branching

Version control (such as Git) records every change to a project as a **commit**, so you can see who changed what, compare versions and roll back mistakes. Its big idea is the **branch**: a separate line of development.

```
git switch -c new-feature    # create a branch and move to it
git add .                  # stage changes
git commit -m "Add login page"
git switch main
git merge new-feature       # combine the work
```

Teams usually work on branches and use a **pull request** so a colleague reviews the code before it is merged. If two people edit the same lines, Git reports a **merge conflict** and a human decides which version to keep.

Automated testing, CI/CD and TDD

Testing by hand does not scale. Teams write automated tests: many fast **unit tests** (one function each), fewer **integration tests** (components together) and a few **end-to-end tests** (whole system through the interface). This is called the *test pyramid*.

Continuous integration (CI) runs the whole test suite automatically every time code is pushed, catching breakages within minutes. **Continuous delivery/deployment (CD)** goes on to package and release passing code automatically.

In **test-driven development (TDD)** you write a failing test first (*red*), write just enough code to pass (*green*), then tidy the code without changing behaviour (*refactor*).

Technical debt and complexity

Taking a shortcut (copy-pasted code, no tests, unclear names) is like borrowing: you ship sooner but pay "interest" later as every change gets slower and riskier. This is **technical debt**. **Refactoring** and code review pay it down.

Cyclomatic complexity measures how tangled a function is: count the decision points (IF, ELSE IF, loops, each case) and add 1. It is also the number of independent paths, hence the minimum number of test cases needed to cover them. Teams set limits (often around 10) and split functions that exceed them.

Challenge questions

1. Two developers change the same line of a file on different branches. What happens when the second branch is merged, and how is it resolved?
2. A function contains one IF, one ELSE IF and one WHILE loop (no other decisions). What is its cyclomatic complexity, and what does it tell you about testing?
3. Before writing the function `IsLeapYear(Year)`, write four test cases (input and expected output) that cover its rules.

Worked answers

Only look once you have had a proper go at each question.

1. Git cannot decide which change to keep, so it reports a merge conflict and marks both versions in the file.
A developer edits the file to the correct final content (keeping one version or combining both), marks it resolved, tests and commits.
2. 3 decision points + 1 = **4**. There are four independent paths through the function, so at least four test cases are needed to exercise each one.
3. Leap years: divisible by 4, except century years unless divisible by 400.
2024 → TRUE (divisible by 4 only); 2023 → FALSE (not divisible by 4); 1900 → FALSE (century, not divisible by 400); 2000 → TRUE (divisible by 400).
Writing these first (TDD) forces you to think about each rule before coding.