Tidy first?

By
Marco Lorenzo, Carlos Lavilla, Marcos Losada, Diego Martínez

Introduction

Motivation

The book **modernizes "*Structured Design*"**[1], a classic work which established the **fundamental laws** of software engineering. It redefines key concepts like ***cohesion*** and ***coupling***, which have become diluted over time, and **critiques** the purely dogmatic approach of **"*A philosophy of Software Design*"**[2].

Kent Beck

Kent Beck, a **prominent American software engineer,** is renowned for his contributions to **software development methodologies**.
He created **Extreme Programming *(XP)***, was one of the seventeen original signatories of the **Agile Manifesto** in 2001, pioneered **Test-Driven Development *(TDD)***, and co-authored the **JUnit testing framework** for Java.

Three main concepts

"Tidy First?" is the first of a trilogy of books that will talk about the three main concepts in software design for Kent Beck: coupling, cohesion, and tidying.
Beck decided to publish it first since tidying is the smallest skill of design, and he wanted the readers to practise it before the other more complex skills.

Coupling

Two elements are coupled when changing one of them implies changing the other one. For example, a function calling another is coupled with respect to changes of the name, because changing the name of the call implies changing the declaration, and vice-versa. It is important to maintain a low coupling.

Cohesion

Coupling cannot be always eliminated, but cohesion aims to keep all the coupling inside of an element. This way, it is much easier to understand and to maintain when having to make changes. For example, a file is 'cohered' when all the functions are coupling between them. We want to have high cohesion.

Tidying

This is the main concept of the book, and a very common situation for every programmer: we want to make changes in a messy code, but we know that it will be very difficult to understand the code and change it. Then we might decide to make tidings, which are simple changes (such as just changing the name of a function by a more understandable one) that will make it easier for us to understand and change the code in the present and in the future. Tidings are always structural changes; they do never change the behaviour of the application.

---

[1] "*Structured Design*" by *Edward Yourdon* and *Larry L. Constantine*.
[2] "*A philosophy of Software Design*" by *John Osterhout*.

Types of changes

Regarding software there are two main types of changes: behavioural and structural. But we also should acknowledge that changes in software are closely related to type 1 and 2 decisions, as explained in the podcast. Type 1 decisions are those that are irreversible and should be analysed before taking them, whereas type 2 decisions are those that are reversible and therefore should be taken without hesitation.

Structural

Structural changes are those that do not affect how the system behaves but does affect its structure. For example, changing the name of a variable or moving a piece of code around a file. These types of changes are usually reversible (like type 2 decisions).

Behavioural

Behavioural changes are those that affect the functionality of a system. For example, adding a case statement to a switch or modifying an algorithm. These types of changes are irreversible (like type 1 decisions).

Handling Pull Requests

Kent Beck proposes some recommendations about how PRs could be done aiming to not mix the two types of changes as it makes the code more likely to have some mistakes.

1. The first option he proposes is to separate structural and behavioural changes in different PRs.
2. The second option he proposes is to separate the different types of changes in commits within a PR.
3. Lastly, if the previous recommendations were not followed and the developer ends up with a mess of changes but accomplishes the objective, Kent Beck proposes other two options:
    a. Discard it, and then do the changes again applying on of the proposed options.
    b. Discard it and do it again in such a way that it can be explained as an easy path from starting point A to an end point B.

When To Tidy

Tidyings are small, structural changes that improve code without altering behavior. Use them strategically to simplify future changes, reduce coupling, and enhance cohesion. Separate them from behavior changes, focus on actively changing areas, and evaluate cost vs. benefit. Experiment empirically and automate wherever possible for consistent, maintainable code.

Glossary

- **Power law**: Pattern where a small number of elements have a disproportionate impact, guiding developers to focus on the most critical areas.
  “80% of changes in 20% of the code” – Pareto.
- **Empirical software design**: Practice of making design decisions based on real-world data, feedback, and experimentation rather than theory or assumptions.
- **Net present value**: Difference between present value of cash inflows and the present value of cash outflows over a period.
  Used by Beck to explain that revenue sooner equals more revenue later.

References

SE-Radio (2024). SE Radio 615: Kent Beck on “Tidy First?”. From https://se-radio.net/2024/05/se-radio-615-kent-beck-on-tidy-first/

Wikipedia Contributors. (n.d.). *Kent Beck - Wikipedia*. Retrieved 2 24, 2025, from Wikipedia: https://en.wikipedia.org/wiki/Kent_Beck