Introduction

SIEMENS Healthineers

Advanced FOCUS

Advanced FOCUS – Day 1

	Topics	Trainer	Tentative Schedule
1.	Introduction	Mandar	8.30
	Tea/Coffee Break		10.30
2.	Technical Debt, Design quality, Design principles	Mandar	10.45
	Lunch		12.00
3.	Design principles (contd.)	Mandar	1.15
	Tea/Coffee Break		2.45
4.	Introduction to Refactoring	Mandar	3.00

Advanced FOCUS – Day 2

	Topics	Trainer	Tentative Schedule
5.	Design Smells	Bimalendu	8.30
	Tea/Coffee Break		10.30
6.	Design smells	Bimalendu	10.45
	Lunch		12.00
7.	Design smells (examples)	Bimalendu	1.00
	Tea/Coffee Break		2.30
8.	Design smells (exercise)	Bimalendu	2.45

Advanced FOCUS – Day 3

	Topics	Trainer	Tentative Schedule
9.	Design smells (Exercise)	Bimalendu	8.30
	Tea/Coffee Break		10.15
10.	Design smells (Presentations)	Participants	10.35
	Lunch		12.30
11.	Designing for testability	Shailesh	1.30
	Tea/Coffee Break		3.30
12.	Concluding remarks		3.45

Software Engineering

What is software engineering? How is it different from software development?

What makes software development “difficult”?

- Software is intangible
- Requirements change often
- Software has to be flexible – has to adapt to the environment quickly
- It is not easy to convey ideas and rationale behind decisions easily
- Big projects involve team work, coordination, geographically distributed locations etc.

Software Engineering is all about managing Complexity!!

SIEMENS Healthineers

Design Quality and tech. debt

Training material sourced from CT RDA

Agenda

- Why care about design quality?
- Q-Minimum design guidelines
- Introduction to design principles and design smells
- Design smell examples
- Design metrics computation and smell detection tools
- Conclusion and Q&A

Why care about design quality?

- **Design quality is important**
  - Organizations develop critical, large, and/or reusable software
- **Design quality means:** flexibility, extensibility, understandability, reusability,...
- **Design errors are costly**
  - Capers Jones*: Up to **64%** of software defects can be traced back to error in software design in enterprise software!

* http://sqgne.org/presentations/2012-13/Jones-Sep-2012.pdf

Capers Jones on
design errors in industrial software

INDUSTRY DATA ON DEFECT ORIGINS

Because defect removal is such a major cost element, studying defect origins is a valuable undertaking.

IBM Corporation (MVS)

45%	Design errors
25%	Coding errors
20%	Bad fixes
5%	Documentation errors
5%	Administrative errors
100%	

SPR Corporation (client studies)

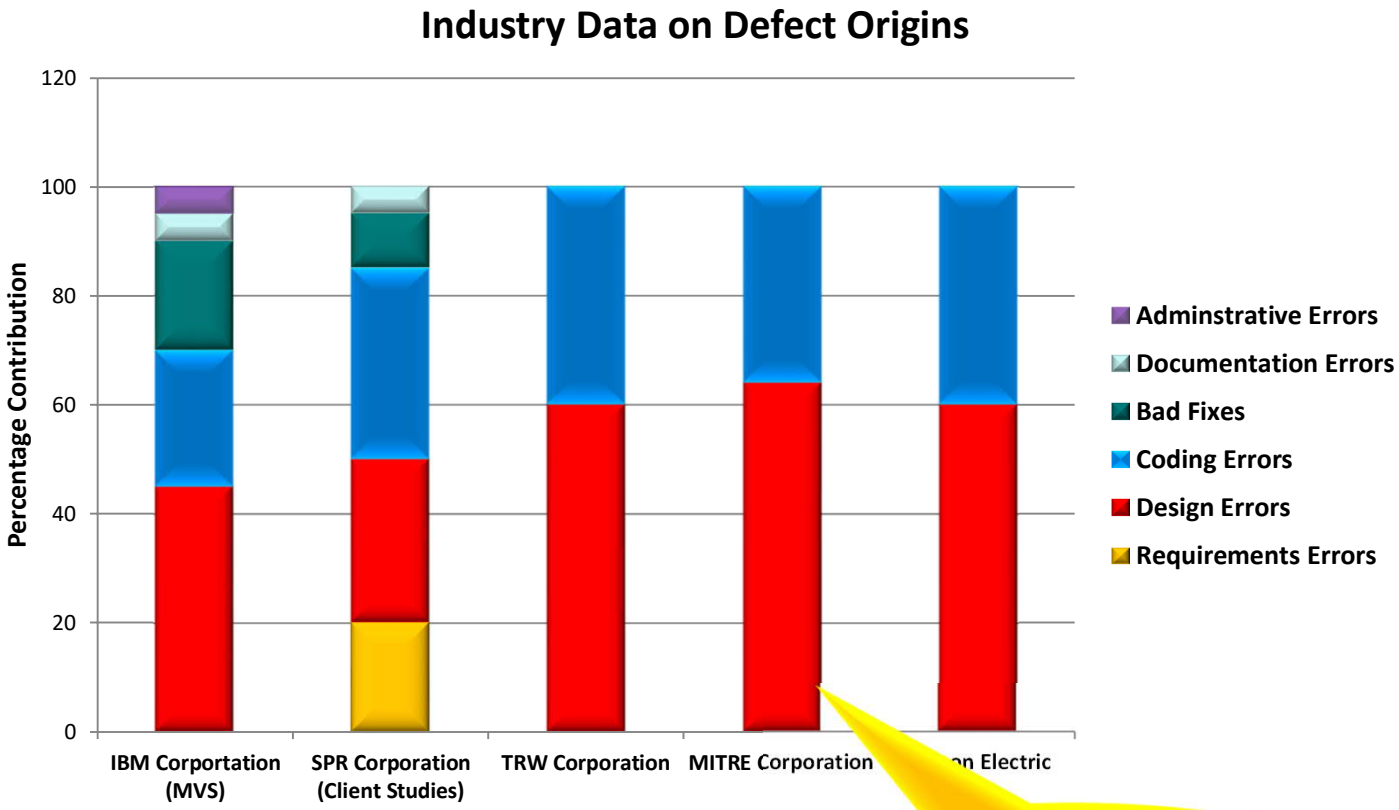
20%	Requirements errors
30%	Design errors
35%	Coding errors
10%	Bad fixes
5%	Documentation errors
100%	

TRW Corporation		MITRE Corporation		Nippon Electric Corp.	
60%	Design errors	64%	Design errors	60%	Design errors
40%	Coding errors	36%	Coding errors	40%	Coding errors
100%		100%		100%	

SWQUAL08W40

* http://sqgne.org/presentations/2012-13/Jones-Sep-2012.pdf

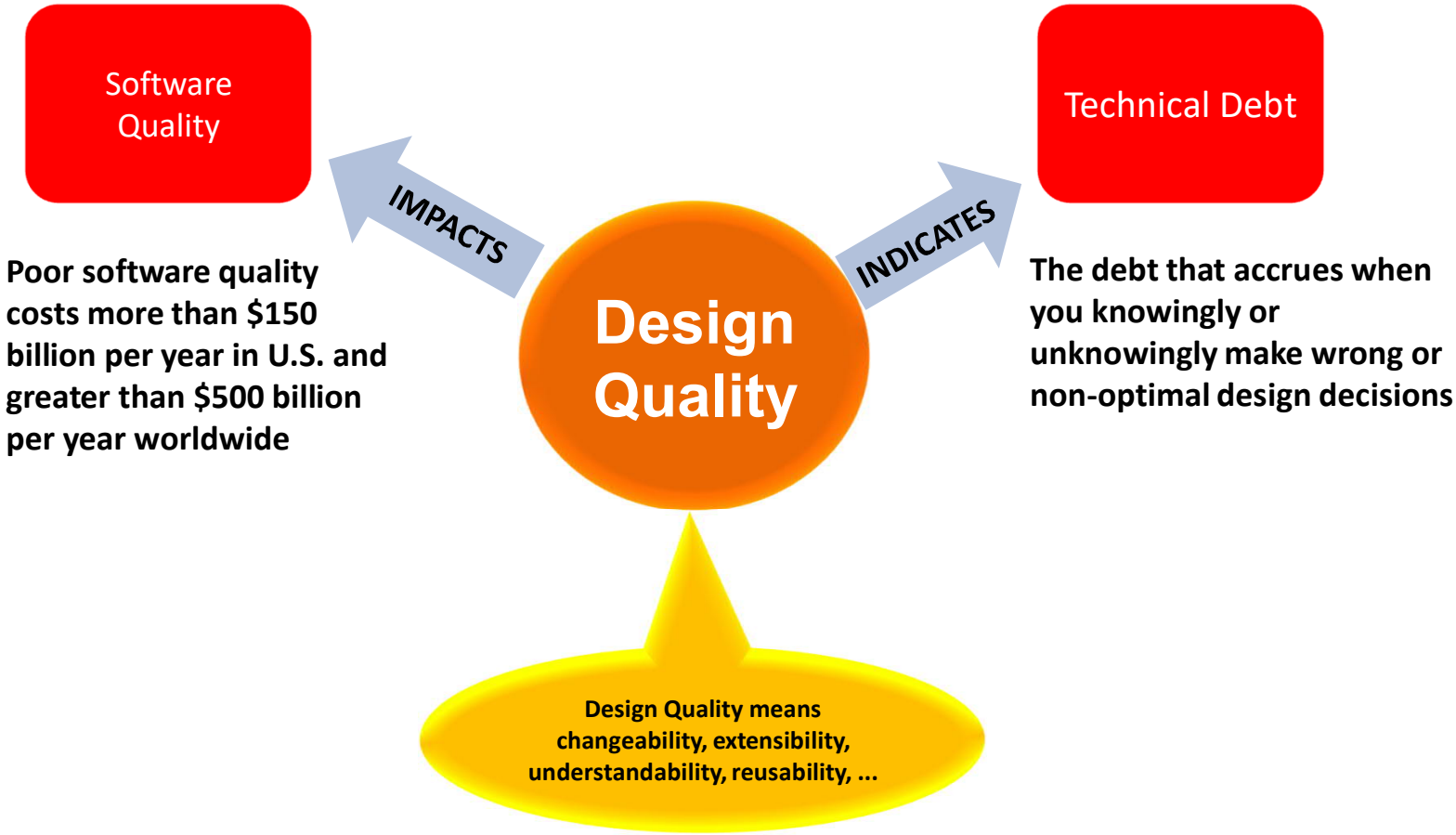
Capers Jones on design errors in industrial software



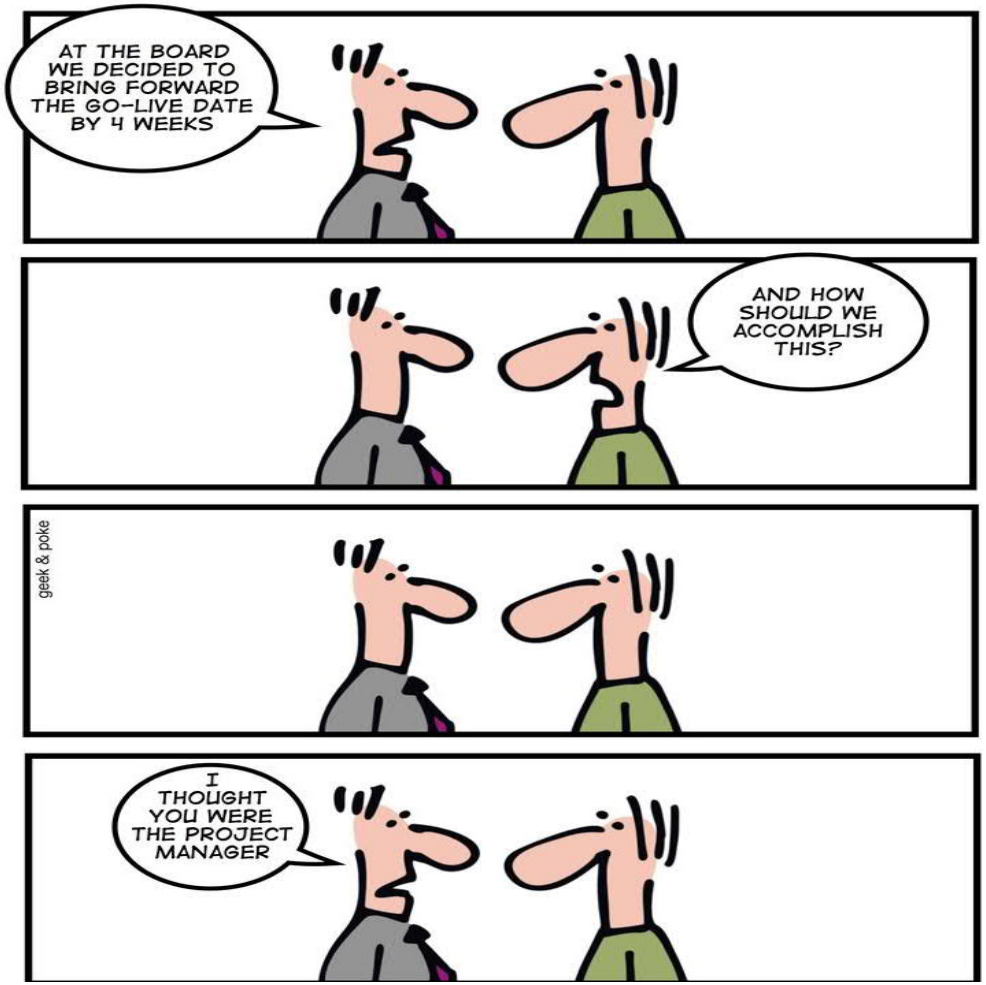
Up to 64% of software defects can be traced back to errors in software design in enterprise software!

* http://sqgne.org/presentations/2012-13/Jones-Sep-2012.pdf

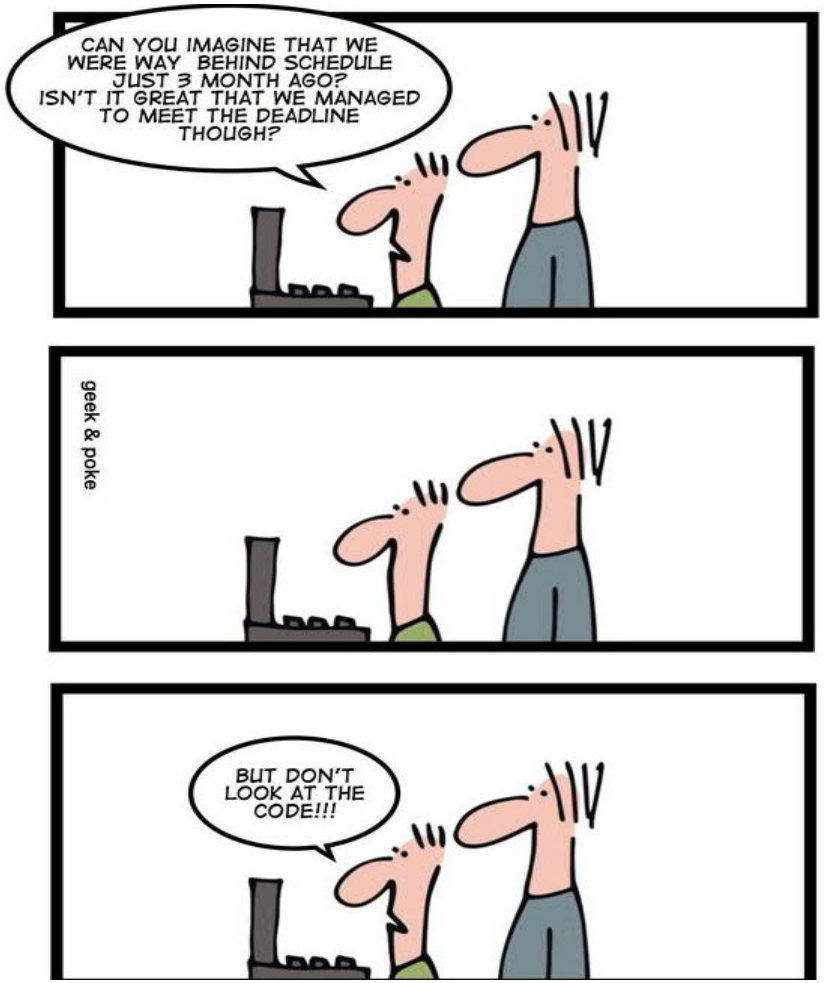
Why care about design quality?



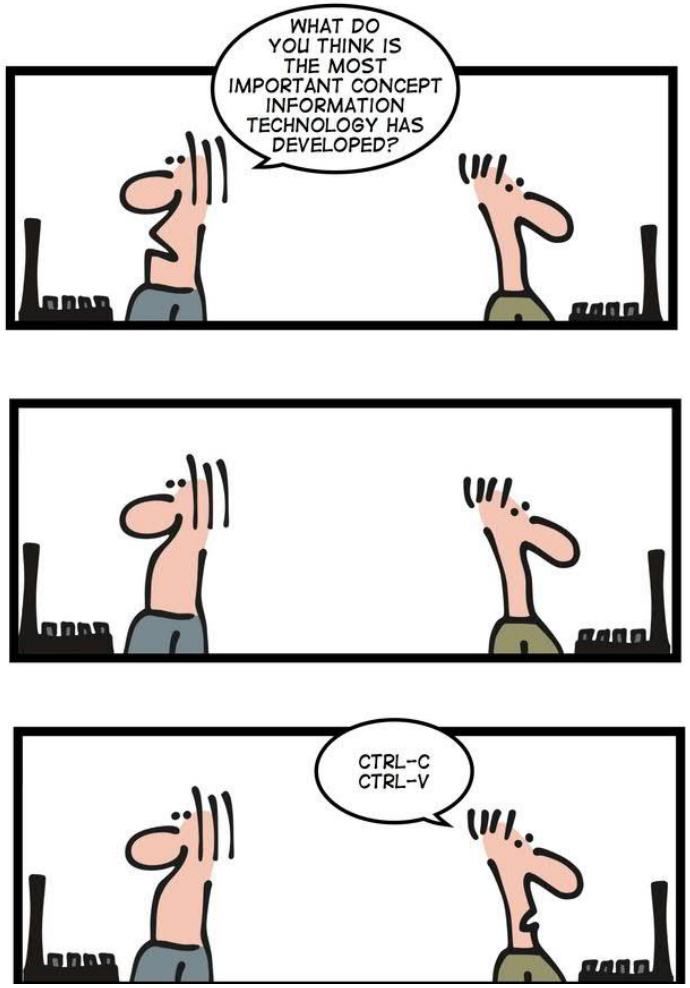
Most common cause of poor design quality



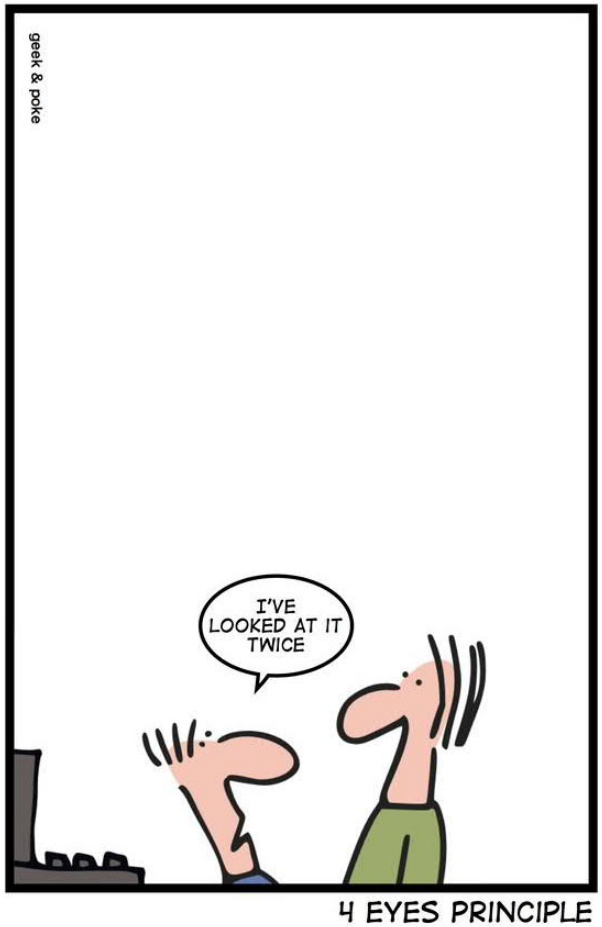
Most common cause of poor design quality



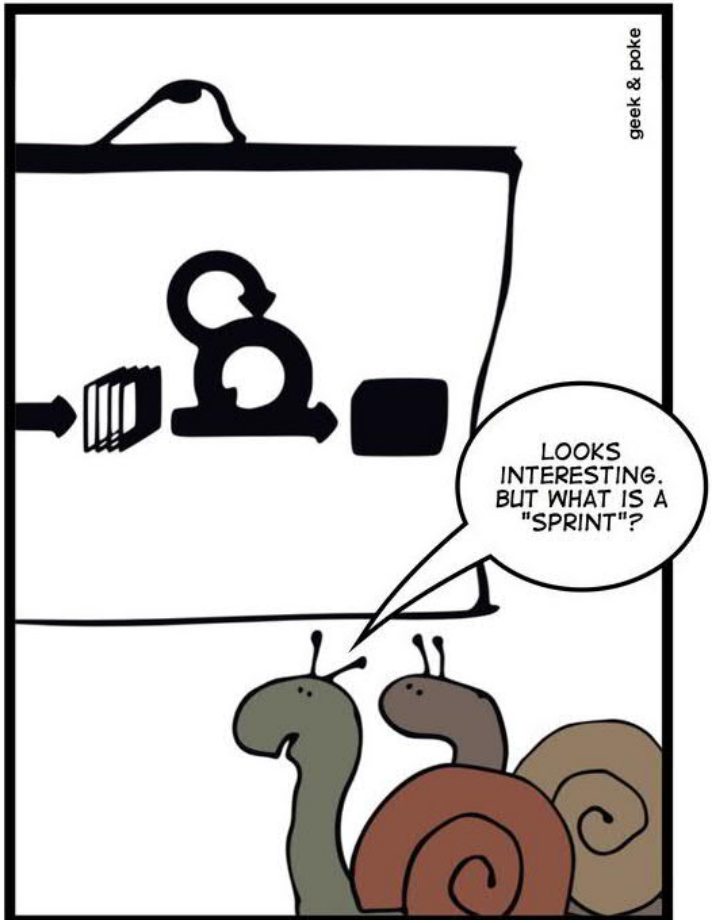
Another “most common” cause



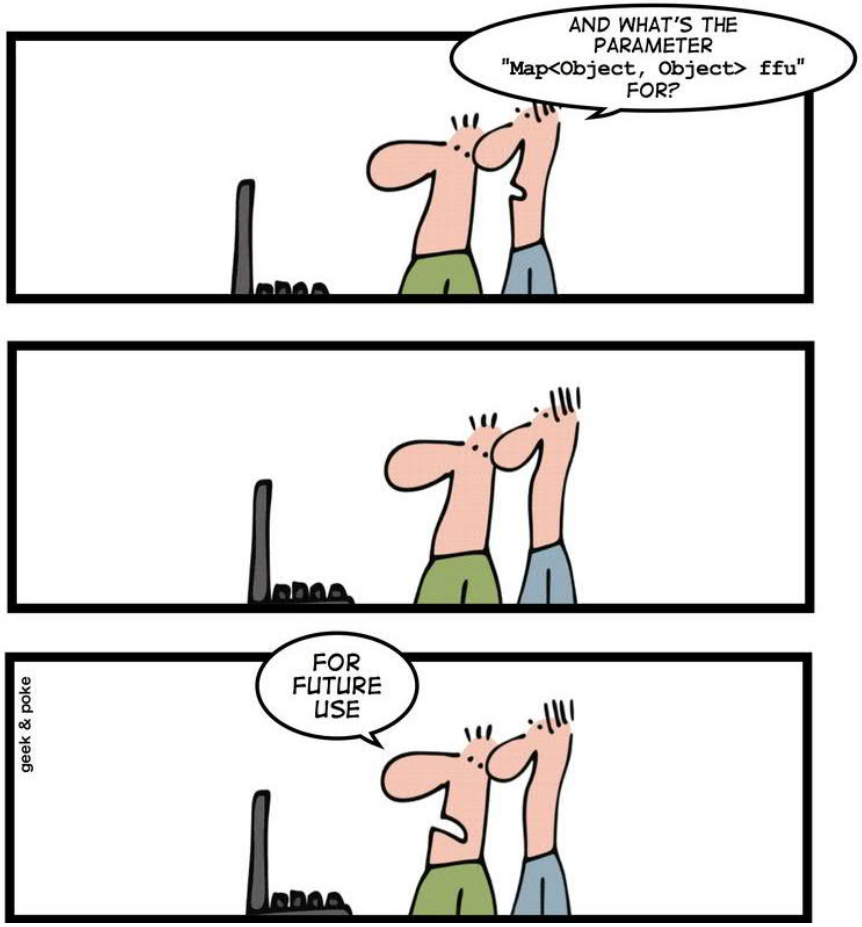
Common causes - not adhering to processes



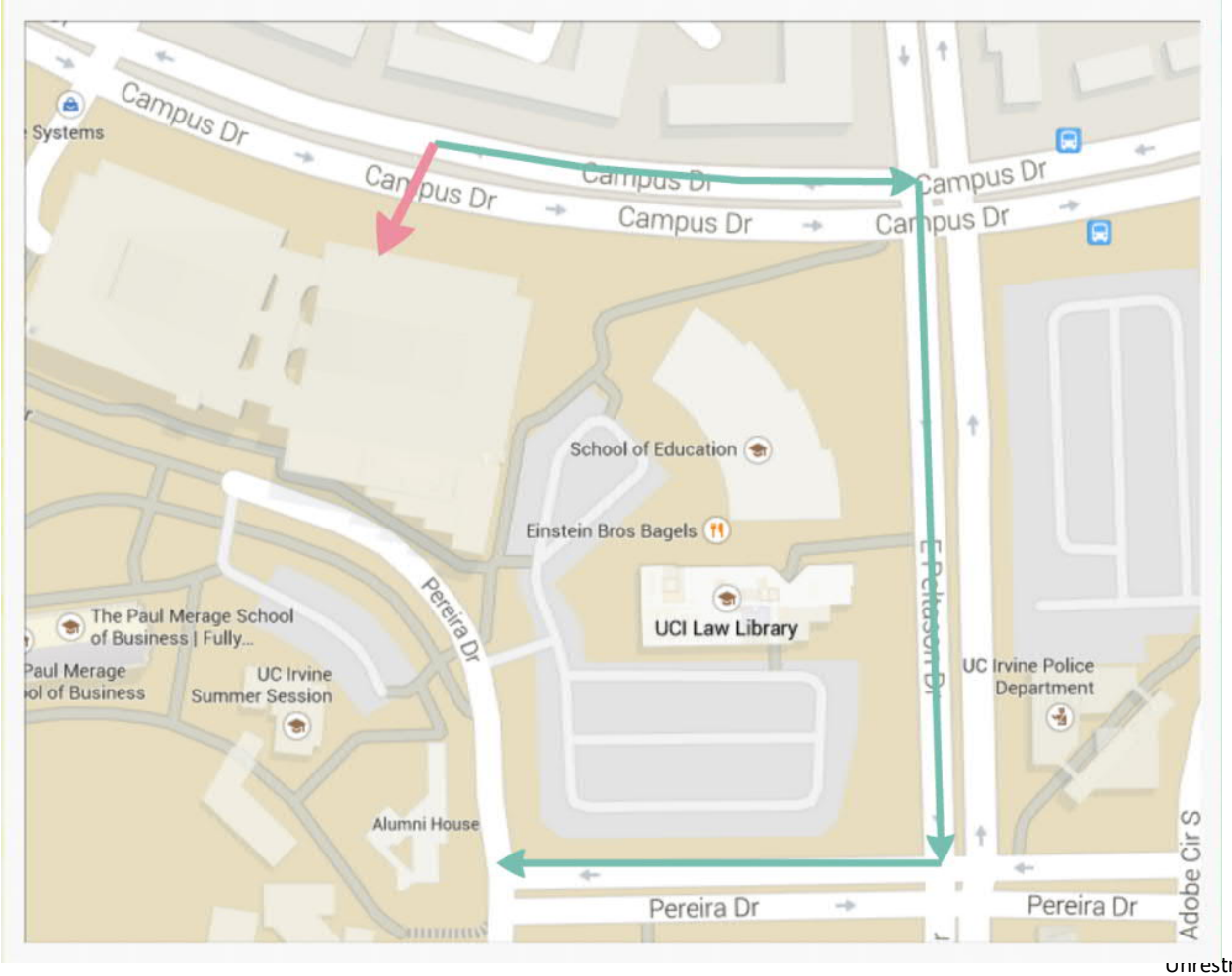
Common causes - Unable to adapt



Common causes - Over-engineering



One-time Violation -> Long-term Habit



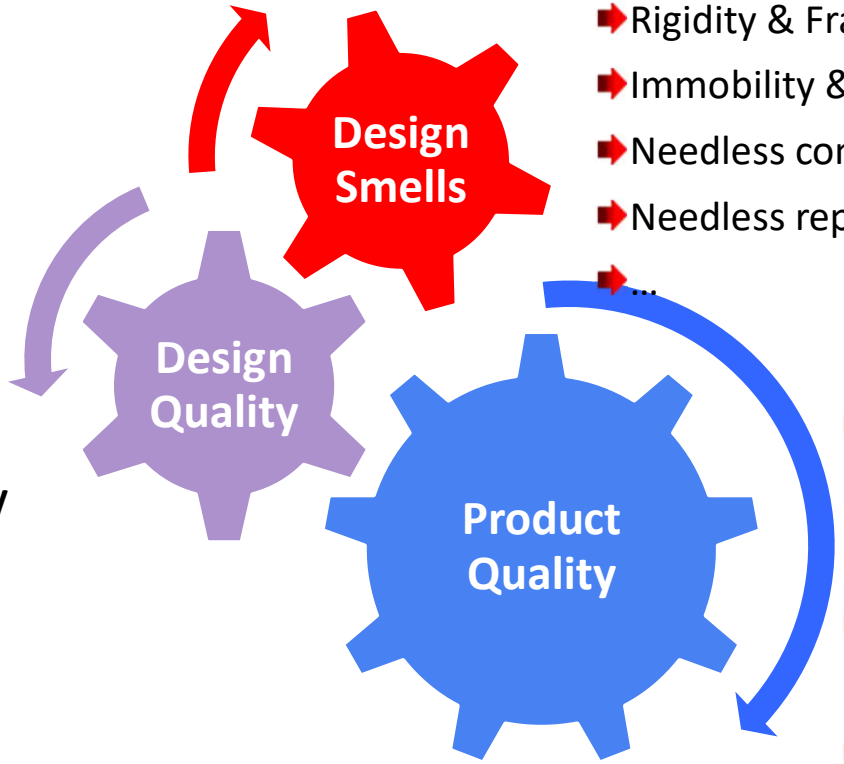
How to achieve good design quality?

- Awareness/training
- Process (eg. QMin)
- Coaching/mentoring (eg. QMin Design)

Why care about smells?

Indicators

- Rigidity & Fragility
- Immobility & Viscosity
- Needless complexity
- Needless repetition
- …



Quality Attributes

- **Reusability**
- **Flexibility**
- **Understandability**
- **Extendibility**
- **…**

- **Maintainability:** Affected by flexibility & extendibility
- **Reliability:** Impacted by poor understandability
- …

Design smell examples?

Output?

```
class Test
    {
        static void Main(string[] args)
        {
          Base b = new Derived("Some message");
        }
    }
```

```
class Base
    {
        protected Base()
        {
            trim();
        }

        protected virtual void trim()
        {
            Console.WriteLine(“Base");
        }
    }
```

```
class Derived:Base
    {
        private readonly string msg = null;

        public Derived(string msg)
        {
            this.msg = msg;
        }
        protected override void trim()
        {
            Console.WriteLine(msg.Trim());
        }
    }
```

Unhandled Exception: System.NullReferenceException: Object reference not set to an instance of an object

Can constructor declaration be improved?

Do not provide public constructors for abstract classes

```
public abstract class ScrollerBase
{
    public ScrollerBase()
    {
      // Code Elided
    }

    // Code Elided
}
```

Issue because of migration from one language to another (C++ to C#)

```
internal class RoutingGenericFilter
{
    public virtual bool IsExpectedResult(string result, string
        filterExpression)
    {
        return true;
    }
}
internal class RoutingContainsFilter : RoutingGenericFilter
{
    public virtual bool IsExpectedResult(string result, string
        filterExpression)
    {
        if (result.Contains(filterExpression))
        {
            return true;
        }
        return false;
    }
}
```

This is correct way of overriding in C++ but hiding in C#

Base class has reference to derived classes

- This violates OCP principle*
- If there is a function which does not conform to the LSP, then that function uses a reference to a base class, but must know about all the derivatives of that base class
- Such a function violates the Open-Closed principle because it must be modified whenever a new derivative of the base class is created.

```
public class ItemSelectorConfigurationModel
{
  //...
    // Get state object depending on type
    if (this instanceof AlarmCodeSelectorConfigurationModel)
    {
      _ItemSelectorState =
           repository.ApplicationState.AlarmCodeSelectorState;
    }
    else if (this instanceof ConverterParameterConfigurationModel)
    {
      _ItemSelectorState =
           repository.ApplicationState.ConverterParameterState;
    }
    // code elided
}
```

```
public class
AlarmCodeSelectorConfiguration
Model extends
ItemSelectorConfigurationModel
{
  // code elided
}
public class
ConverterParameterConfiguratio
nModel extends
ItemSelectorConfigurationModel
{
  // code elided
}
```

*FUNCTIONS THAT USE POINTERS OR REFERENCES TO BASE CLASSES MUST BE ABLE TO USE OBJECTS OF DERIVED CLASSES WITHOUT KNOWING IT.

Considerable amount of code duplication

- There is high amount of code duplication (around 15%) observed in the code
- Code duplication can cause maintenance challenges like inconsistent changes, poor readability and understandability etc.
- Refactorings to address code duplication should be taken up first

```
The following methods have large amount of code duplication between them (and poor naming
of entities. Refer next slide also for naming):


 1) private void UpdateDataGridProgress(object index)
 2) private void UpdateDataGrid1Progress(object index)


The name of methods also indicate the same thing. Consider refactoring of code present in
methods using one method and add a parameter to differentiate.
```

How we can detect design and/or code smells?

- Manual analysis/review
- Automated analysis
- Semi-automated (with the help of metrics)

Understand the limitations of tools/metrics

- Not all the smells can be detected by metrics
- Not all the smells can be detected by tools

Technical Debt

A long time ago in a galaxy far, far away….

There are developers that are “living” peacefully on a planet …

By “living”, we mean they are quietly writing good-quality code that they hope will change the universe

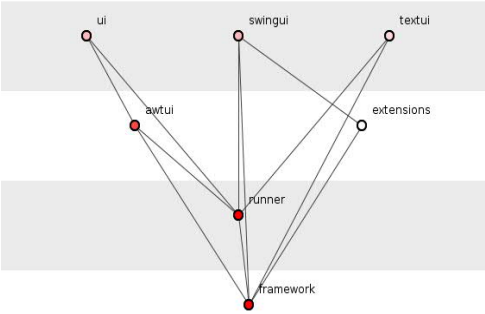
.. When they receive some new requirements and are asked to integrate 10 new features into the release that is due in a month ..

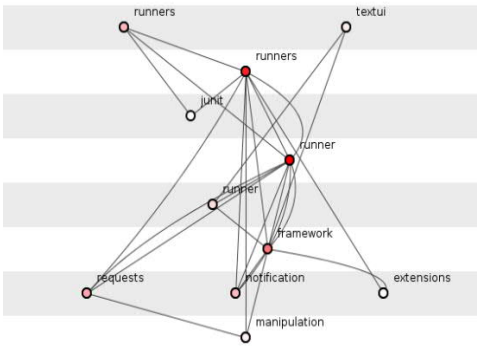
Some of the developers are tempted by the dark side

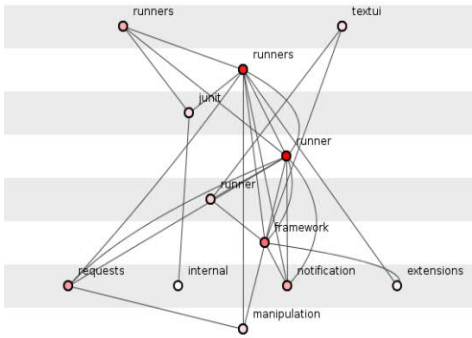
They resort to hacks and shortcuts to meet the stakeholder requirements

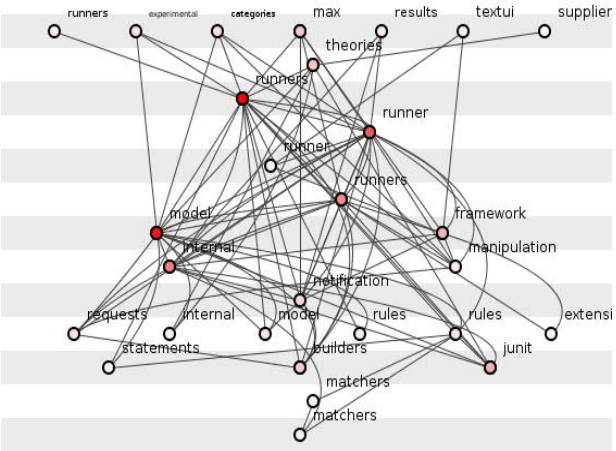
In the process, they compromise on the core guiding principles of software development …

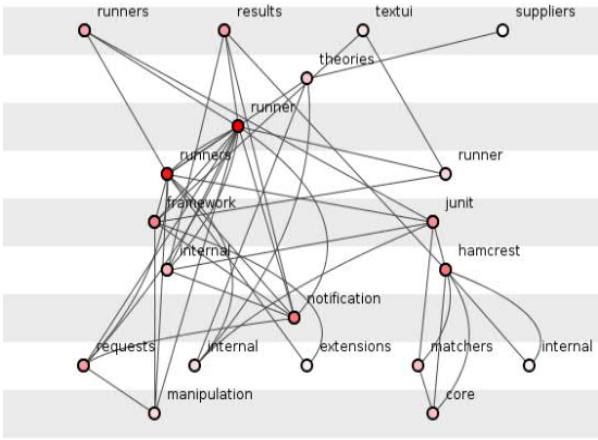
Evolution of the Codebase





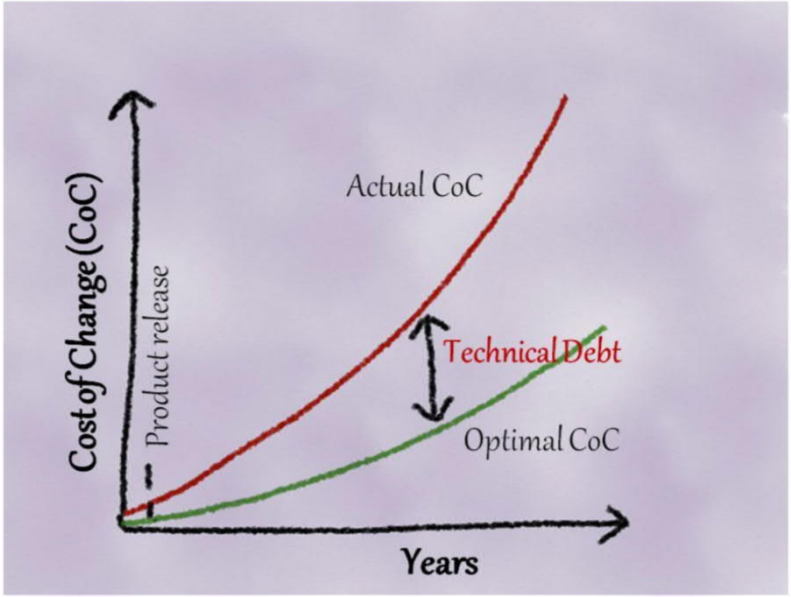






Technical debt

The debt that accrues when one knowingly or unknowingly make technical decisions that are wrong or non-optimal in the current context

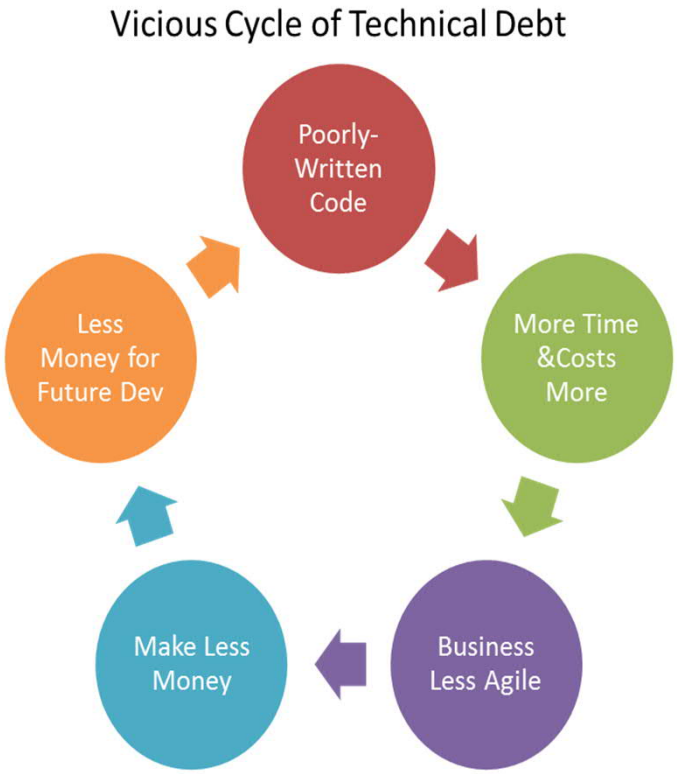


Technical Debt = Principal + Interest

Impact of Technical Debt

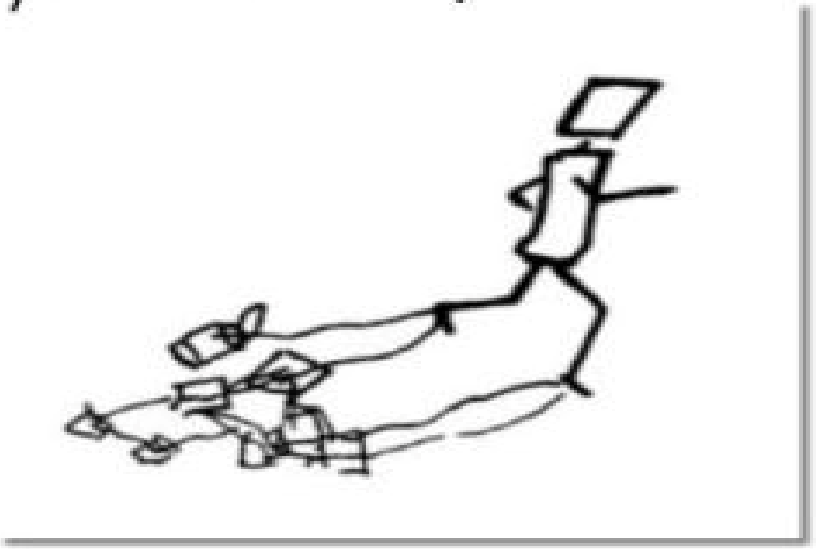
Technical Debt impacts

- Morale
- Productivity
- Quality
- Risk



The project starts to stumble ..

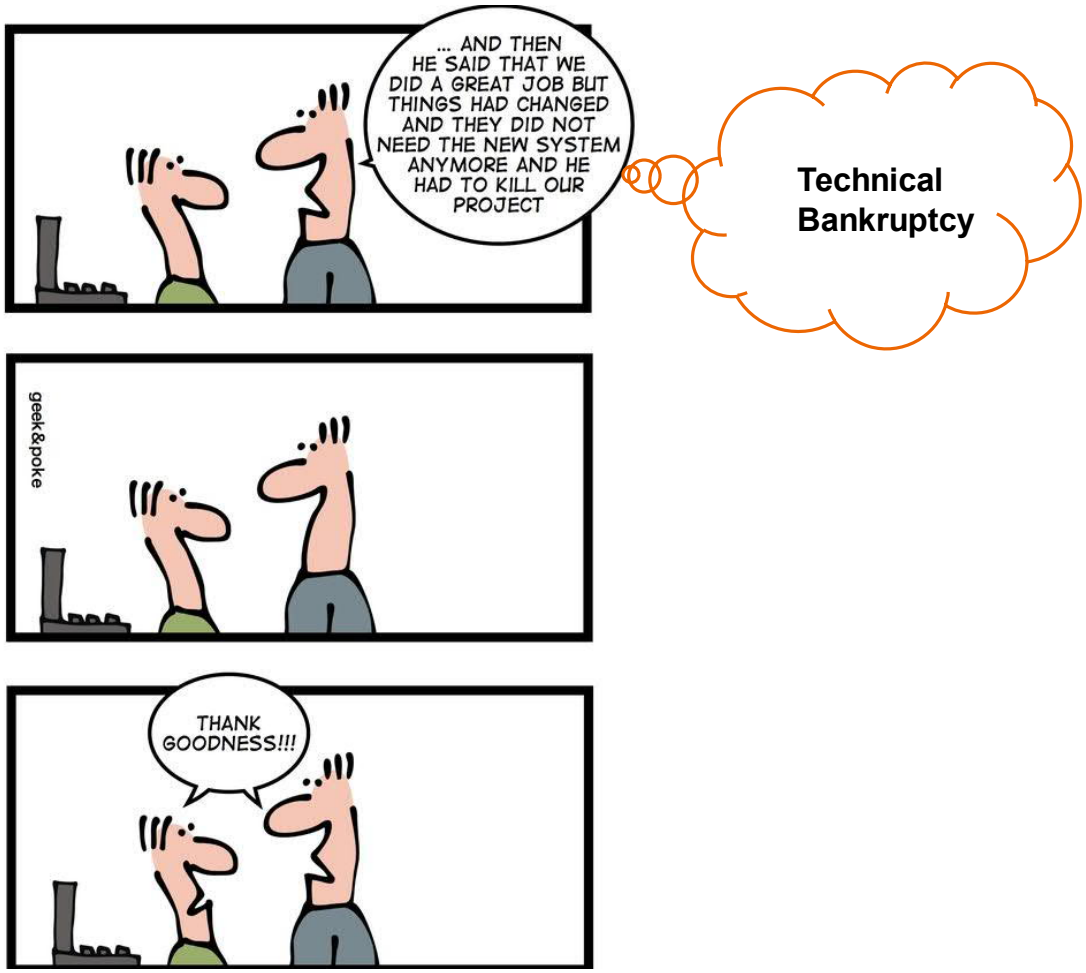
because mounting technical debt
ties the system to the past



by its shoelaces!

How does the story end?

STORY ENDING 1



How does the story end?

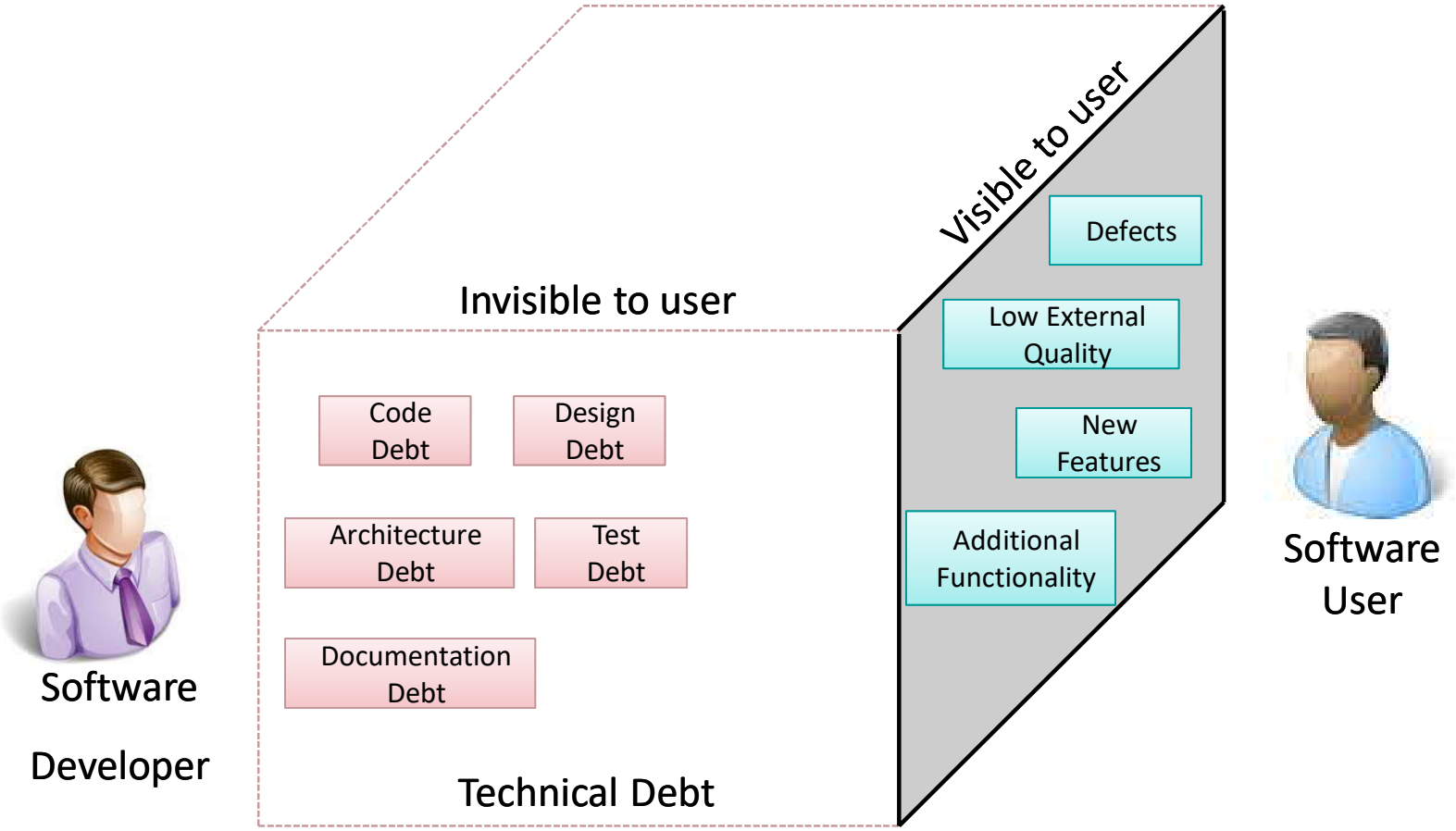
STORY ENDING 2

Employ practical technical debt management techniques

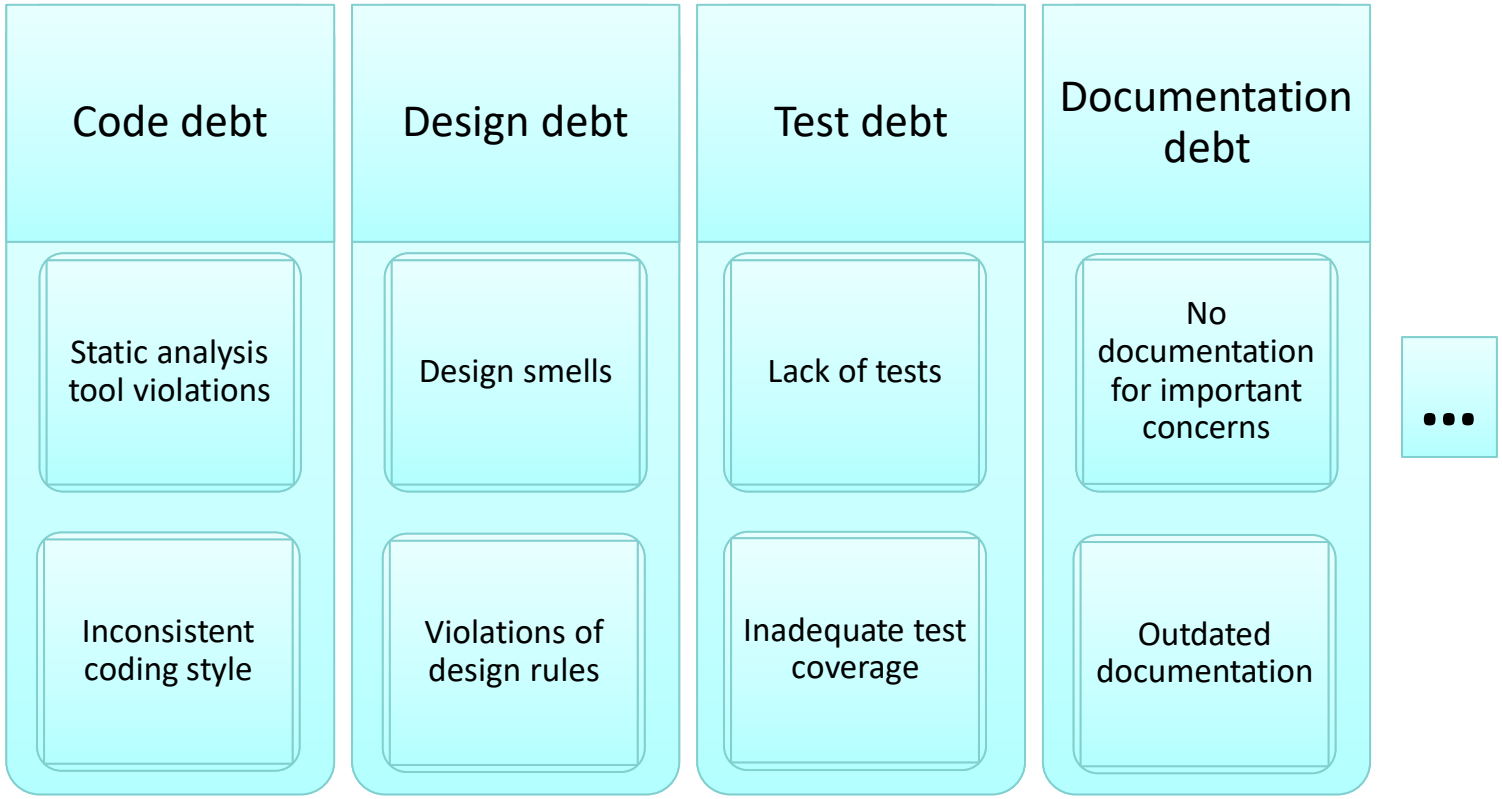
- Continuous monitoring of technical debt
- Continuous refactoring
- Preventing accumulation of technical debt
- Conducting workshops
- …

… and the quality of the project slowly and gradually improves

Organizing the Technical Debt landscape



Dimensions of Technical Debt



What causes Technical Debt?

A MAJOR CAUSE IS VIOLATIONS OF

- **PRINCIPLES**
- **BEST PRACTICES**
- **PROCESSES**

Thank you!