Guidelines

*G1: Write short units of code* (Chapter 2)

Shorter units (that is, methods and constructors) are easier to analyze, test, and

reuse.

Do:

• Limit the length of code units to 15 lines of code

By:

• Not writing units that are longer than 15 lines of

code

• Splitting long units into multiple smaller units

Rationale:

• Easy to understand

• Easy to test

• Easy to reuse

Positive effects on other guidelines:

• Unit Complexity

• Size of codebase

How you can measure it

Count lines of code:

• Every line in the unit that is non-empty and does not contain only comments is a line of

code (15 lines of code)

*G2: Write simple units of code* (Chapter 3)

Units with fewer decision points are easier to analyze and test.

Do:

• Limit the number of branch points to 4

(if, case, ?, &&, II, while, for, catch = conditions)

By:

• Splitting complex units into simpler ones

• Avoiding complex units

Rationale:

• Easy to test

• Easy to understand

• Easy to modify

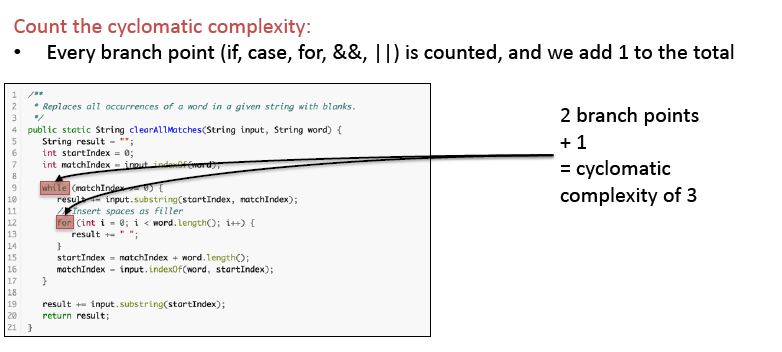
The guideline

Positive effects on other guidelines:

• Unit Size

• Unit Interfacing

How you can measure it

**

*G3: Write code once* (Chapter 4)

Duplication of source code should be avoided at all times, since changes will need to be made in each copy. Duplication is also a source of regression bugs.

*G4: Keep unit interfaces small* (Chapter 5)

Units (methods and constructors) with fewer parameters are easier to test and

reuse.

*G5: Separate concerns in modules* (Chapter 6)

Modules (classes) that are loosely coupled are easier to modify and lead to a more

modular system.

*G6: Couple architecture components loosely* (Chapter 7)

Top-level components of a system that are more loosely coupled are easier to

modify and lead to a more modular system.

*G7: Keep architecture components balanced* (Chapter 8)

A well-balanced architecture, with not too many and not too few components, of

uniform size, is the most modular and enables easy modification through reparation of concerns.

*G8: Keep your codebase small* (Chapter 9)

A large system is difficult to maintain, because more code needs to be analyzed,

changed, and tested. Also, maintenance productivity *per line of code* is lower in a

large system than in a small system.

*G9: Automate development pipeline and tests* (Chapter 10)

Automated tests (that is, tests that can be executed without manual intervention)

enable near-instantaneous feedback on the effectiveness of modifications. Manual tests do not scale.

*G10: Write clean code* (Chapter 11)

Having irrelevant artifacts such as TODOs and dead code in your codebase

makes it more difficult for new team members to become productive. Therefore,

it makes maintenance less efficient.

Do:

• Write clean code

By:

• Avoiding code smells

• Not leaving smelly code behind

Because clean code:

• Is more maintainable

• Is less confusing than unhygienic code

Seven ‘Boy Scout rules’

How-to

1. Leave no **unit level code smells** behind

2. Leave no **bad comments** behind

3. Leave no **code in comments** behind

4. Leave no **dead code** behind

5. Leave no **long identifiers** behind

6. Leave no **magic constants** behind

7. Leave no **badly handled exceptions** behind

1. Measure your units and refactor where needed

2. Delete worthless comments

3. Delete code in comments and revert when needed

4. Delete unreachable code, and unused private methods

5. Use conventions, and limit the length of identifiers

6. Extract magic constants to a global context

7. Catch all exceptions, catch specific exception, and

translate specific exceptions to general messages for the

end-user