Green Coding
Concepts and Practices

Prophet-Greenaden Hack, 23.-25.9.2024

Jari Porras, Tuomas Mäkilä, Oshani Weerakoon, Jari-Matti Mäkelä

UNIVERSITY OF TURKU

How do we define green coding?

Green coding defined (by elicit)

Green coding is an **approach to software development that aims to minimize energy consumption and environmental impact** of information and communication technologies (ICT) (Junger et al., 2024; Junger et al., 2023). It encompasses **various strategies**, including energy-efficient algorithms (Palacios et al., 2014), optimized source code (Corral-García et al., 2015), and principled approximation techniques (Baek & Chilimbi, 2009). Green coding practices can be **integrated into existing industrial processes** and education curricula to **promote sustainable software development** (Junger et al., 2024). **Tools like** Android Lint can be extended to enforce green coding rules in mobile app development (Le Goaër, 2020). The concept of "green codes" also **extends to communication systems**, where energy efficiency is considered **in both transmission and processing** (Grover & Sahai, 2008). These approaches collectively contribute to reducing the energy and resource consumption of ICT systems.

Complexity of green coding - initial thoughts

- Software itself does not use energy, hardware does
- When software is run on hardware, it requires hardware "resources" and thus uses energy
- Two important parts for running a software - processing and communication
  - If you want to minimize energy usage, you need to minimize these two operations

- Note: One might also need to consider the human interface - display, etc.

Sounds simple ?

What does academic literature reveal so far?

- Search query: (“Green”) AND (“Coding” OR “Programming”)
- Data source: Web of Science database (combines various sources)
- Outcome:
  - 1870 articles in the computer science field
  - 78 selected based on titles and 58 after abstracts
- **Categorization into interesting categories related to green coding**
- Snowballing (finding connected material)

Examples of articles

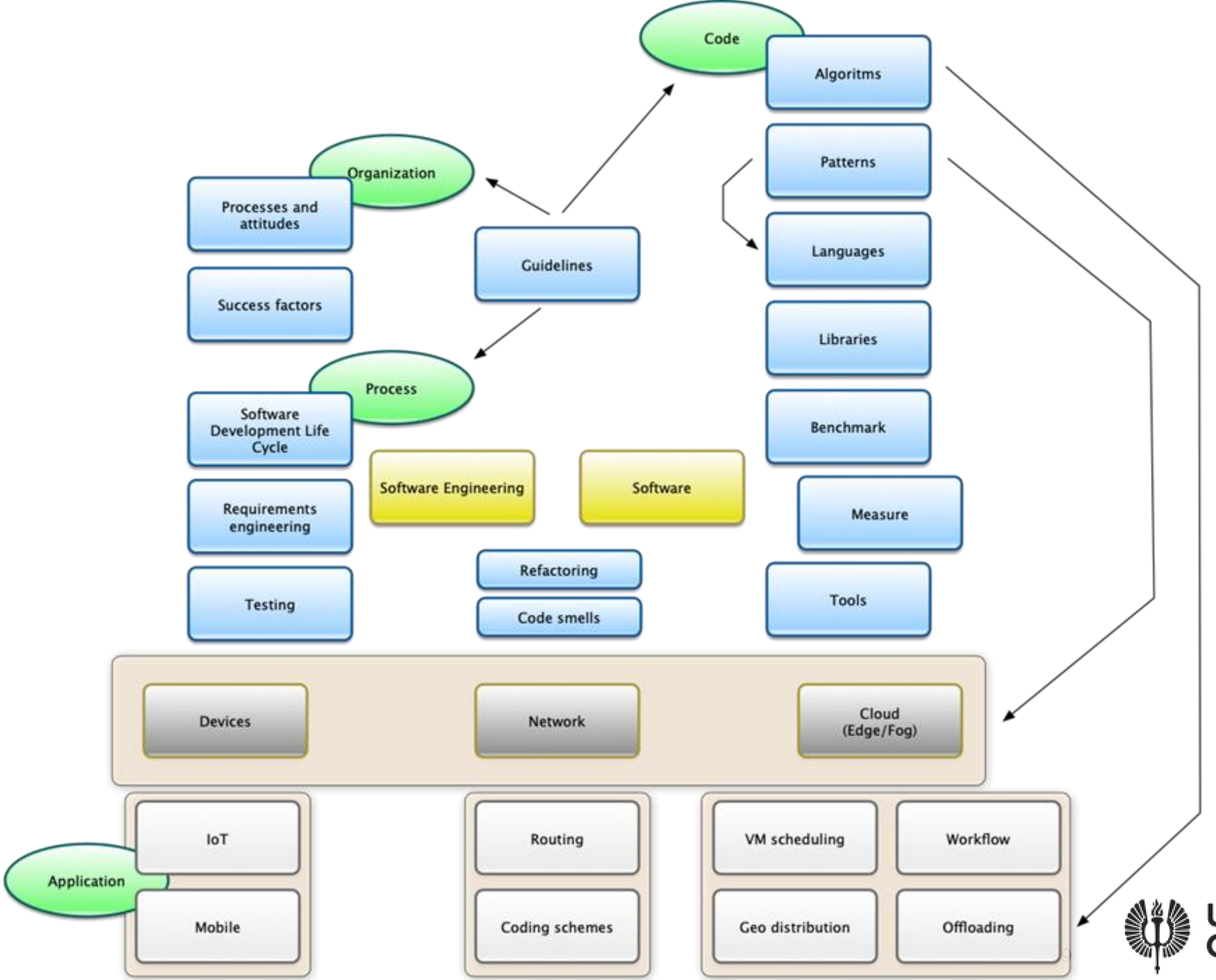
- Salam, M., & Khan, S. U. (2018). **Challenges in the development of green and sustainable software for software multisourcing vendors**: Findings from a systematic literature review and industrial survey.
- Salam, M., & Khan, S. U. (2016). **Developing green and sustainable software: Success factors for vendors**.
- Poth, A., & Momen, P. (2024) **Sustainable software engineering—A contribution puzzle of different teams in large IT organizations**.
- Radersma, R. (2022). **Green Coding: Reduce Your Carbon Footprint**.
- Georgiou, S., Rizou, S., & Spinellis, D. (2019). **Software development lifecycle for energy efficiency: techniques and tools**.
- Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A., & De Lucia, A. (2019). **On the impact of code smells on the energy consumption of mobile applications**.

Examples of articles

- Maleki, S., Fu, C., Banotra, A., & Zong, Z. (2017). **Understanding the impact of object-oriented programming and design patterns on energy efficiency**.
- Connolly Bree, D., & Ó Cinnéide, M. (2023). **Energy efficiency of the Visitor Pattern: contrasting Java and C++ implementations**.
- Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). **Ranking programming languages by energy efficiency**.
- Koedijk, L., & Oprescu, A. (2022). **Finding significant differences in the energy consumption when comparing programming languages and programs**.
- Goaër, O. L. (2020). **Enforcing green code with Android lint**.

UNIVERSITY OF TURKU

Categorization



UNIVERSITY OF TURKU

On organization-level, the focus is on understanding what is important

Based on literature

S.No	Risk Factors
01	Lack of green requirements engineering practices
02	High power consumption (process, resources, and the product itself)
03	High carbon emission throughout the software development
04	Poor software design (architectural, logical, physical, and user interface)
05	Lack of information and communication technologies coordination and communication
06	High resource requirements
07	Lack of coding standards

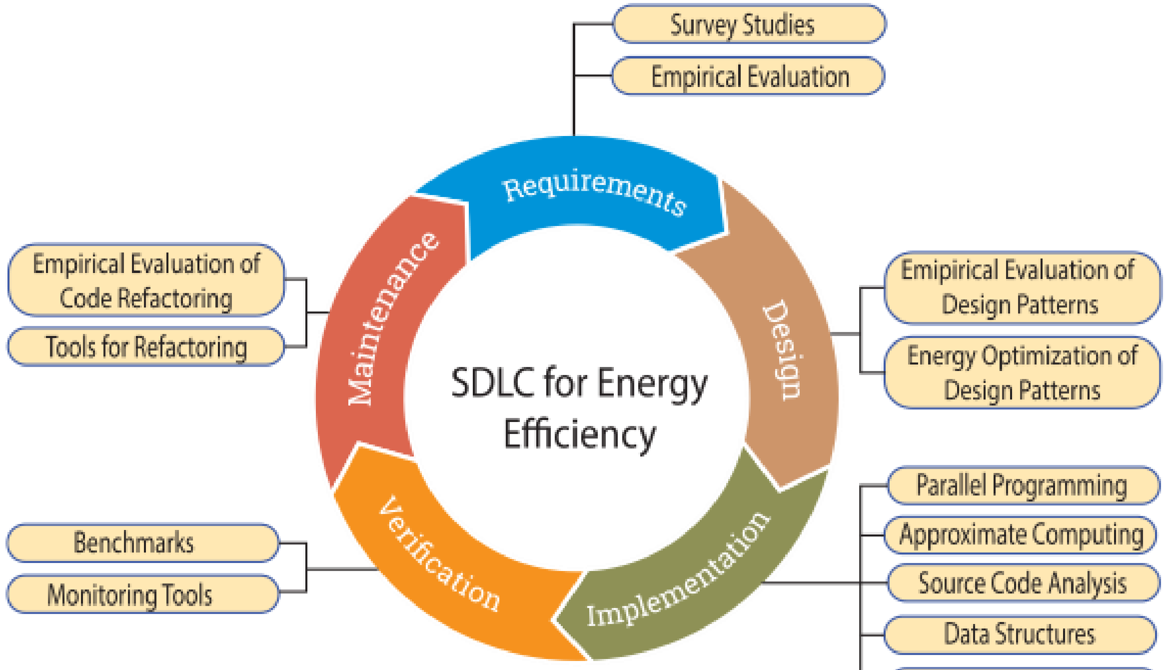
Based on survey

S.No	Risk Factors	Total Responses from Industry Practitioners = 108									
		Optimistic				Pessimistic				Impartial	
		ES	MS	SS	Optimistic %	SD	MD	ED	Pessimistic %	Neither	%
1	High resource requirements	20	34	37	84	5	3	3	10	6	6
2	High power consumption (process, resources, and the product itself)	21	37	31	82	7	4	2	12	6	6
3	Poor software design (architectural, logical, physical, and user interface)	22	39	28	82	4	3	3	9	9	8
4	Lack of ICTs for coordination and communication	19	39	29	81	7	6	2	14	6	6
5	Lack of social and ethical responsibility	25	33	29	81	3	5	5	12	8	7
6	Lack of green software development knowledge	38	25	22	79	7	1	4	11	11	10

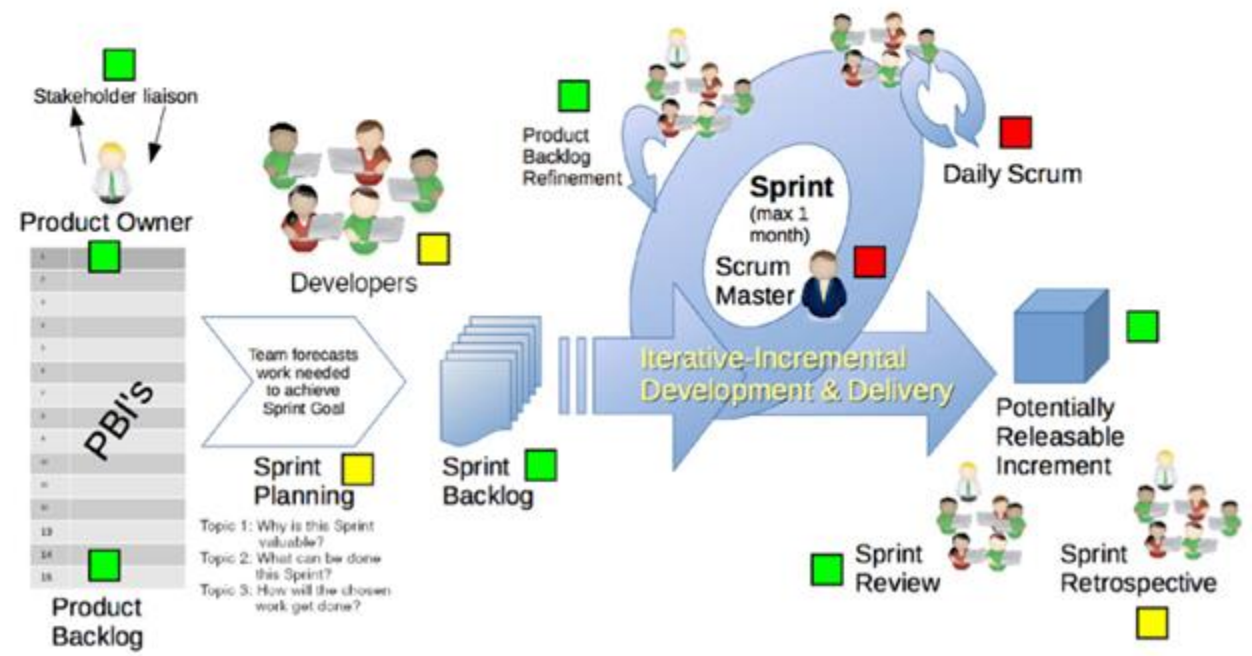
S.NO.	Success Factor	N=74	%
SF1	Green software design and efficient coding	57	77
SF2	Power-saving software strategies	55	74
SF3	Low carbon emission throughout the software development process	45	60
SF4	Efficient resources utilization	44	59

Salam, M., & Khan, S. U. (2016). Developing green and sustainable software: Success factors for vendors.
Salam, M., & Khan, S. U. (2018). Challenges in the development of green and sustainable software for software multisourcing vendors

In software engineering level, the focus is on understanding what aspects should be considered and who should be doing that.



Georgiou, S., Rizou, S., & Spinellis, D. (2019). **Software development lifecycle for energy efficiency: techniques and tools**.



Bambazek, P., Groher, I., & Seyff, N. (2022). **Sustainability in agile software development: A survey study among practitioners**.

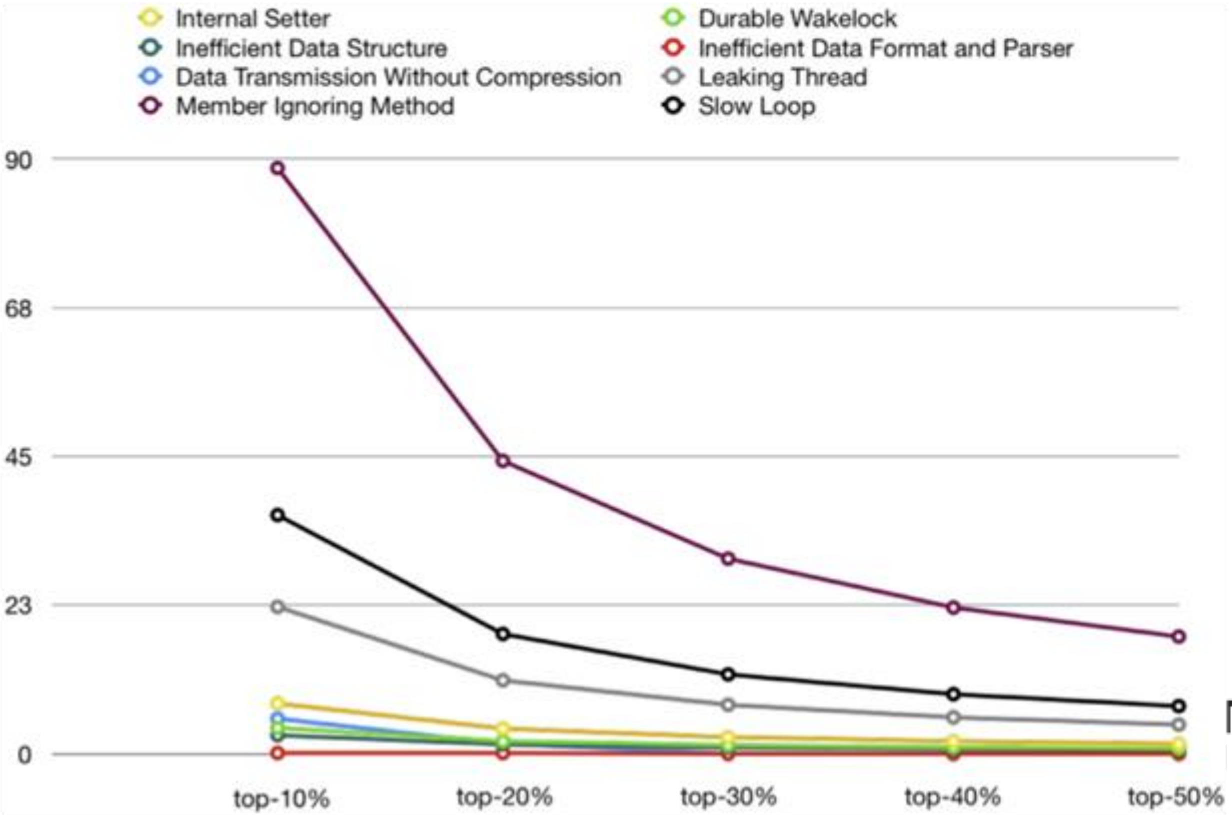
In green coding, the focus has been on comparing (benchmarking) rather specific problems in some predefined environments

- Generalization of the results is challenging

	Energy (J)
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13

	Time (ms)
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52



Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A., & De Lucia, A. (2019). **On the impact of code smells on the energy consumption of mobile applications**

Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). **Ranking programming languages by energy efficiency.**

contributed articles

DOI:10.1145/2714560

This framework addresses the environmental dimension of software performance, as applied here by a paper mill and a car-sharing service.

BY PATRICIA LAGO, SEDEF AKINLI KOÇAK, IVICA CRNKOVIC, AND BIRGIT PENZENSTADLER

Framing Sustainability as a Property of Software Quality

SUSTAINABILITY IS DEFINED as the "capacity to endure"[34] and "preserve the function of a system over an extended period of time."[13] Discussing sustainability consequently requires a concrete system (such as a specific software system) or a specific software-intensive system. Analysis of the sustainability of a specific software system requires software developers weigh four major dimensions of sustainability—economic, social, environmental, and technical—affecting their related trade-offs.[32]

The first three stem from the Brundtland report,[4] whereas technical is added for software-intensive systems[27] at a level of abstraction closer to implementation. The economic dimension is concerned with preserving capital and value. The social dimension is concerned with maintaining communities. The environmental dimension seeks to improve human welfare by protecting natural resources. And the technical dimension is concerned with supporting long-term use and evolution of software-intensive systems. Sustainability is achievable only when accounting for all dimensions. Including the environmental dimension makes it possible to aim at dematerializing production and consumption processes to save natural resources.[11] Connections among the four dimensions involve different dependencies and stakeholders.[28,31] Potential conflicts among stakeholder interests means software developers must understand the relationships among goals of the four dimensions.

The shortcoming of current software engineering practice with regard to sustainability is that the technical and economic dimensions are taken into account while the environmental and social dimensions are not. The question we address here is how these concepts relate to software and how to break down the respective concerns into software-quality requirements. We focus on the (currently neglected) environmental dimension and its relation to the other dimensions. While most efforts in environmental sustainability through software have focused on energy efficiency, we tie the concept of environmental sustainability to other sustainability dimensions of a software system, particularly to ad-

» key insights

- **The sustainability analysis framework enables software developers to specifically consider environmental and social dimensions relative to technical and economic dimensions.**
- **Sustainability requirements and concerns will increase system scope, requiring extended analysis during requirements engineering.**
- **The framework helps draw a more comprehensive picture of the relevant quality dimensions and, as a result, improve decision making.**

"LIKE PERFORMANCE, RELIABILITY, SECURITY,

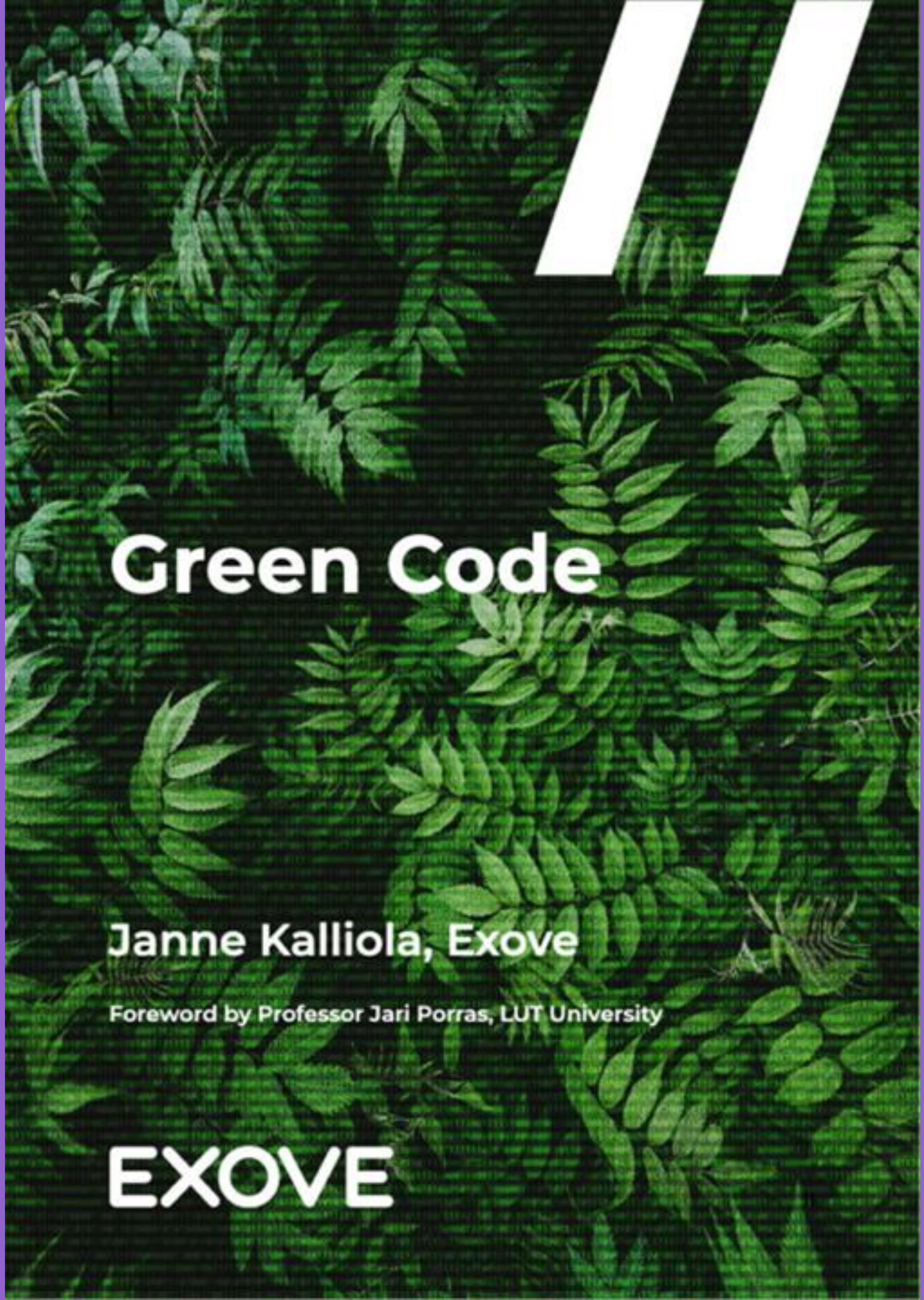
SUSTAINABILITY DOES NOT JUST HAPPEN

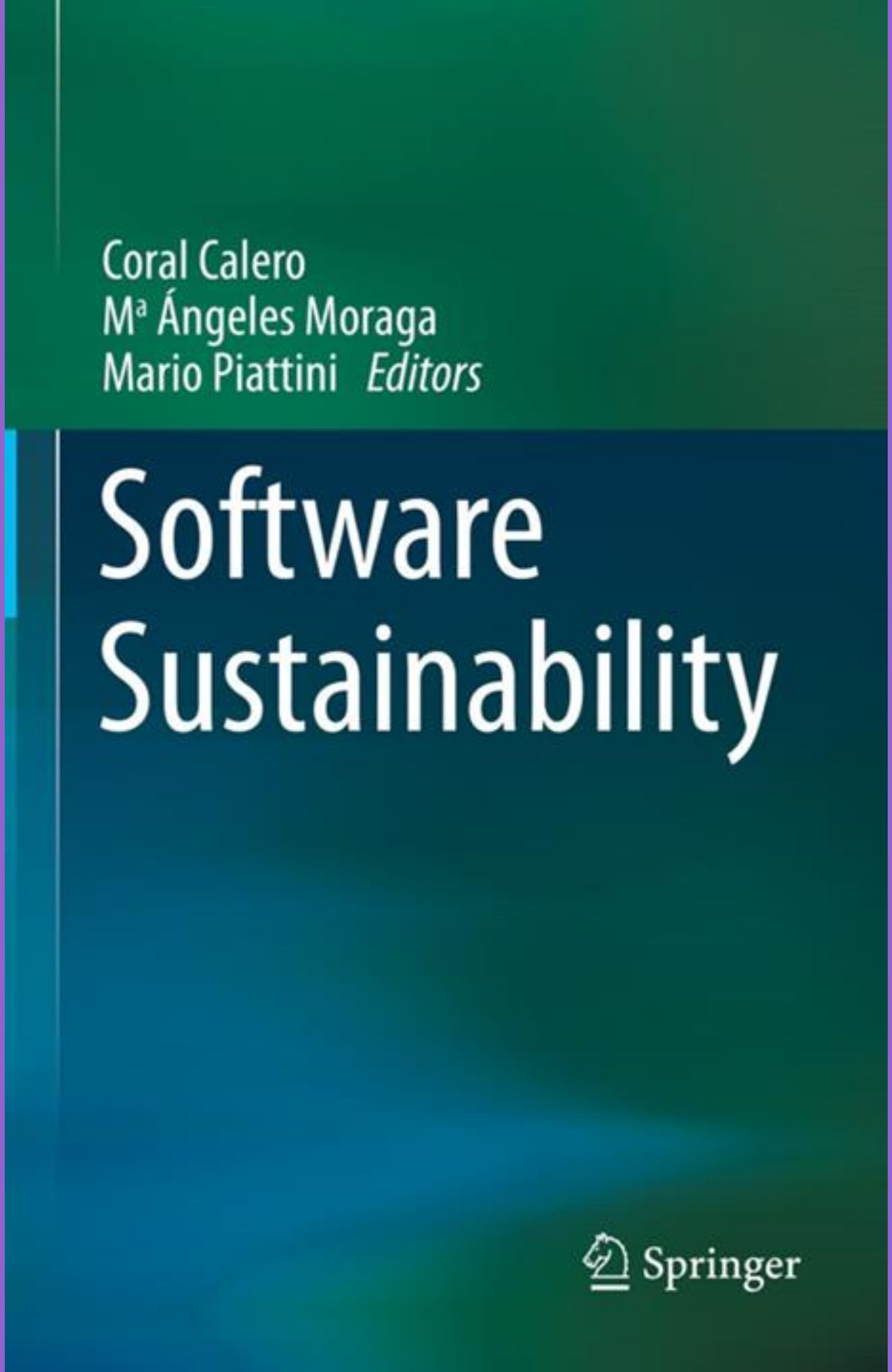
UNLESS WE PLAN FOR IT."

Patricia Lago @ 2016 Inaugural speech

High level approach

- Raiders of lost efficiency, i.e. waste, e.g.
  - Algorithmic inefficiency
  - Non-optimized data
  - Non-optimized communication
- Solutions, e.g.
  - Minimize transferred data
  - Reduce code





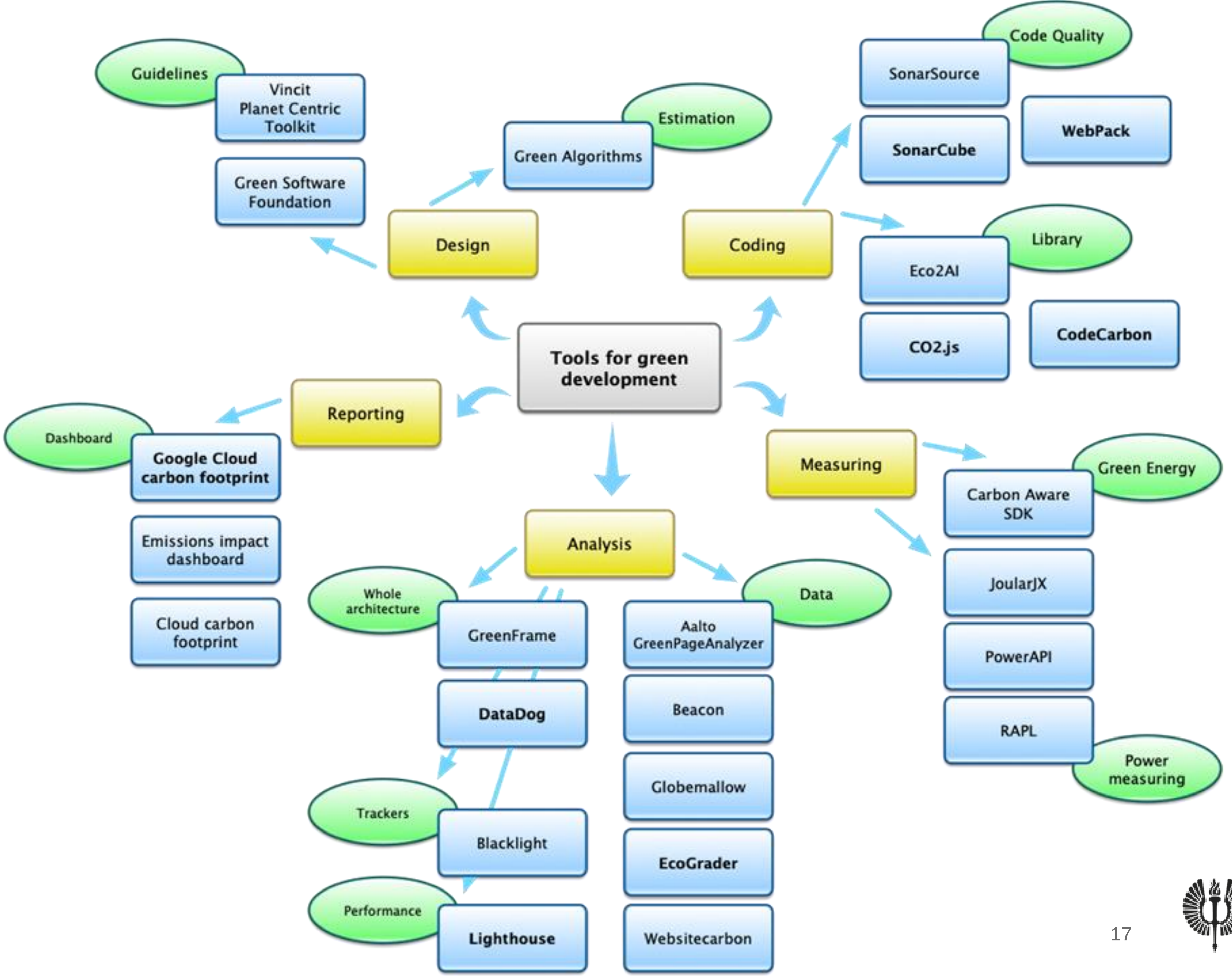
Holistic view

- Already some years old book but gives a nice perspective on software sustainability on all levels

Final thoughts on green coding

- Topic is broad and complex, therefore difficult to teach in just one course
- There is no simple ”make it green” command or statement in a programming language
- Instead, one must understand and use a wide selection of tools and techniques to make code/software/service greener

- On the other hand making code green is in the end making sre your code works efficiently (good quality)



UNIVERSITY OF TURKU

It’s Simple!
Green Coding in Practice

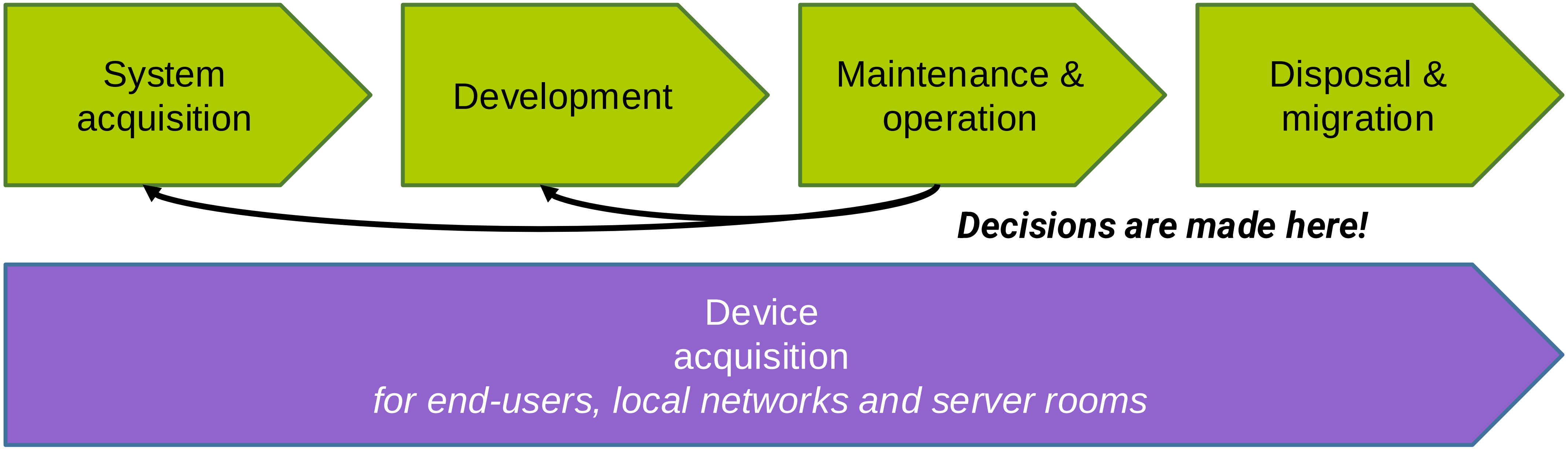
MitViDi
Mittarit vihreän digitalisaation julkisiin ICT-hankintoihin

Decisions that affect the whole lifecycle

Organizational practices & emissions

Carbon emissions through <u>power consumption</u>

Re-use of code and portability of data



Power efficiency,
working life,
material choises,
manufacting emission

MitViDi
Mittarit vihreän digitalisaation julkisiin ICT-hankintoihin

Maintenance & operation

Power consumption =

Front-end: Measure and model based on usage data

+

Internet / network: Model based on network traffic

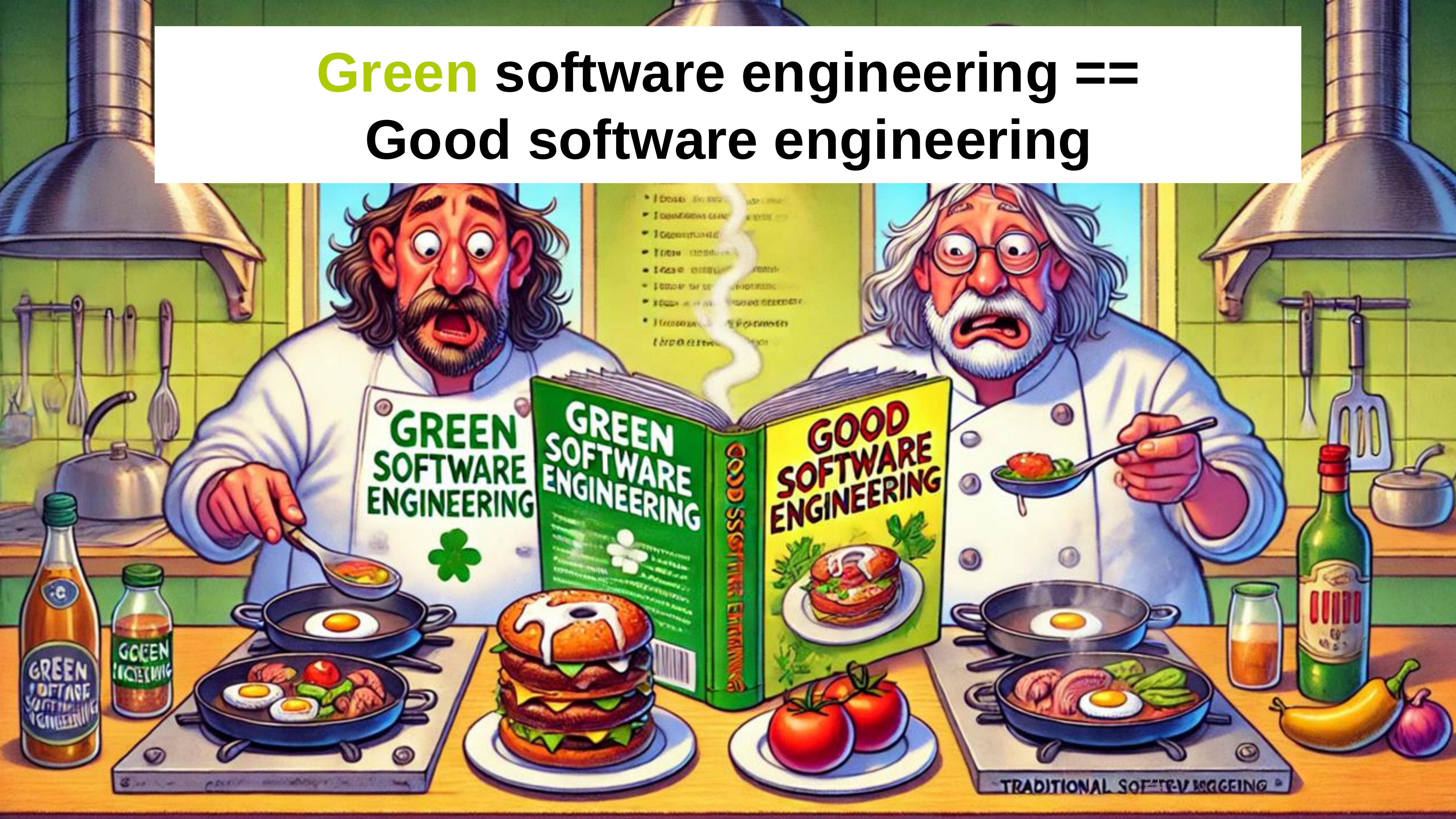
+

Back-end / own servers: Measure or model based on used resources

+

Back-end / cloud servers: Model based on used resources / invoicing

Green software engineering == Good software engineering



GREEN SOFTWARE LIFE CYCLE

Kivimäki et al. (2024)

Example criteria, indicators, and metrics

7 No unnecessary leftover data, usable data must be simple to transfer

6 Maintaining environmental criteria, metrics of usage [30], qualitative indicators of user experience [22], energy efficiency of other services required for software to function, avoidance of feature creep and requirements bloat [24]

5 Lines of code, technical debt, code smells, found and fixed defects, defect density, project estimates vs reality, validating environmental criteria [11][26]

1 Environmental certificates and standards [24], life cycle costs, comparing reduction in energy consumption and fit for purpose, reflectiveness, return on green investment, feasibility [22][30][47]

2 General environmental criteria, energy consumption, hardware requirements [26], environmentally friendly functionalities [24]

3 Modular and lasting software, supports sustainable use by default, software and its data are portable and transparent [24], effective GUI design [11]

4 Energy efficiency [22], readability of documentation and code, efficiency of algorithms and architecture [3][30], maintaining sustainability requirements in practice, sustainable development practices, minimal waste during development [47]

UNIVERSITY OF TURKU

1. Acquisition

Evaluation of sustainability debt and/or organizational maturity

Indicators:

- Environmental certificates and standard
- Life cycle costs
- Comparing reduction in energy consumption and fit for purpose
- Reflectiveness
- Return on green investment
- Feasibility

2. Requirements

Demands on sustainability stakeholders

Indicators:

- General environmental criteria
- Energy consumption
- Hardware requirements
- Environmentally friendly functionalities

3. Design

Simplicity and clarity

Indicators:

- Modular and lasting software
- Supports sustainable use by default
- Software and its data are portable and transparent
- Effective GUI design

4. Construction

Optimization of sustainability during development

Indicators:

- <u>Energy efficiency</u>
- Readability of documentation and code
- Efficiency of algorithms and architecture
- Maintaining sustainability requirements in practice
- <u>Sustainable development practices</u>
- Minimal waste during development

5. Testing

Efficient testing incorporating environmental requirements

Indicators:

- Lines of code
- <u>Technical debt</u>
- Code smells
- Found and fixed defects
- Defect density
- Project estimates vs reality
- <u>Validating environmental criteria</u>

6. Maintenance & Operation

Sustainable usage, strict change control

Indicators:

- Maintaining environmental criteria
- Metrics of usage
- Qualitative indicators of user experience
- Energy efficiency of other services required for software to function
- Avoidance of feature creep and requirements bloat

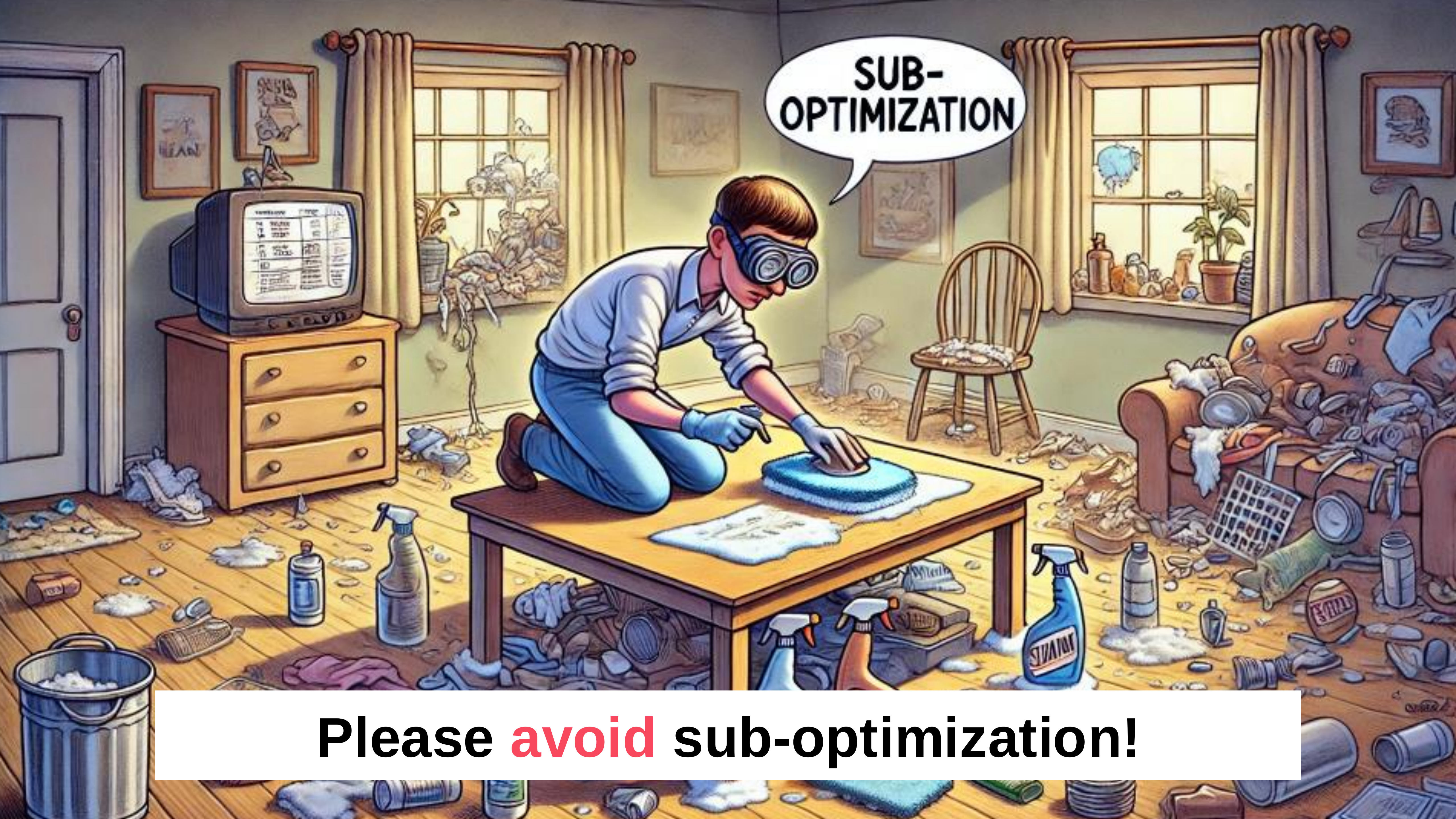
7. Disposal

Cleanly removable

Indicators:

- No unnnecessary leftover data
- Usable data must be simple to transfer

UNIVERSITY
OF TURKU



Please avoid sub-optimization!

Four Core Engineering Practices for Greener Software

1. <u>Implement</u> functionalities supporting green use and operations
2. <u>Follow</u> architectural patterns supporting green development and maintenance throughout the software life-cycle
3. <u>Optimize</u> the software based on measurements
   a. *Technical optimization* (green coding) – optimize the algorithmic and operational efficiency
   b. *User experience optimization* (green design) – optimize the usage effiency and screen time for different end-user roles
4. <u>Support</u> hand-print effects through (green?) service design and requirements practices

UNIVERSITY OF TURKU

Four Core Engineering Practices for Greener Software

1. Implement functionalities supporting green use and operations
2. Follow architectural patterns supporting green development and maintenance throughout the software life-cycle
3. Optimize the software based on measurements
   a. *Technical optimization* (green coding) – optimize the algorithmic and operational efficiency
   b. *User experience optimization* (green design) – optimize the usage effiency and screen time for different end-user roles
4. Support hand-print effects through (green?) service design and requirements practices

UNIVERSITY OF TURKU

It’s hard to improve something you cannot see!

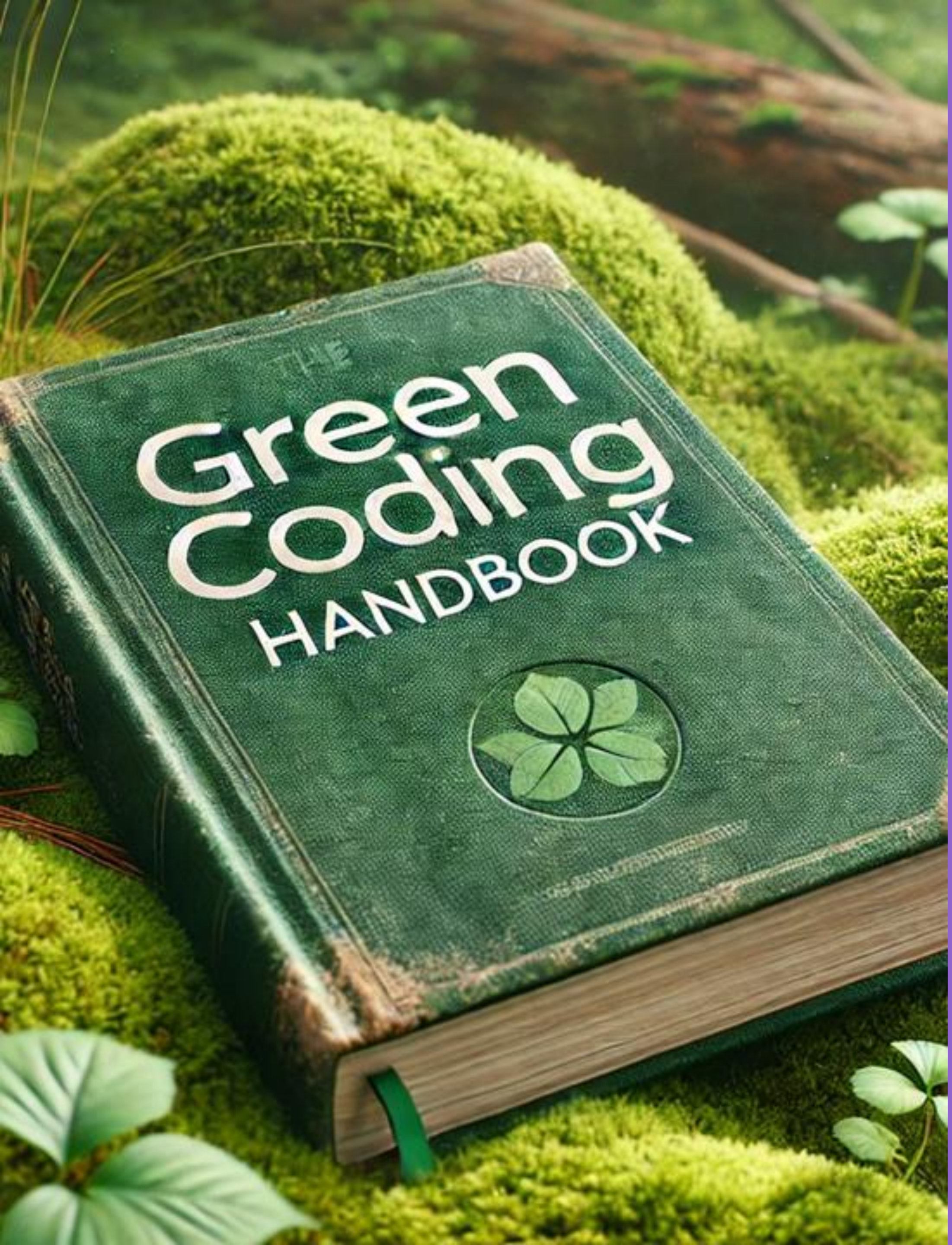
Developer Level
Green Optimization Loops

Technical optimization:

1. **Design** optimizations
2. **Implement** and refactor
3. **Measure** change in *energy consumption*
4. **Repeat** from 1

UX optimization:

1. **Design** optimizations
2. **Implement** and streamline
3. **Measure** change in *task times*
4. **Repeat** from 1



Green Coding Handbook

https://tinyurl.com/greencoding2

*For personal use only

Chapters

1. Introduction
2. Green programming practices
3. Green UI/UX design
4. Measurements of power consumption
5. Code optimization
6. Green cloud services

Introduction

- Defines green programming
- Indicators of greenness
  - **Carbon footprint**
- Anatomy of a computer
  - **To understand the parts consuming energy**
- Introduction to energy consumption and measurements
- A few observations on the carbon footprint of the development process

UNIVERSITY OF TURKU

Green programming practices

- Languages
  - Not all languages are equal.
    - **C << Perl, JRuby**
    - **WebAssembly-C < JavaScript**
  - But there are limitation what we can use
- Frameworks effect
  - Do we even need one?
- A few selected practices
  - Mostly concerning web apps
    - **Unoptimized data handling**

UNIVERSITY OF TURKU

Green UI/UX design

- Main points:
  - User interface creation with energy efficiency in mind

Measurements of power consumption

- Can we call our software green if we don't know its energy consumption?
- Or how can we optimize if we can measure the change?
- The most important topic on the course
  - Without the measurements, we hardly have any idea about the green-ness

Code optimization

- A broad and difficult topic
- Use common sense i.e. non-pessimization
- How to assess code quality?
- A couple of optimization examples

UNIVERSITY OF TURKU

Green cloud services

- If we consider the whole life cycle of an application, a large part of the energy consumption takes place during its use
- Often during the development of an application, decisions have to be taken that affect the environment in which it is deployed to
- Therefore it is beneficial to understand how to compare different services

UNIVERSITY OF TURKU

Green Coding Best Practises (from handbook)

1 Unoptimized data handeling in web

☞ Refer Chapter 2, Section 2.2, Page 22

- 12 javascript examples of handling data in an **energy-efficient way** in web applications.
- Some examples:
  - Reducing data transfer quality
  - Data compression before transmission
  - Identifying immutable data
  - Transmitting only changed data

2 Green UI/UX design

☞ Refer Chapter 3, Page 20

- **Avoiding user errors**: minimizing the interaction the user has, only the essential.
- **Avoiding the use of dark patterns**: Dark patterns are UI/UX design tricks that are intentionally made to distract or mislead. E.g.: Cookie banners.
- **Eliminating unnecessary elements**. E.g.: on-mouse-hover effects
- **Streamlining functionality**: Prioritizing/highlighting the most used features in the design. E.g.: Menus/ Search bar

3 Code optimization

☞ Refer Chapter 5, Section 5.1 and 5.2, Page 30

- Non-pessimization - Just don’t write bad code!
- Algorithm design
  - Big O
- Dependencies in code
  - 4 types: True, Anti, Output, Input Dependencies.
- Loop interchange
- Parallel loops
  - DISTRIBUTED Loop
  - DOALL Parallelism
- Using AI in code optimization:
  - Github Co-Pilot

4 Green cloud services

☞ Refer Chapter 5, Section 5.1 and 5.2, Page 30

- Google Cloud Services - Carbon Footprint
- Microsoft Azure - Emissions Impact Dashboard
- Amazon Web Services (AWS) - Customer Carbon Footprint Tool36

But are clouds actually green?

- Data-deduplication
- Cloud overflow