university of groningen

faculty of science
and engineering

object -oriented design (patterns) and energy consumption

(in between the lines)

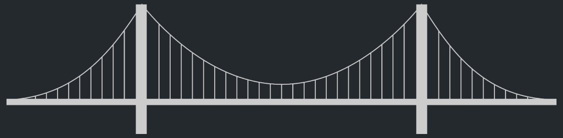
Daniel Feitosa

d.feitosa@rug.nl

A bit about me

Software Quality

Architecture



Code

insights obtained from architectural level (e.g., decisions) impacting code practices and processes (e.g., frameworks)

recurrent development challenges that can be easined by improving architecting processes and informing decisions

Image by Pechristener, licensed under CC BY-SA 4.0

A bit about me

Weapons of choice:

- empirical software engineering
- static and dynamic source code analysis
- mining software repositories

Some topics of interest:

- Technical Debt
- Green Software Engineering
- ML4SE ; SE4ML
- SW-HW co-design (e.g., IoT)

Assistant Professor

Software Engineering and Architecture Group

www.cs.rug.nl/search/People/DanielFeitosa

feitosa-daniel.github.io

Today's menu

Software patterns

Empirical SE

Source code analysis (OO)

Technical debt

energy consumption 👀

Examples

- Energy consumption of GoF instances
- Cost management in multi-service cloud applications 🍲

Software Patterns

Software Patterns

Proven solutions for recurrent problems

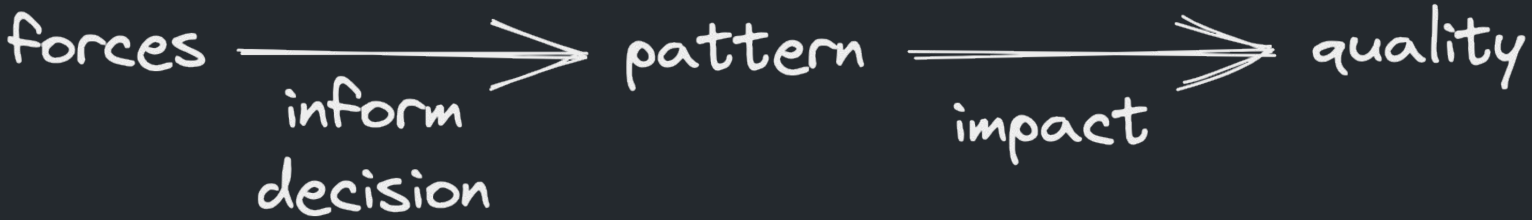
Patterns have

- Problem description
- Solution
- Usage examples
- Forces (reasons to apply a pattern)

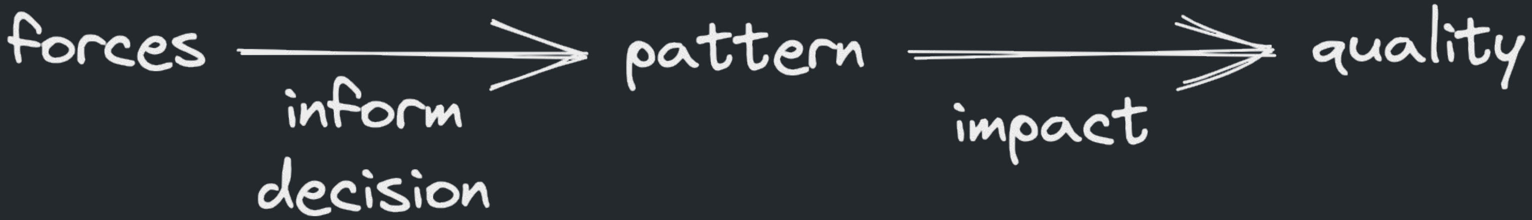
E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture Volume 1: A System of Patterns, 1st ed. Wiley, 1996.

Patterns impact quality



Patterns impact quality



BUT . . .

- A system's design isn't static (e.g., forces change)
- Other design elements (e.g., patterns) may have negative impact
- Some impacts are invisible (or not accounted for)

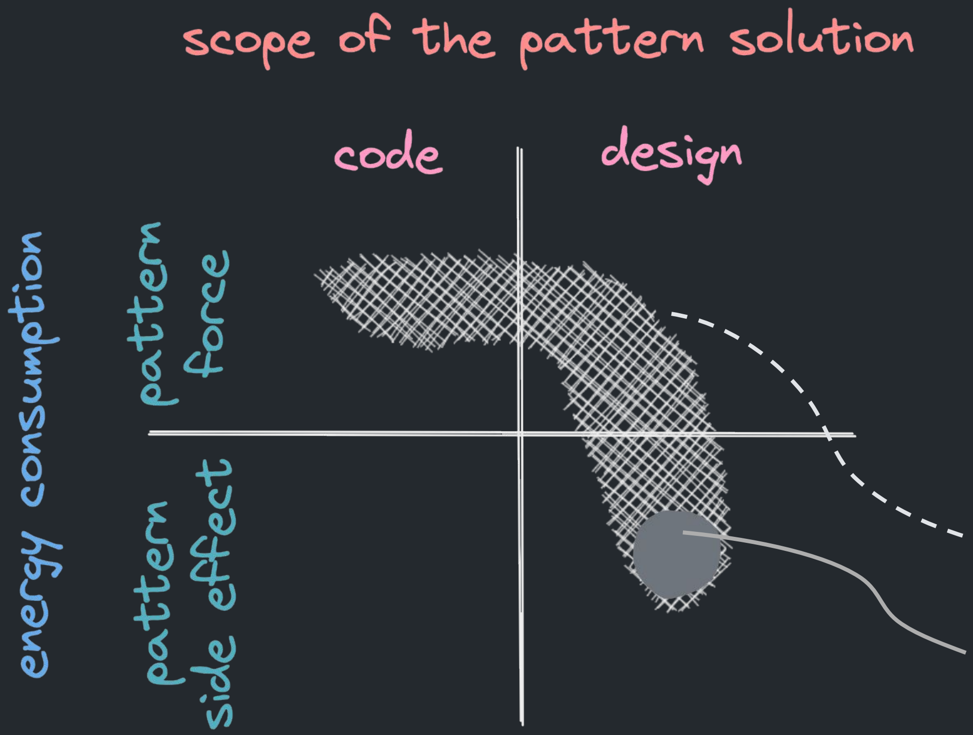
N. B. Harrison and Paris Avgeriou, " Using Pattern-Based Architecture Reviews to Detect Quality Attribute Issues - an Exploratory Study," Transactions on Pattern Languages of Programming III , vol. 7840, pp. 168–194, 2013, doi: 10.1007/978-3-642-38676-3_5.

I. Ozkaya, R. Kazman, and M. Klein, "Quality-Attribute-Based Economic Valuation of Architectural Patterns," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, techreport CMU/SEI-2007-TR-003, 2007. doi: 10.1184/R1/6582686.V1.

R. Wojcik et al., "Attribute-Driven Design (ADD), Version 2.0," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, techreport CMU/SEI-2006-TR-023, 2006. doi: 10.1184/R1/6572066.v1.

Study

Pattern and energy



D. Feitosa, L. Cruz, R. Abreu, J. P. Fernandes, M. Couto, and J. Saraiva, “Patterns and Energy Consumption: Design, Implementation, Studies, and Stories,” in Software Sustainability, Springer International Publishing, 2021, pp. 89–121. doi: 10.1007/978-3-030-69970-3_5.

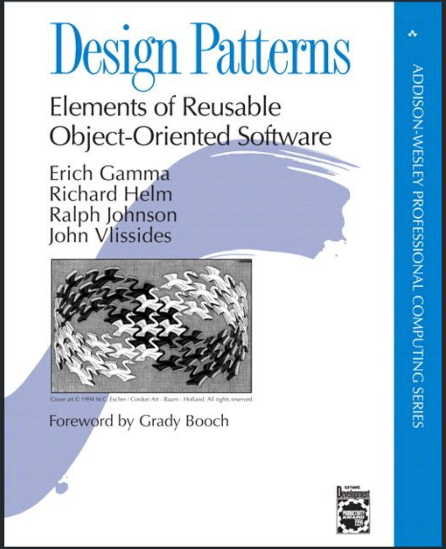
D. Feitosa, R. Alders, A. Ampatzoglou, P. Avgeriou, and E. Y. Nakagawa, “Investigating the effect of design patterns on energy consumption,” Journal of Software: Evolution and Process, vol. 29, no. 2, p. e1851, Jan. 2017, doi: 10.1002/smr.1851.

How do design patterns impact energy efficiency?

Research Question

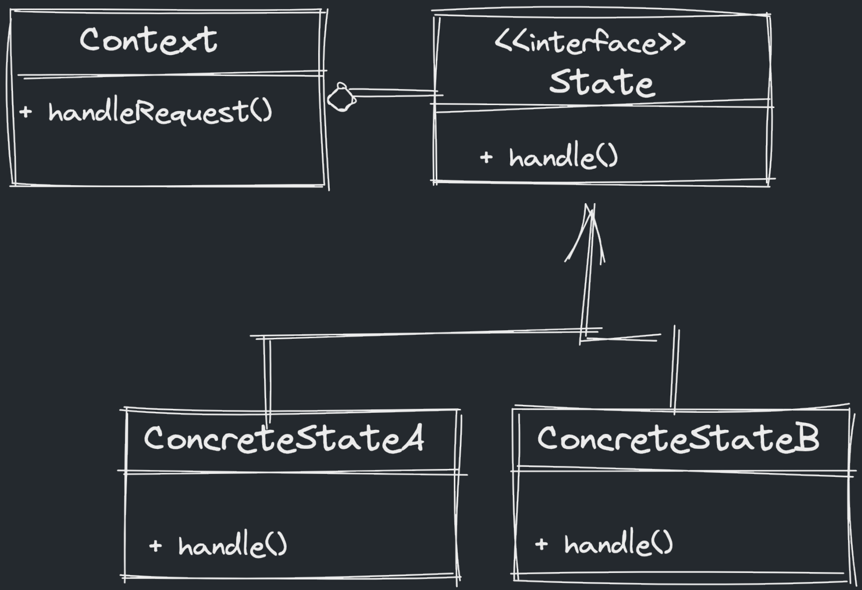
What is the energetic difference between
a pattern instance and
an alternative (non-pattern) design?

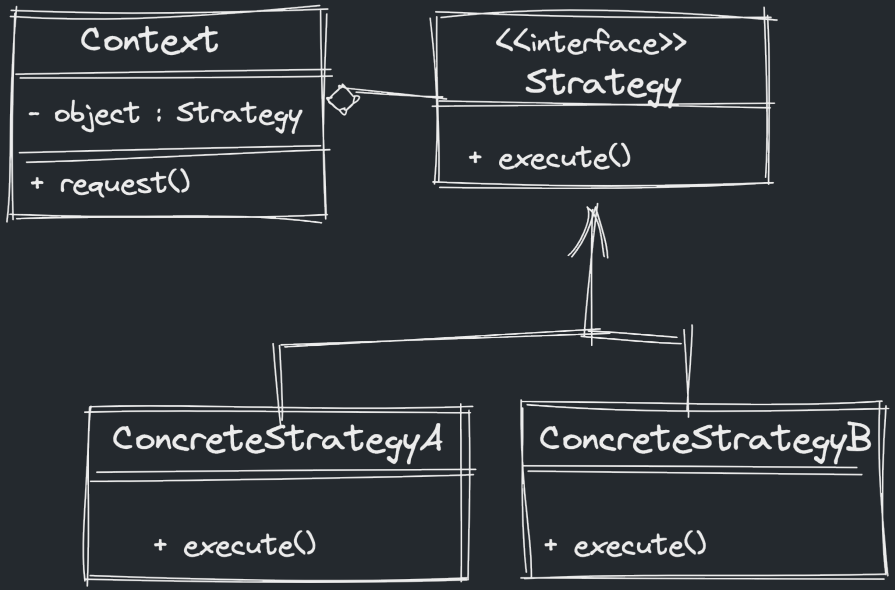
(object-oriented design)


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

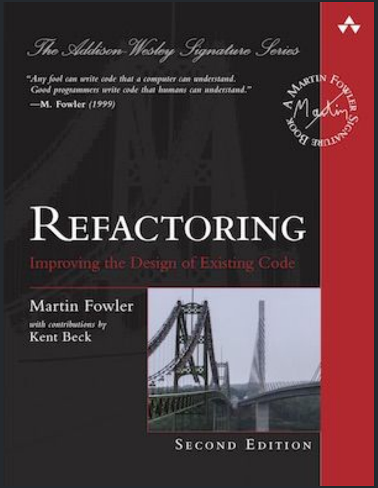
E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, Design Patterns:
Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

State & Strategy





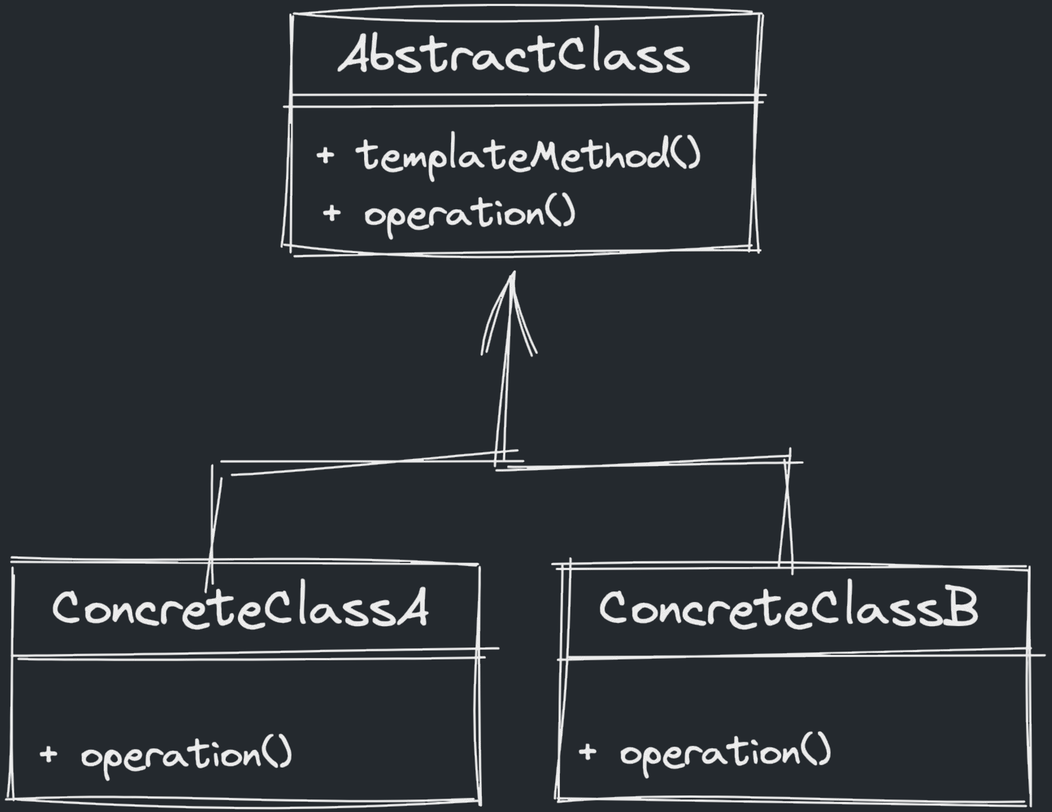
Alternative to State/Strategy


The Addison-Wesley Signature Series
"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
—M. Fowler (1999)
A Martin Fowler Signature Book
Martin
Refactoring
Improving the Design of Existing Code
Martin Fowler
with contributions by
Kent Beck
Second Edition

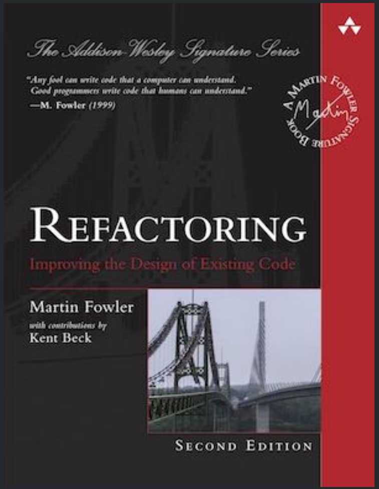
```
public class Strategy {
    public enum Strategies { Strategy1, Strategy2, Strategy3 };
    private Strategies currentStrategy;
    public void execute (A attribute) {
        switch (currentStrategy) {
            case Strategy1:
                // Implementation of Strategy 1
            break;
            case Strategy2:
                // Implementation of Strategy 2
            break;
            case Strategy3:
                // Implementation of Strategy 3
            break;
        }
    }
    // . . .
```

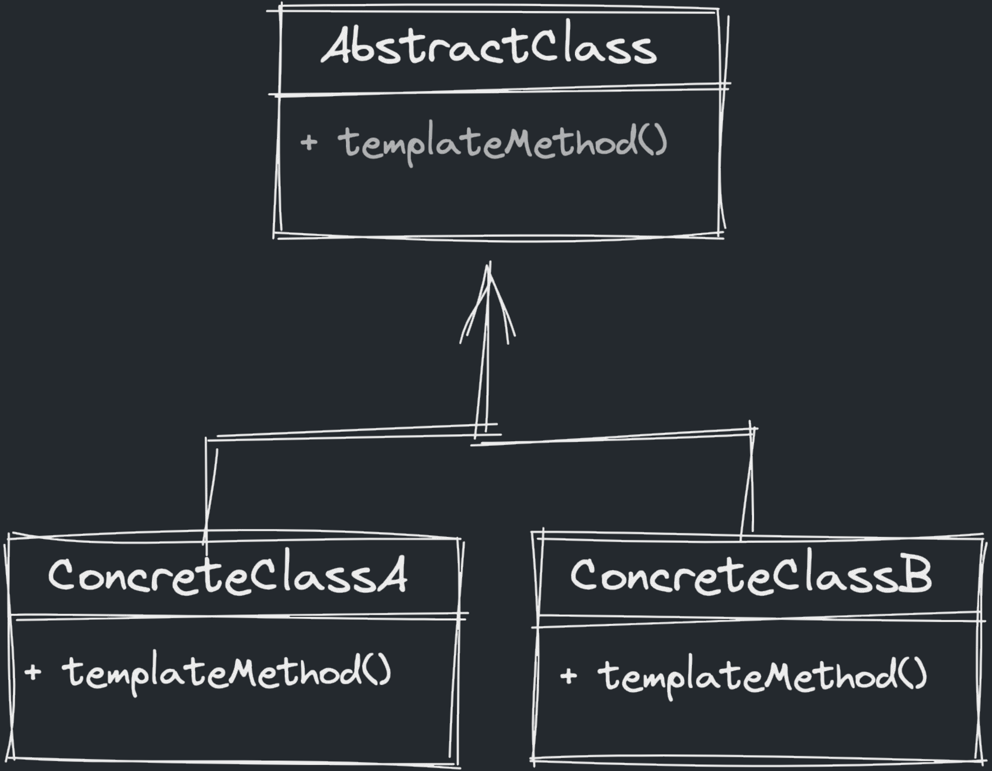
M. Fowler and K. Beck, Refactoring: Improving the Design of Existing Code, 2nd ed. Addison-Wesley, 2013.

Template Method



Alternative to Template Method


The Addison-Wesley Signature Series
"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
—M. Fowler (1999)
A Martin Fowler Signature Book
Refactoring
Improving the Design of Existing Code
Martin Fowler
with contributions by
Kent Beck
Second Edition



M. Fowler and K. Beck, Refactoring: Improving the Design of Existing Code, 2nd ed. Addison-Wesley, 2013.

Before we get to business...

What are the pros and cons of each design solution?

What would you expect w.r.t. energy consumption?

What are your hypotheses?

Empirical Study Design

Protocol

How to answer the research question?

- What type of empirical study?
- What do we measure and how?
- What are the limitations?

What type of study?

We want to compare equivalent artefacts (design to solve a problem)

- The design differs (pattern vs non-pattern)
- Functionality (and everything else) should be the same

What type of study?

We want to compare equivalent artefacts (design to solve a problem)

- The design differs (pattern vs non-pattern)
- Functionality (and everything else) should be the same

i.e., experiment where the design are the different treatments

What do we measure?

We measure the energy consumed by the two designs...

Under what context?

- Do we create example systems?
- Do we use "real-world" systems?

What do we measure?

We measure the energy consumed by the two designs...

Under what context?

- Do we create example systems?
- <u>Do we use "real-world" systems?</u>

had been done

how?

What do we measure?

We measure the energy consumed by the two designs...

Under what context?

- "Real-world" (i.e., non-trivial) systems
- Write a test case that uses a pattern instance in a regular scenario

Which patterns?

What do we measure?

We measure the energy consumed by the two designs...

Under what context?

- "Real-world" (i.e., non-trivial) systems
- Write a test case that uses a pattern instance in a regular scenario

Which patterns?

- Find patterns in the system
- Implement non-pattern equivalents
- Write test that can run either design solution

What do we measure?

We measure the energy consumed by the two designs...

Under what context?

- "Real-world" (i.e., non-trivial) systems
- Write a test case that uses a pattern instance in a regular scenario

Which patterns?

- Find patterns in the system
- Implement non-pattern equivalents
- Write test that can run either design solution

why implementing the non-pattern solution and not the other way around?

What do we measure?

We measure the energy consumed by the two designs...

Under what context?

- "Real-world" (i.e., non-trivial) systems
- Write a test case that uses a pattern instance in a regular scenario

Which patterns?

- Find patterns in the system
- Implement non-pattern equivalents
- Write test that can run either design solution

mitigate selection bias

the pattern was the intended solution all along

Selecting suitable pattern instances

- Used within the application

  (e.g., no API features)

- Reachable

  (i.e., easy to test to mitigate measurement bias)

- Performing deterministic tasks

  (e.g., no IO functionality)

- Not too complex

  (i.e., discard exceptionally large instances, e.g., with 20 or more concrete states/strategies)

How to measure energy consumption?

What procedure?

Which (type of) tool?

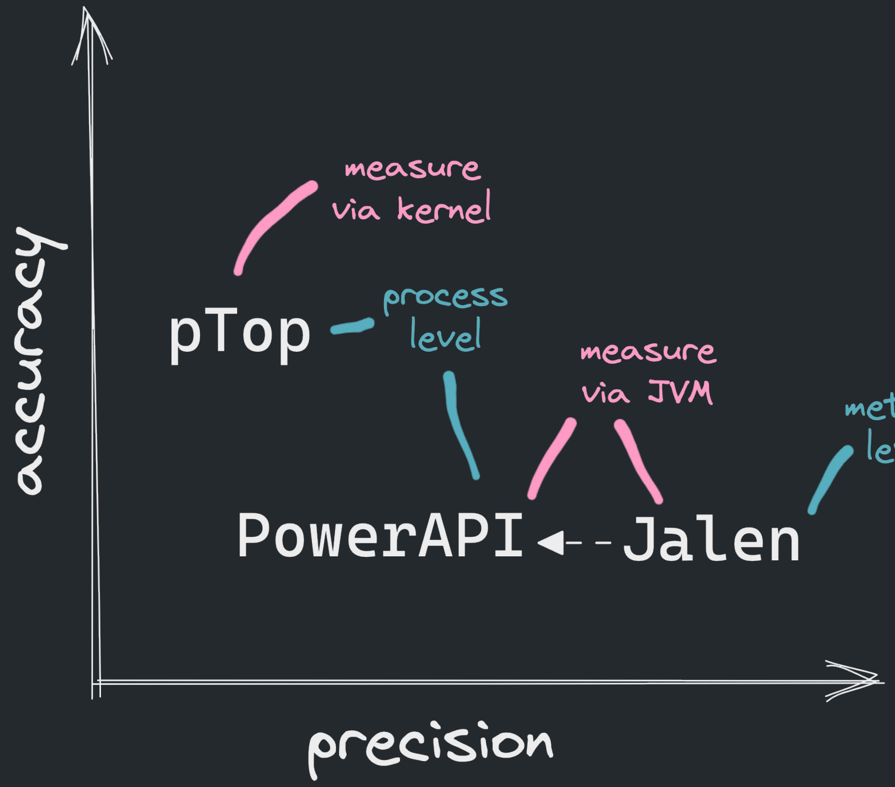
How to measure energy consumption?

One measurement

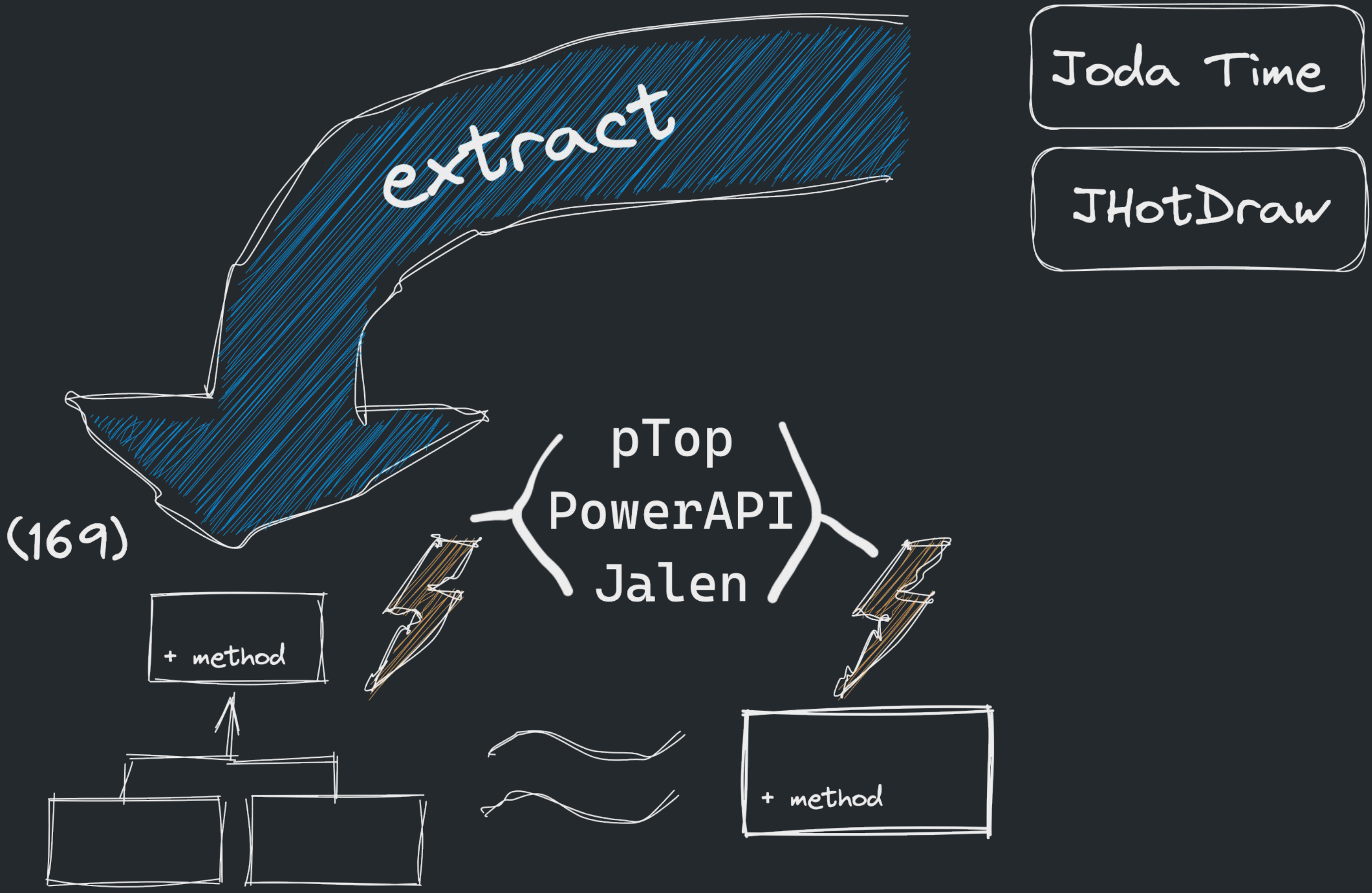
1. Ensure all non-essential applications are closed;
2. Choose a design at random (pattern or non-pattern);
3. Let the computer rest (e.g., 30 seconds);
4. Start energy measurement tools;
5. Run a test case multiple times to produce to measurable energy draw (depended on test; between 10 and 100 times);
6. Repeat 3–5 for the second design.

Repeat each measurement 100 times to produce reliable results

The energy measurement tools



The study in a nutshell



Joda Time

JHotDraw

extract

(169)

+ method

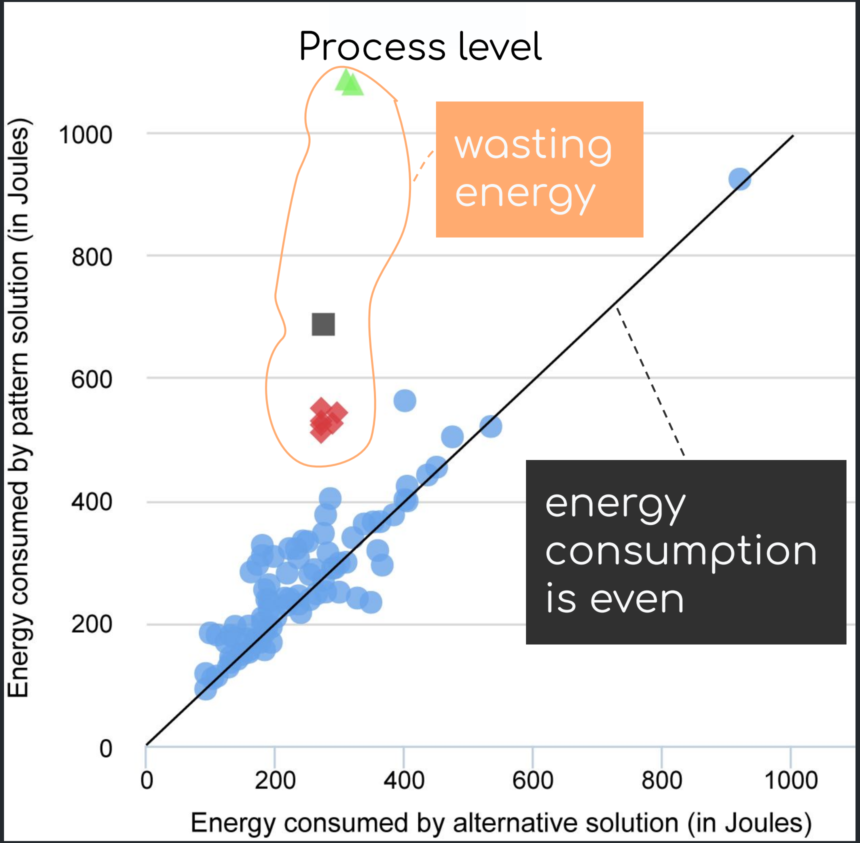
pTop
PowerAPI
Jalen

+ method

Object-Orientation & Energy

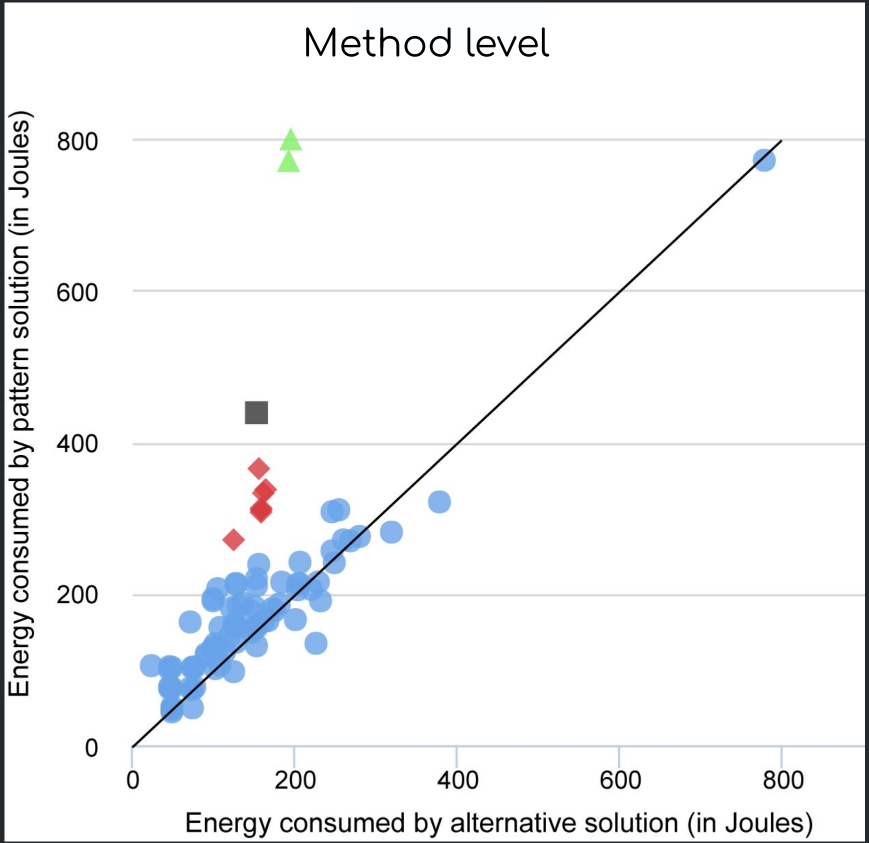
Let's check those hypotheses

Template Method

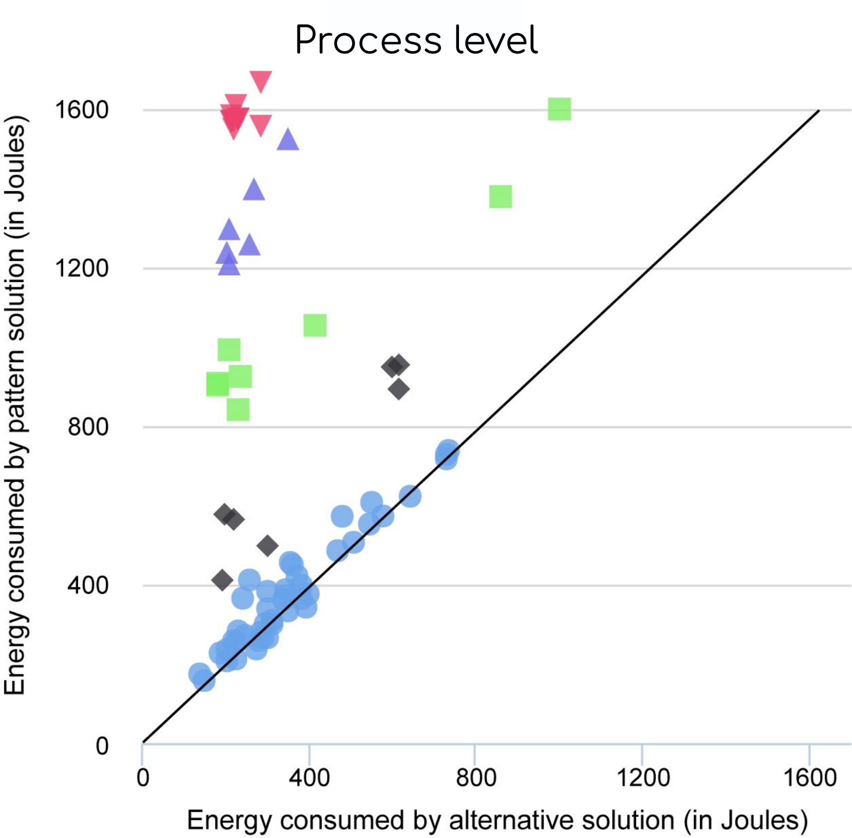


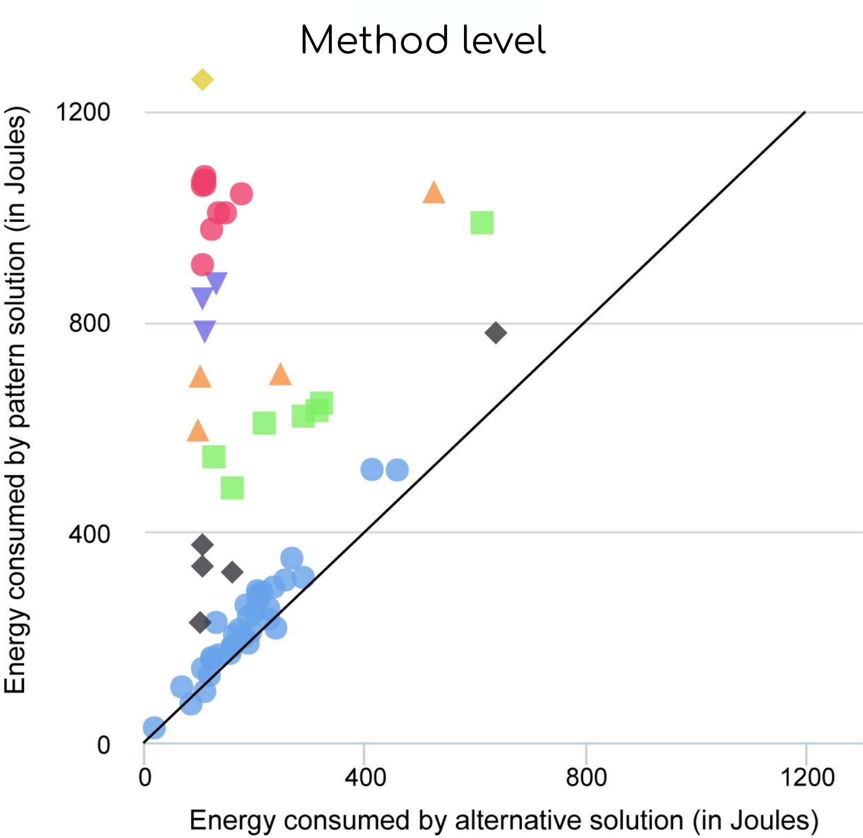
wasting energy

energy consumption is even

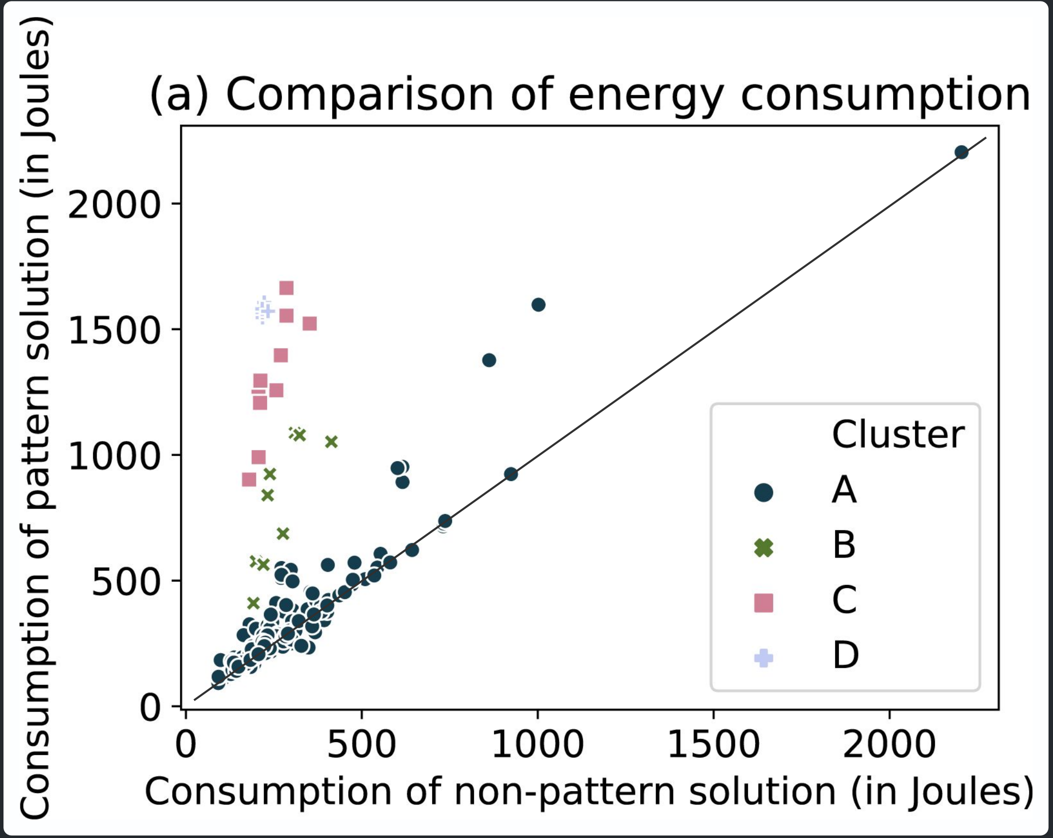


State/Strategy

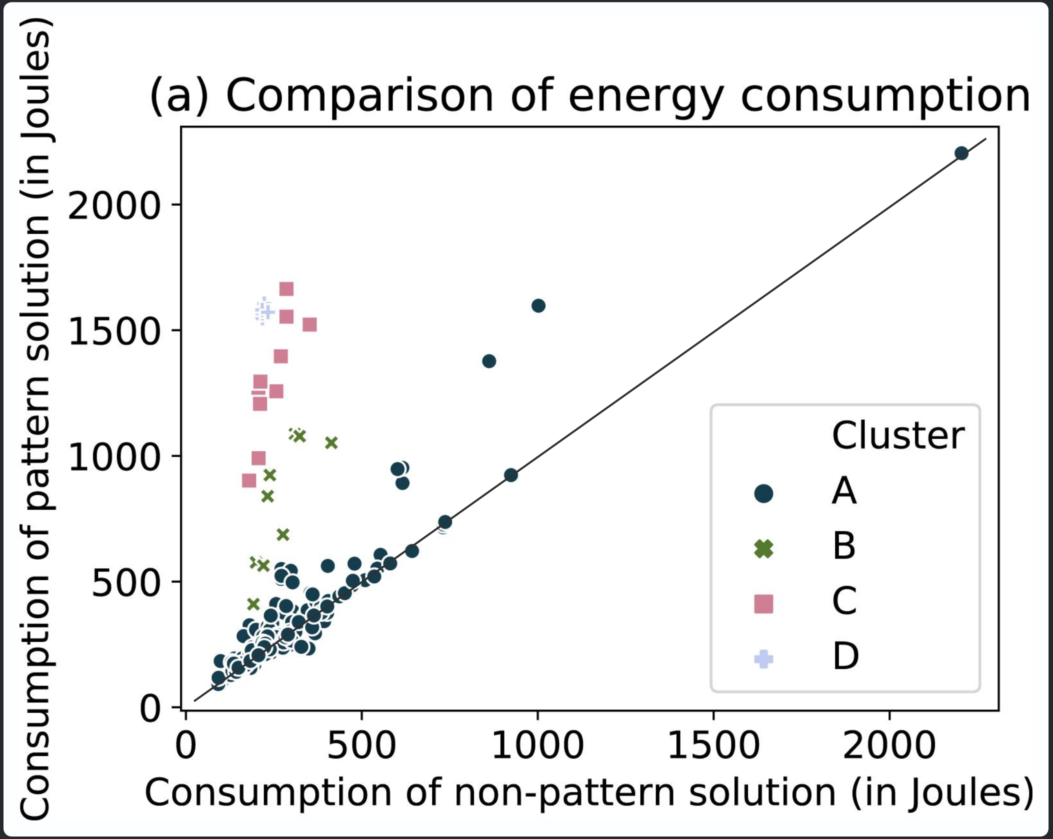


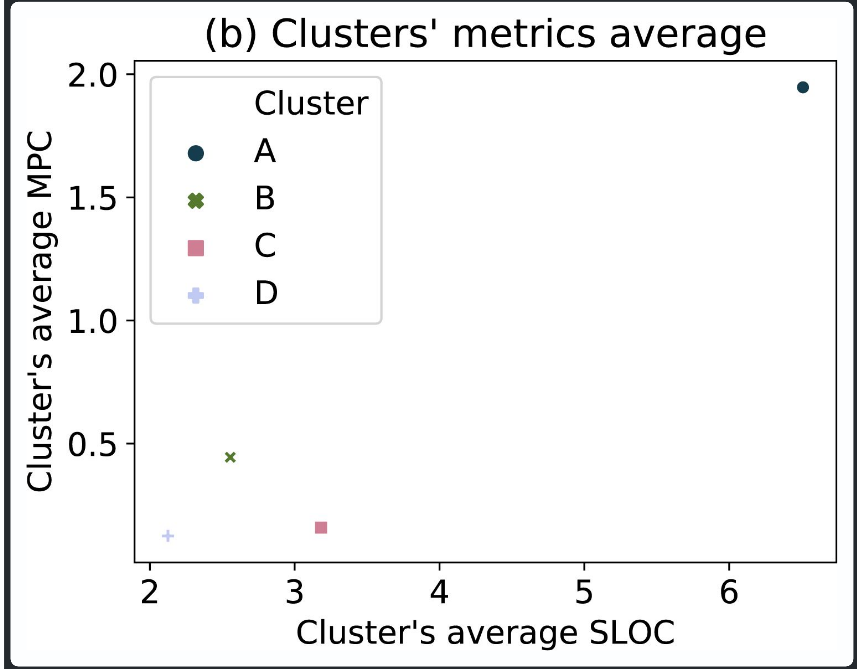


All data points together



All data points together

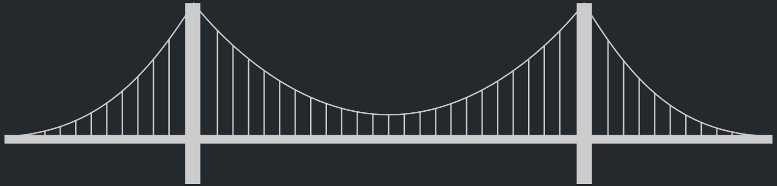




How do design patterns impact energy efficiency?

Is a monolithic design worth it?

Can I harvest more performance by avoiding OO features?



(keep the study design in mind)

Technical Debt

Have you ever written “poor” code to save time?

What is Technical Debt (TD)?

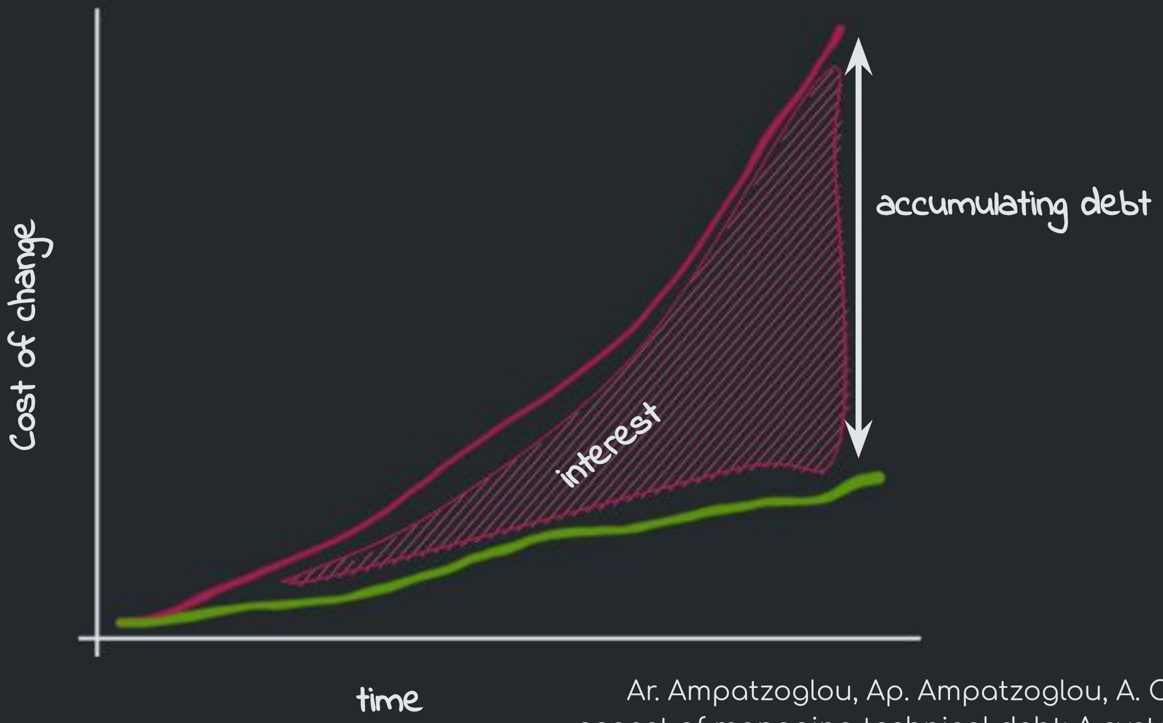
A collection of design and implementation decisions that solve problems but make future changes more costly or impossible.

Based on explanation coined by Cunningham on "The WyCash Portfolio Management System." OOPSLA92 Experience Report. http://c2.com/doc/oopsla92.html

A trade-off between the short-term benefits of "*cutting corners*" in software development and the long-term sustainability of a software system.

Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and management," Journal of Systems and Software, vol. 101, no. C, pp. 193–220, 2015 https://doi.org/10.1016/j.jss.2014.12.027

The finance of Technical Debt

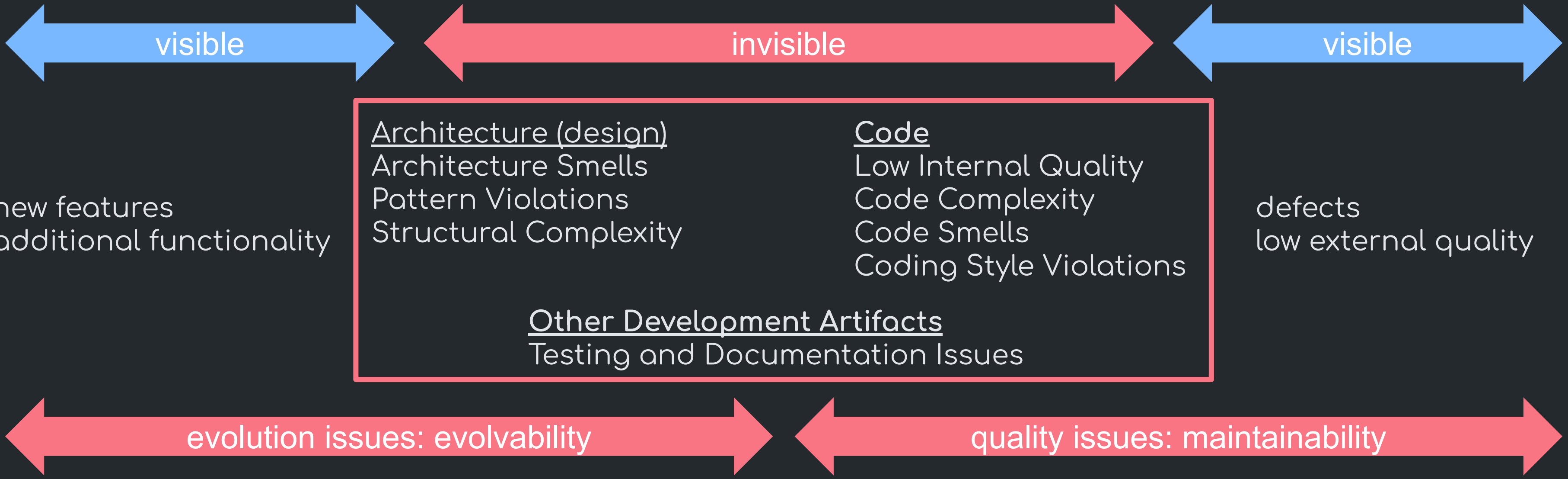


interest: the additional development effort required to modify the software (adding new features or fixing bugs)

principal: the effort required to eliminate inefficiencies in the current design or implementation of a software system

Ar. Ampatzoglou, Ap. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou The financial aspect of managing technical debt: A systematic literature review Information and Software Technology, 64 (Aug. 2015), pp. 52-73 https://doi.org/10.1016/j.infsof.2015.04.001

How to find symptoms (and debt)?



visible

invisible

visible

new features
additional functionality

Architecture (design)
Architecture Smells
Pattern Violations
Structural Complexity

Code
Low Internal Quality
Code Complexity
Code Smells
Coding Style Violations

Other Development Artifacts
Testing and Documentation Issues

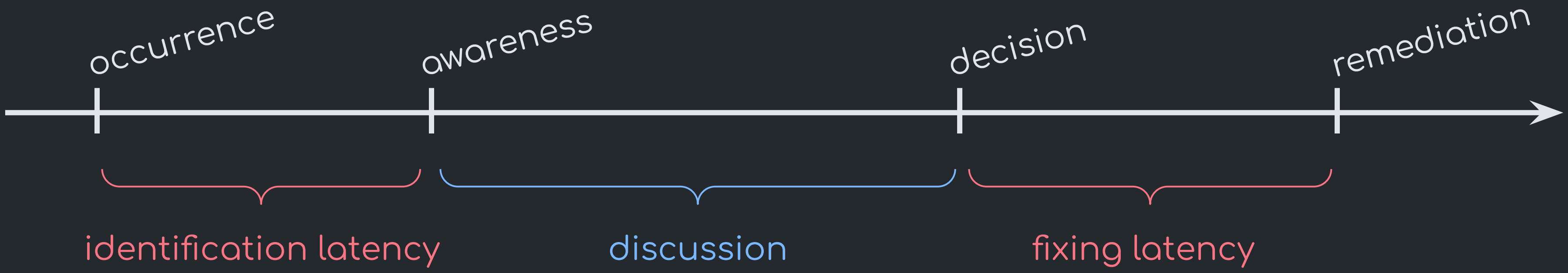
defects
low external quality

evolution issues: evolvability

quality issues: maintainability

P. Kruchten, R. L. Nord and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," in IEEE Software, vol. 29, no. 6, pp. 18-21, 2012. https://doi.org/10.1109/MS.2012.167

Technical Debt timeline



occurrence

awareness

decision

remediation

identification latency

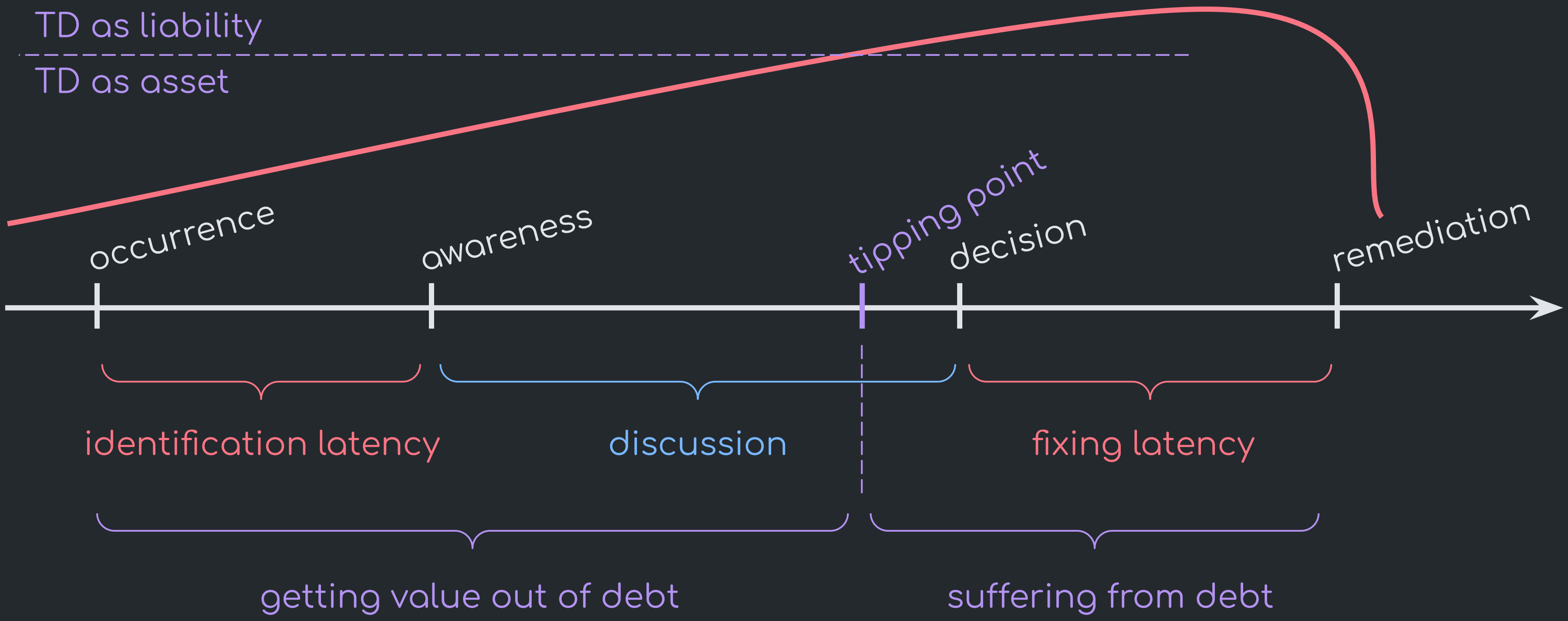
discussion

fixing latency

TD can be unintentional

knowing what to do
is just the beginning

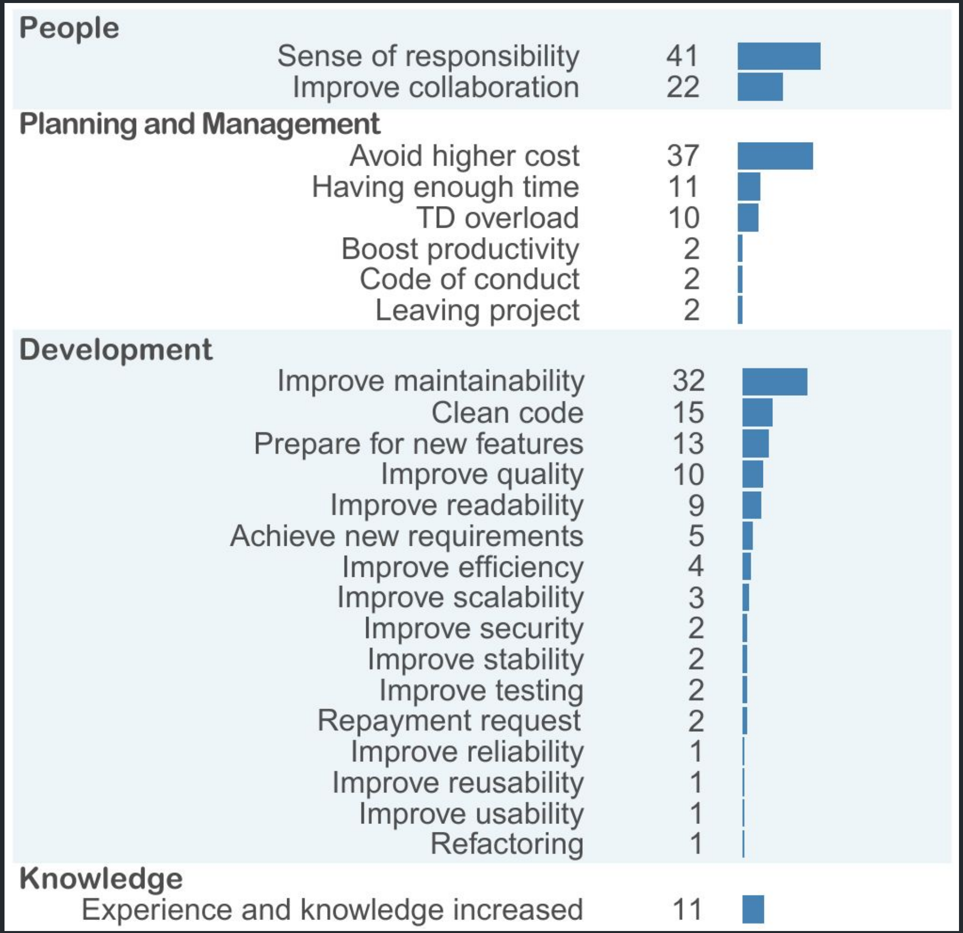
Technical Debt management



TD as a bridge

Team members often understand the system at different levels of abstraction
- managers
- product owner
- scrum master
- technical leader
- developers

But TD can serve as a common language



People		
	Sense of responsibility	41
	Improve collaboration	22
Planning and Management		
	Avoid higher cost	37
	Having enough time	11
	TD overload	10
	Boost productivity	2
	Code of conduct	2
	Leaving project	2
Development		
	Improve maintainability	32
	Clean code	15
	Prepare for new features	13
	Improve quality	10
	Improve readability	9
	Achieve new requirements	5
	Improve efficiency	4
	Improve scalability	3
	Improve security	2
	Improve stability	2
	Improve testing	2
	Repayment request	2
	Improve reliability	1
	Improve reusability	1
	Improve usability	1
	Refactoring	1
Knowledge		
	Experience and knowledge increased	11

J. Tan, D. Feitosa, P. Avgeriou, "Do practitioners intentionally self-fix Technical Debt and why?" in ICSME '21, pp. 251-262, 2021. https://doi.org/10.1109/ICSME52107.2021.00029

What is next?

Emerging topics

Machine learning code

but you hear about already 😁

Infrastructure as code

cloud orchestration

Cloud infrastructure

Cloud orchestrators
(e.g., Terraform, Cloudify)

Control deployment

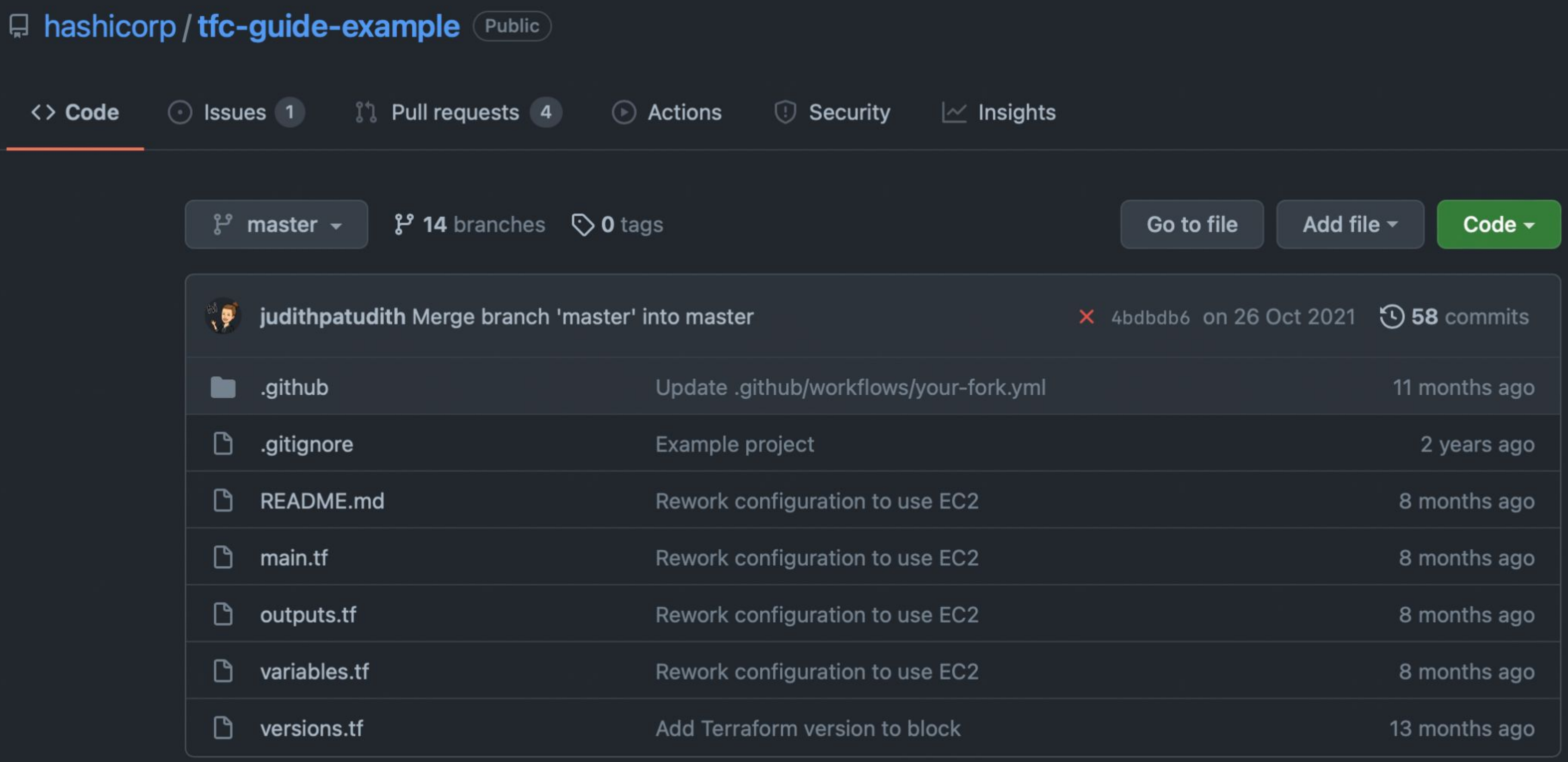
- flexibility to demand
- cloud-agnostic

Infrastructure as code (IaC)

```
variable "service_image" {
 type        = string
 description = "Image ID of the service"
 default     = "my_proj-0126dac26fa89b32"
}

variable "instance_type" {
 type        = string
 description = "Instance for service"
 default     = "t3.micro"
}

variable "geographical_zone" {
 type        = string
 description = "Zone for deployment"
 default     = "eu-nl"
}
```



hashicorp / tfc-guide-example Public

14 branches 0 tags

judithpatudith Merge branch 'master' into master ✕ 4bdbdb6 on 26 Oct 2021 58 commits

.github	Update .github/workflows/your-fork.yml	11 months ago
.gitignore	Example project	2 years ago
README.md	Rework configuration to use EC2	8 months ago
main.tf	Rework configuration to use EC2	8 months ago
outputs.tf	Rework configuration to use EC2	8 months ago
variables.tf	Rework configuration to use EC2	8 months ago
versions.tf	Add Terraform version to block	13 months ago

Takeaway messages

- Energy consumption should be managed at both design and code levels.
- Object-oriented features do not imply energetic waste; but must used with caution (the polymorphic mechanism overhead).
- Technical choices with negative energetic impact are inevitable (technical debt); monitor and manage it!
- Application code is only software (especially in the cloud era); infrastructure can also be optimized.

d.feitosa@rug.nl