**Chapter 3 Functions**

**Small!**

* They should be small
* Functions should not be 100 lines long.

**Blocks and Indenting**

* The blocks within if statements, else statements, while statements and so on should be one line long.
* Should not be large enough to hold nested structures.
* Indent level should not be greater than one or two

**Do One Thing**

* Functions should do one thing, they should do it well and they should do it only
* It is not doing one thing when you can still create another function from it

**Sections within Functions**

* Functions that do one thing cannot be reasonably divided into sections

**One Level of Abstraction per Function**

* Mixing levels of abstraction within a function is always confusing

**Reading Code from Top to Bottom: The Stepdown Rule**

* Read the program descending one level of abstraction at a time as we read down the list of functions
* Learning this trick very important
* It is the key to keeping functions short and do one thing

**Use Descriptive Names**

* You know you are working on clean code when each routine turns out to be pretty much what you expected
* The smaller the focused of a function the easier it is to choose descriptive name
* Use naming convention that give the function a name that says what it does
* Be consistent in your names

**Function Arguments**

* Ideal number of arguments for a function is zero
* More than three requires very special justification
* Arguments are even harder from a testing point of view
* With more than two arguments, testing every combination of appropriate values can be daunting
* Output arguments often cause us to do a double- take

**Common Monadic Forms**

* Asking a question or operating on that argument
* Calling an event
* Should be clear to the reader that this is an event
* Output argument instead of return value for a transformation is confusing

**Flag Arguments**

* Flag arguments are ugly
* Passing a Boolean into a function is terrible practice

**Dyadic Functions**

* Two arguments is harder to understand than a monadic function
* The parts we ignore are where the bugs will hide.

**Triads**

* Functions that take three arguments are significantly harder to understand than dyads

**Argument Objects**

* Likely to be wrapped into a class of their own
* Deserve a name of its own

**Verbs and Keywords**

* Choosing good names for a function can go a long way
* Function and argument should form a very nice verb/noun pair

**Have No Side Effects**

* Side effects are lies
* Creates a temporal coupling

**Output Arguments**

* Most naturally interpreted as inputs to a function
* Anything that forces you to check the function signature is equivalent to a double-take
* Output arguments should be avoided

**Command Query Separation**

* Functions should either do something or answer something but not both

**Prefer Exceptions to Returning Error Codes**

* Returning error codes from command functions is a subtle violation of command query separation.

**Error Handling is One thing**

* A function that handles error should do nothing else

**Don’t Repeat Yourself**

* The duplication is a problem because it bloats the code
* Duplication may be the root of all evil in software

**Structured Programming**

* Should have one entry and one exit
* Every system is built from a domain-specific language
* Functions are the verbs of the language, classes are the nouns
* Master programmers think of systems as stories to be told

**Chapter 4 Comments**

* Nothing can be quite helpful than a well-placed comment
* Comments are necessary evil
* Compensate for our failure to express ourself in code
* Inaccurate comments are far worse than no comments at all

**Comments Do Not Make Up for Bad Code**

* One motivation for writing comments is bad code
* Clean your mess rather than spending time writing comments

**Explain Yourself in Code**

**Legal Comments**

* Corporate coding standards force us to write certain comments for legal reasons
* Copyright and authorship statements are necessary

**Informative Comments**

* It is sometimes useful to provide basic information with a comment

**Explanation of Intent**

* Comment goes beyond just useful information, it provides the intent behind a decision

**Clarification**

* Translate the meaning of some obscure argument or return value into something that’s readable

**Warning of consequences**

* Sometimes it is used to warn other programmers for certain consequences

**TODO Comments**

* To leave or to do notes

**Bad Comments**

* Whatever else a todo might be, it is not an excuse to leave bad code in the system

**Amplification**

* may be used to amplify the importance of something

**Misleading Comments**

* a programmer makes a statement in his comments that isn’t precise enough to be accurate
* Journals should be completely removed

**Nonlocal Information**

* Make sure it describes the code it appears near

**Inobvious Connection**

* The connection between a coment and a code should be obvious
* It is a pity when a comment need its own explanation