Code That Fits in Your Head

By Mark Seemann

Art or Science ?

Are we a :
- Scientist or Artist ?
- Engineer or Craftsman ?
- Gardener or Chef ?
- Poet or an Architect ?

None of the above
And little of all of the above

Follow heuristics

That can be taught

Rules of thumb

Guidelines

Checklists

An **aid** to **memory**
- Help focus on the hard parts
- Taking your mind off the trivial things

Not to constrain
- Should enable / support / liberate

"This book can help transition from programmer to software engineer."

Checklist for starting a new code base

- [x] Use Git
- [x] Automate the build
- [x] Turn on all error messages
  - Treat warnings as errors
  - 0 tolerance for warnings
  - Linter / static code analysis warnings as errors

Tackling Complexity

"Optimise code for readability."

Sustainability
- Much software lives for decades
- A continual effort to make it evolving

Short-Term Memory
- From 4 to 7 pieces of information
- Our brain can't keep track of all

Value
- Code should produce value
- Some code produces no *immediately measurable* value

Should not be prohibited

Readability

When writing code

Context — Clear infos

When reading code

Context is gone...

Organise your code so that the relevant info is activated

Place related code together

WYSIATI
What You See Is All There Is

> *The core problem that software engineering should solve is that it's so complex that it doesn't fit the human brain.*

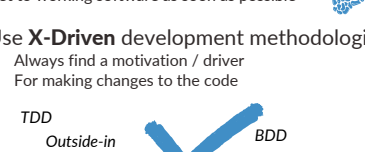
Vertical Slice

"Don't trust yourself to write correct code."

Walking skeleton

Get to working software as soon as possible

Use **X-Driven** development methodologies
- Always find a motivation / driver
- For making changes to the code



Unpolluted Domain Model

Unpolluted by implementation details

Humble Objects
- Objects like "Repository"
- Hard to unit test : depends on a subsystem

Perform smoke test
- Favor automated tests
- Can use cURL for example

Encapsulation

A contract introduces / formalises a level of trust

Transformation Priority Premise
- Use it as a **driver** for changes
- From one working state to another
- Move in small increment
- Driven by tests

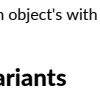
```
{} → nil
nil → constant
...
array → container
...
expression → function
variable → assignment
...
```

Design by contract
- Interact with an object without knowing implementation details
- Enables us to change the implementation (refactor)
- Think of an object in an abstract way
- Replace details of an object's with a simpler contract

Protection of invariants

Guard clause + Postel's Law

Always valid



Object should never be in an invalid state
Not the caller's responsibility

"Be conservative in what you send, be liberal in what you accept"

Science of TDD

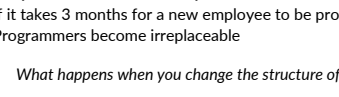
Form a hypothesis
- Prediction of falsifiable outcome
- Perform an experiment
- Measure the result

Compare
The actual to the predicated one

Refactor
Can you improve the code ?

Legacy code and memory
- If it takes 3 months for a new employee to be productive
- Programmers become irreplaceable

What happens when you change the structure of code ?



- Information in Long Term Memory becomes outdated
- Gets harder to work with the code base
- Acquired knowledge no longer applies

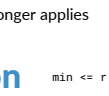
Triangulation

```
min <= r.At <= max
```

Add more test cases

until

Have defeated the Devil



"as the tests get more specific, the code gets more generic"

> *"Be careful editing unit test code; there's no safety net."*

Decomposition

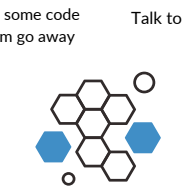
Code rot
- Code gradually becomes more complicated
- If no one pays attention top the overall quality

Thresholds
- Agree on a threshold can help curb code
- Cyclomatic complexity (<8 for ex)

80 / 24 rule
- Stay within a 80x24 character box
- Can help keep method smalls

Establish a culture that actively pays attention to code quality

Code that **fits** in your head



Plot outcome related to a branch in the code

No more than 7 things in a single piece of code

fits in a Hex Flower

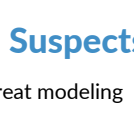
Cohesion

"Things that change at the same rate belong together. Things that change at different rates belong apart - Kent Beck"

Parse don't validate

Instead of "IsValid"

Return a ***Maybe***



Callers will be "forced" to handle **both cases**

"If you can measure the essence of a method in the signature, then that's a good abstraction"

API Design

Affordance : An interface is an affordance
- A set of methods, values, functions , objects
- Enables you to interact with an encapsulated package of code

Poka-Yoke
- Means "mistake-proofing"
- Mistake-proof artefacts and processes

Hierarchy of communication

```
1. Guide the reader by giving APIs distinct types
2. Guide the reader by giving methods helpful names
3. Guide the reader by writing good comments
4. Guide the reader by providing illustrative examples as automated tests
5. Guide the reader by writing helpful commit messages
6. Guide the reader by writing good documentation
```

"Don't say anything with a comment that you can say with a method name. Don't say anything with a method name you can say with a type"

Clear how to use it from its shape

on API

dot-driven development
Degree of discoverability

Command Query Separation

Command
- Methods with **side-effects**
- Should return no data (void)

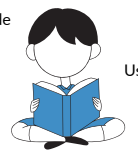
Query
- Methods that do return data
- Should have no side effects

"We can distinguish them without knowing implementation details."

Write code for Readers

Favor well-named code over comments

It may be you



Use types to keep you honest

X-out your code

```
public interface IReservationsRepository
{
    Task Xxxx(Reservation reservation);
    Task<IReadOnlyCollection<Reservation>> Xxx(DateTime dateTime);
    Task<Reservation?> Xxx(Guid id);
}
```

- See if you can still figure out what they do
- Helps you empathize with future readers

Teamwork

Use commit messages
- The best place to explain "why"
- Follow 50/72 rule

Continuous Integration

Integration means merging

Make small changes

Decrease integration risks

Merge as often as you can

it is a practice

Integrate at least every 4 hours

If you can't complete a feature in 4 h

Hide it behind a feature flag

Collective Code Ownership

Bus / Lottery factor
- If a single person 'owns'a part of the code base
- You're vulnerable to team changes

How many team members can be hit by a bus before development halts ?

"Any code changes should involve more than one person"

Pair Programming
Prevent knowledge silos

Mob Programming
Great for knowledge transfer

Rejection is an option

Can introduce latency

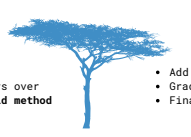
Set aside time for them

Code review
Check wether the code fits in your head

Augmenting code

Strangler Pattern

- Add a **new method**
- Gradually move callers over
- Finally delete the **old method**



- Add a **new class**
- Gradually move callers over
- Finally delete the **old class**

"For any significant change; don't make it in place; make it side-by-side."

Editing Unit Tests

Separate refactoring

Test code — Production code

Failure and trust

Don't trust a test that you haven't seen fail

Rhythm

Personal

Pomodoro technique
- Work in time-boxed intervals
- Like 25 minutes

Take breaks

Use time deliberately
- I start my day with two 25-minute time-boxes
- Where I try to educate myself

Team

Regularly update dependencies
- Dangerous to fall too far behind
- Beginning of each sprint for ex

Schedule other things
- Certificates management
- Database backups
- ...

Troubleshooting

Understanding

Scientific method
- Make a prediction / hypothesis
- Perform the experiment
- Compare outcome to prediction

Simplify
- Consider if removing some code
- Can make the problem go away

Explaining a problem

Write a question on Stack Overflow instead

tends to produce new insight

Rubber ducking
Talk to a rubber duck it will solve your problems

Defects

Reproduce defects as tests

`git bisect`

Identify the commit that caused the problem

Maximum time for a test suite : 10 seconds

Create different "containers" to isolate slow tests

"Abstraction is the elimination of the irrelevant and the amplification of the essential" - Robert C. Martin

Separation of concerns

Functional Core, Imperative Shell

Non deterministic queries / behaviors
- With side effects
- Close to the edge of the system

Complex logic
Write complex logic as **pure functions**

Logging

The more your code is composed from pure functions

The less you need to log

"Log all impure functions, but no more."

The Usual Suspects

STRIDE threat modeling
- Spoofing
- Tampering
- Repudiation
- Info disclosure
- Denial of Service
- Elevation of privilege

Other techniques
- Property-Based Testing
- Behavioral code analysis
- ...