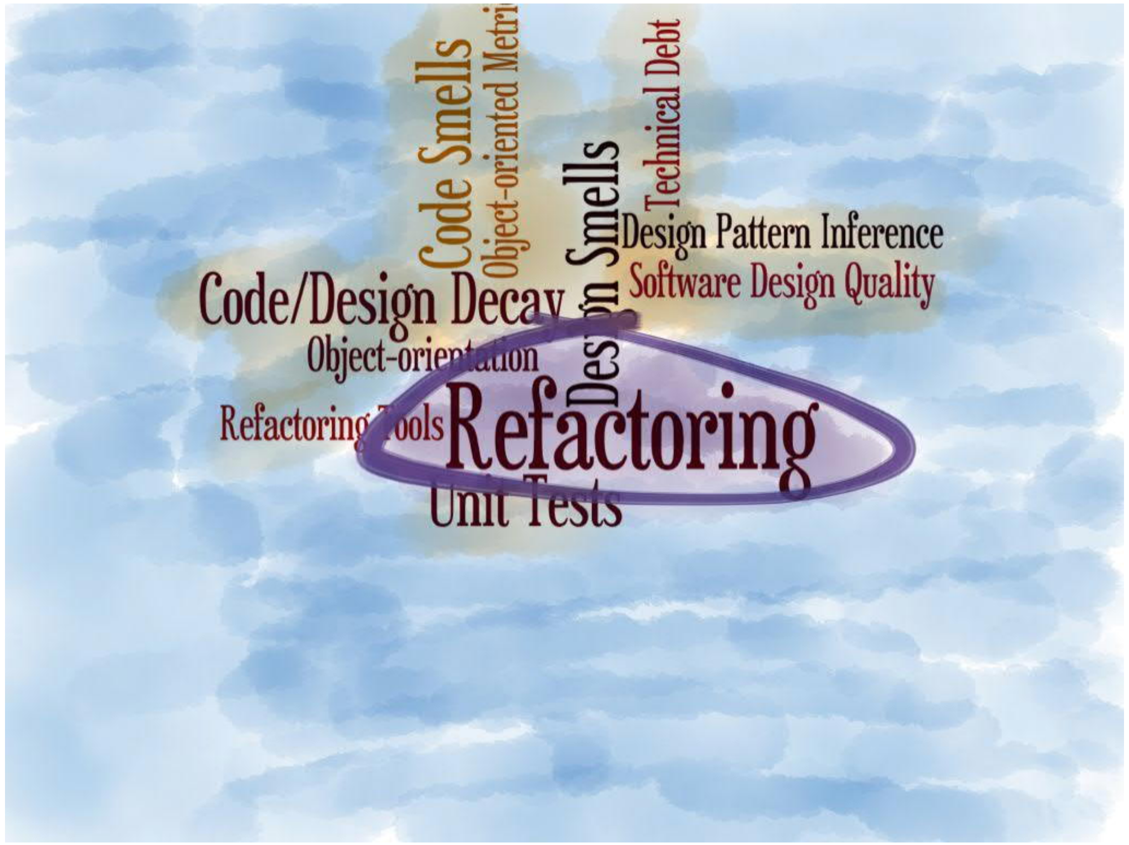
SIEMENS
Healthineers

Introduction to Refactoring

Training Material sourced from CT RDA

Outline

- ❑ Definition
- ❑ Motivation
- ❑ Advantages
- ❑ Steps
- ❑ Classification



Your car

An Example

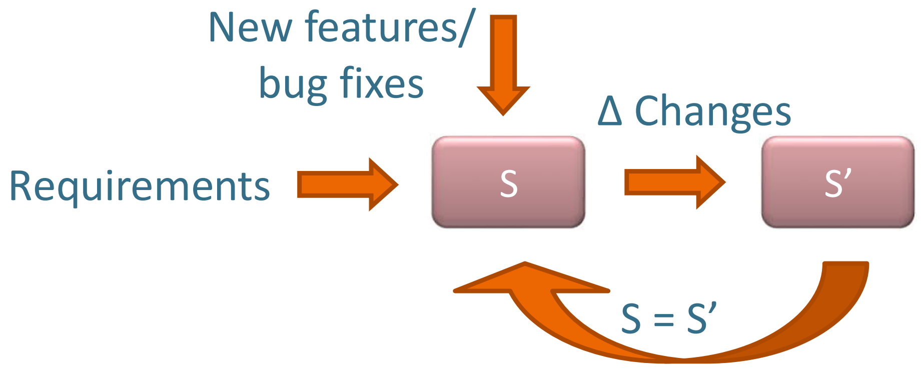
- A software system that comes in three different variants (standard, pro, ultimate) along with trial and release versions.
- Andy was asked to fix a bug and release a new set of variants with both versions.
- The code was filled with conditional compiling statements that make the code messy.
- Even worse, he needs to make changes (such as version number, product name) at huge number of places to release a variant.
- Frustrated and annoyed with the exercise, Andy summarized the list of places where a change is required to release a variant.
- He brought the changes to one file. Thus, changes are required only in one file in order to release a variant/version.
- What did Andy do? Bingo!…. **REFACTORING**!

Definitions

- ❑ Opdyke introduced the term “Refactoring” and defined as “**behavior-preserving program restructuring**”.[Opdyke92]

- ❑ According to Fowler “Refactoring is the process of changing a software system in such a way that it **does not alter the external behavior** of the code, yet **improves its internal structure**.” [Fowler99]

Motivation



- ❑ These Δ changes tend to disturb/distort the design of the software.
- ❑ A developer tends to adopt "**quick fixes**" due to **inexperience** and/or **time constraints**.
- ❑ This results in **code/design decay**.
- ❑ In order to prevent the decaying quality, it is required to refactor the software **periodically**.

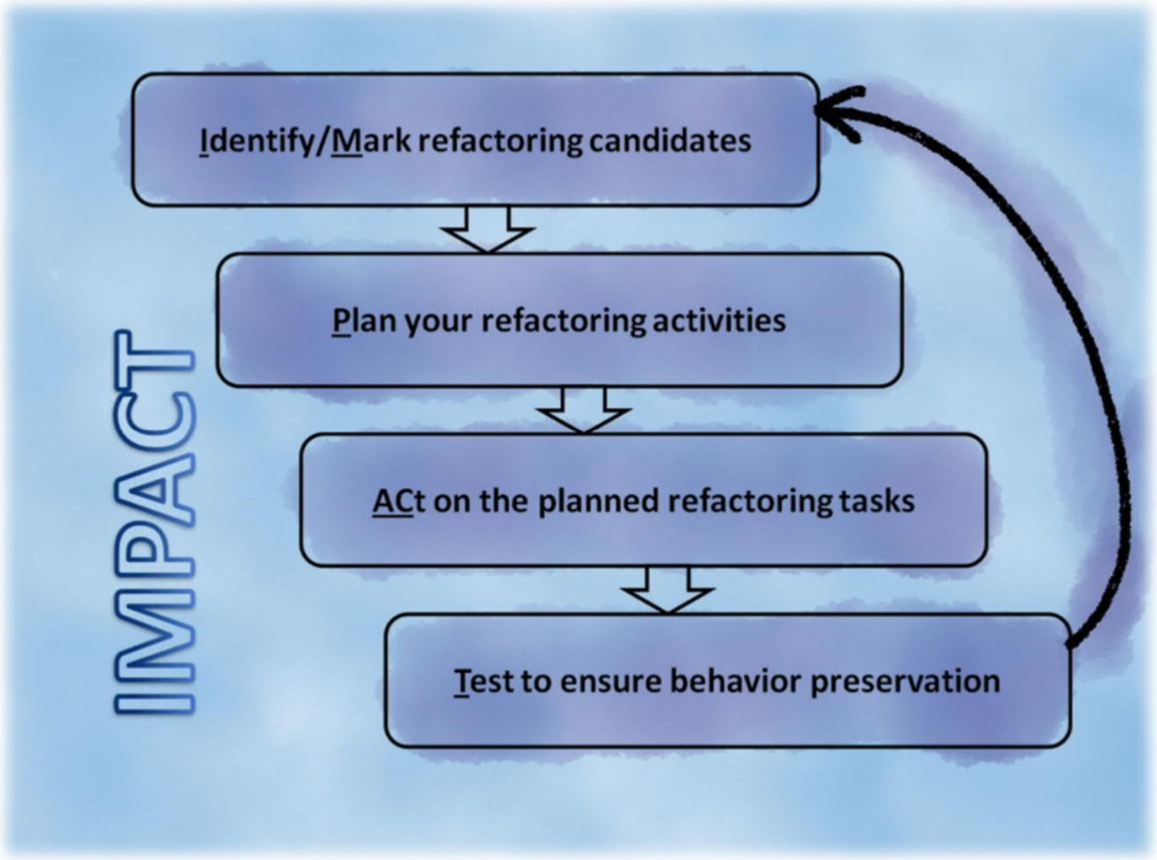
Why to Refactor?

- ❑ Refactoring improves
- ❑ Understandability
- ❑ Extensibility and Flexibility
- ❑ Testability
- ❑ Reusability

- ❑ Refactoring reduces
- ❑ Technical debt

Better code/design quality leads to **improved productivity** as well as **high morale and motivation**

Steps – Refactoring Process Model


IMPACT
Identify/Mark refactoring candidates
Plan your refactoring activities
ACt on the planned refactoring tasks
Test to ensure behavior preservation

Classification

- Classification based on **Operation**
- Atomic Refactoring
- Composite Refactoring

- Classification based on **Abstraction-level**
- Implementation Refactoring
- Design/Architectural Refactoring

When?

- ❑ **When to refactor?**
- ❑ Anytime (when you see a better way of doing a thing)
- ❑ However, you have to know the impact of the change and you need to make sure that the software still works

- ❑ **When not to refactor?**
- ❑ Stable with no changes
- ❑ Unfamiliar with impact

Challenges

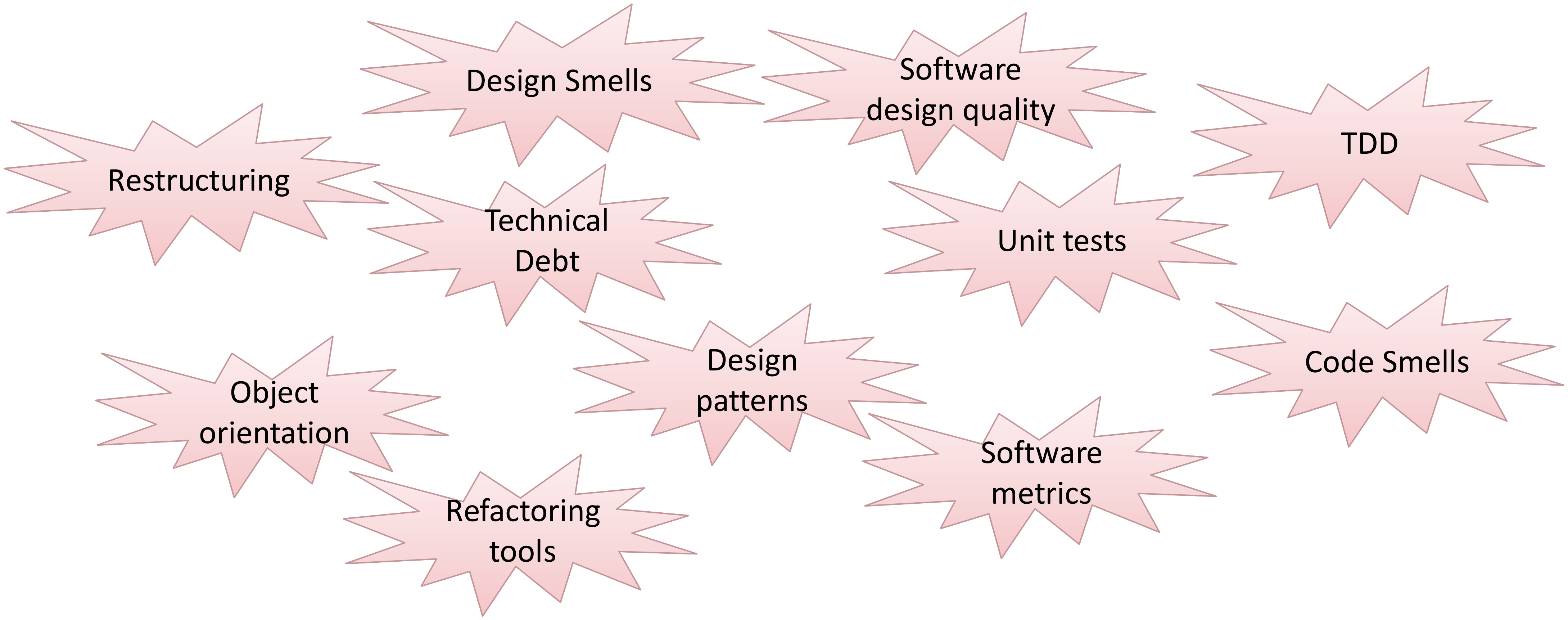
- ❑ Complexity
- ❑ Understandability is prerequisite
- ❑ Changing design is hard
- ❑ Danger of breaking the code
- ❑ Lack of awareness

Despite the challenges

- ❑ The longer you wait before paying your debt, the bigger the bill.
- ❑ The bigger the mess, the less you want to clean it up. (Joshua Keriovsky)

- ❑ The bottom line: “Adopt Refactoring or (technically) bankrupt sooner or much sooner”

The world of Refactoring



References

- **Opdyke92:**
  W. F. Opdyke, Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1992.

- **Fowler99:**
  M. Fowler, Refactoring: Improving the Design of Existing Programs, Addison-Wesley, 1999.

Q &A