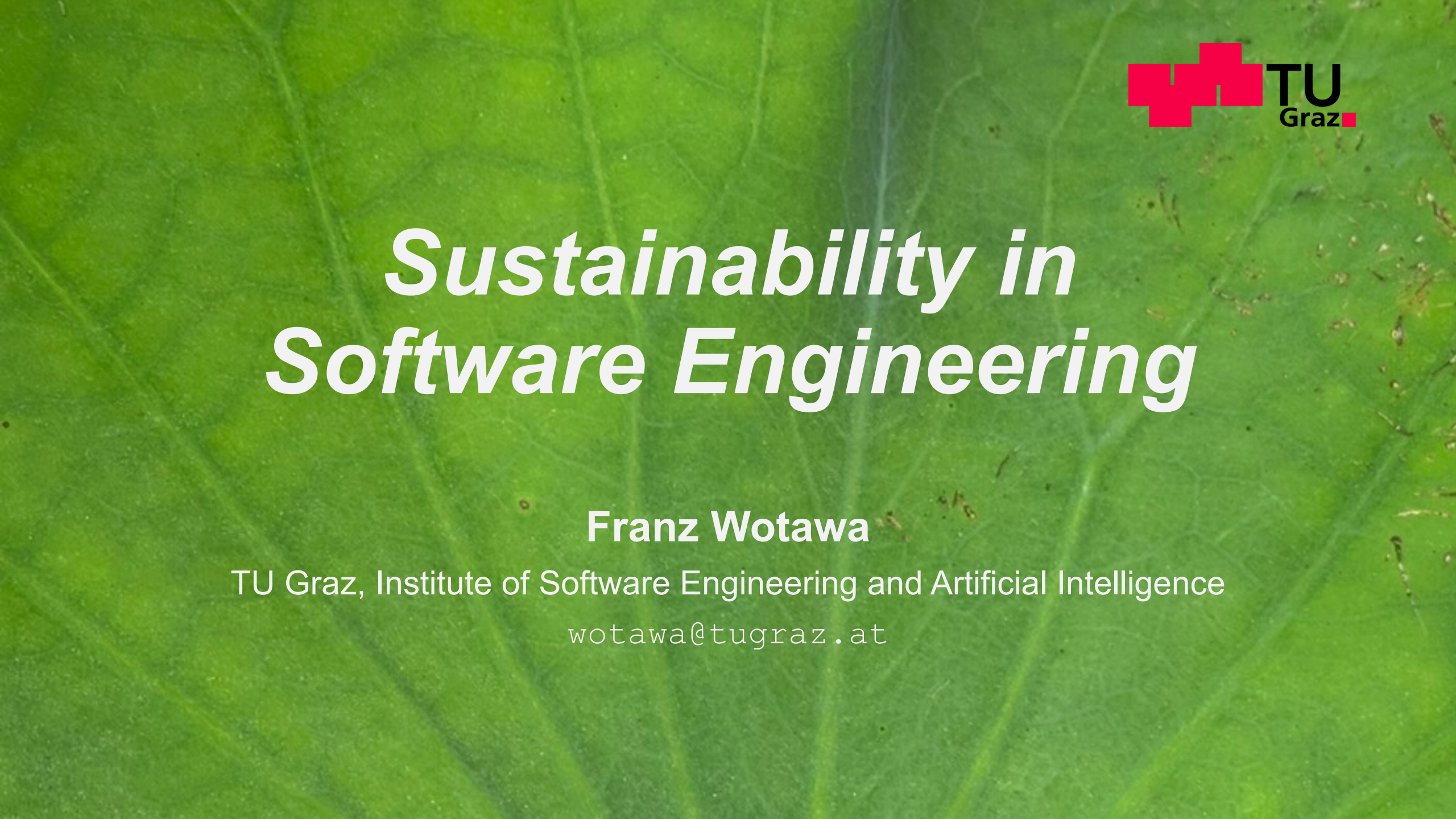


Sustainability in Software Engineering

Franz Wotawa

TU Graz, Institute of Software Engineering and Artificial Intelligence

wotawa@tugraz.at

Motivation

- The energy consumption of computers depends on the hardware AND the running software
- Example: Function `isElement(a)`
  - Checking the content of variable `a` to be an element of a collection (list,…)

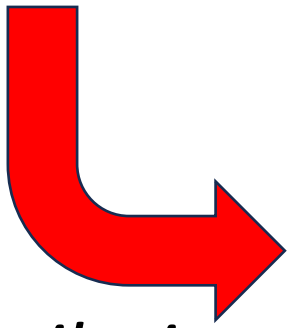
```
isElement(a) {
   for i=1 to length(A) do:
      if (a==A[i]) then return true;
   end for;
   return false; }
```

O(n) n=length(A)

```
isElement(a) {
   if (a==A[hash(a)]) then
       return true;
   else
       return false; }
```

O(1)

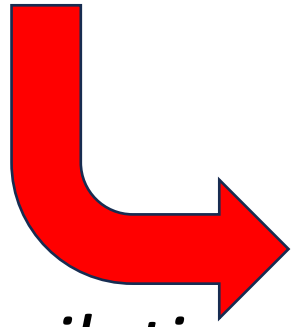
```
isElement(a) {
   for i=1 to length(A) do:
      if (a==A[i]) then return true;
   end for;
   return false;
```



compilation

```
A:
.zero 4000
isElement(int):
push rbp
mov rbp, rsp
mov DWORD PTR [rbp-20], edi
mov DWORD PTR [rbp-4], 0
jmp .L2
.L5:
mov eax, DWORD PTR [rbp-4]
cdqe
mov eax, DWORD PTR A[0+rax*4]
cmp DWORD PTR [rbp-20], eax
jne .L3
mov eax, 1
jmp .L4
.L3:
ret
```

```
isElement(a) {
   if (a==A[hash(a)]) then
      return true;
   else
      return false
```



compilation

```
A:
.zero 4000
hash(int):
push rbp
.....
pop rbp
ret
isElement(int):
push rbp
mov rbp, rsp
sub rsp, 8
mov DWORD PTR [rbp-4], edi
mov eax, DWORD PTR [rbp-4]
mov edi, eax
call hash(int)
cdqe
mov eax, DWORD PTR A[0+rax*4]
cmp DWORD PTR [rbp-4], eax

al

1

0
.L5:
leave
ret
```

COMPARE RESOURCES REQUIRED AFTER BEING EXECUTED!

Motivation

- Runtime and energy consumption depend on the number of statements to be processed:
  - The proper data structure for the right job matters
  - Need to reduce the overall number of statements to be executed
- Need to measure energy consumption (have an example later)
- But, is this all?

Objectives

- Answer the following questions:
  - Why does sustainability matter for programs (and software engineering)
  - What is sustainable software engineering?
  - How can we support sustainability in the case of software engineering?

- The focus is on software engineering, not the use of software to improve sustainability

Sustainability and software

- Important to note
  - Software is becoming increasingly important
    - E.g., see Goldman Sachs, Software Is Taking Over the Auto Industry, Nov. 2022 (https://www.goldmansachs.com/intelligence/pages/software-is-taking-over-the-auto-industry.html)
  - ICT accounts for about 2-4% of all global GHG emissions, and software is the backbone
  - When developing software that has a low energy footprint, we are able to save GHG emissions!
- United Nations (UN)'s Brundtland report defines **sustainable development** as the ability to "meet the needs of the present without compromising the ability of future generations to satisfy their own needs". According to the UN, sustainable development needs to satisfy the requirements of three dimensions, which are society, the economy, and the environment.

Sustainable Software Engineering

- Microsoft:

  "*Sustainable Software Engineering is an emerging discipline at the intersection of climate science, software, hardware, electricity markets, and data center design. The Principles of Sustainable Software Engineering are a core set of competencies needed to define, build, and run sustainable software applications*."

- Alexander Belokrylov (The Power Of Sustainable Software, Forbes, Aug 18, 2022):

  „*Sustainable software engineering (or sustainable software development) is becoming more popular nowadays. This approach aims to design software that will increase overall application efficiency and reduce energy and space consumption requirements*."

Some definitions

From: Naumann, S., Dick, M., Kern, E. et al.: The GREENSOFT model: a reference model for green and sustainable software and its engineering. SUSCOM 1(4), 294–304 (2011). doi:10. 1016/j.suscom.2011.06.004

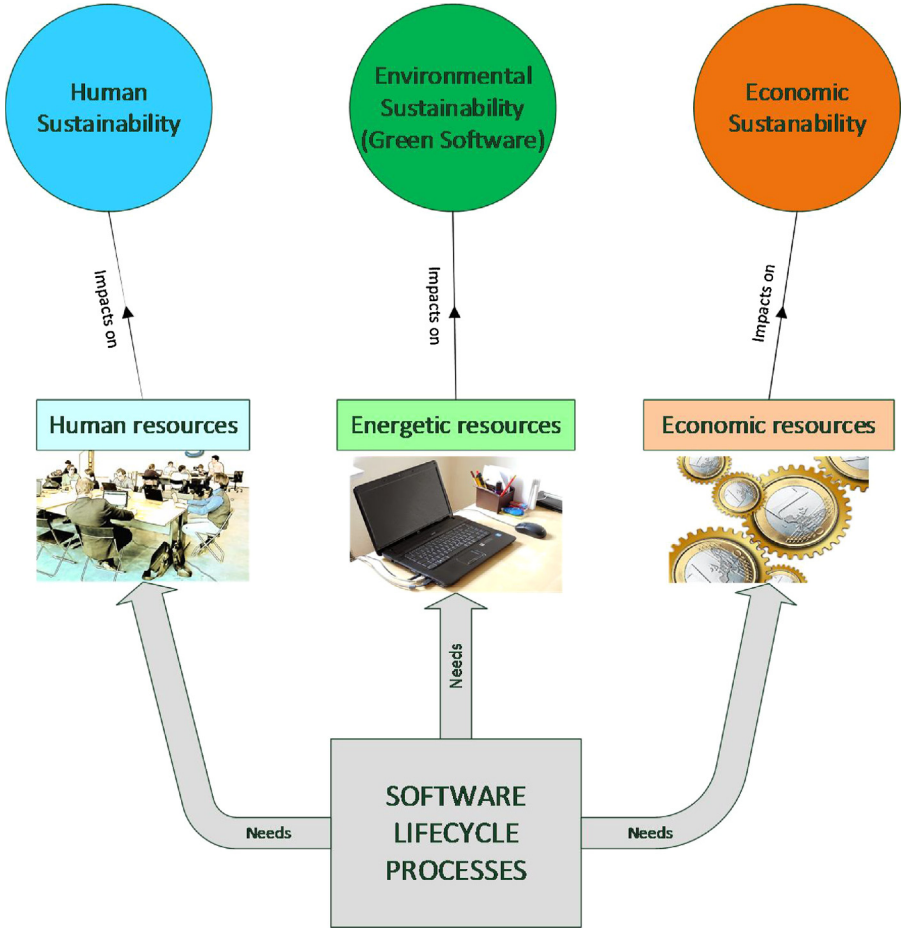
- **Sustainable Software** is software whose development, deployment, and usage results in minimal direct and indirect negative impacts or even positive impacts on the economy, society, human beings, and the environment.
- A prerequisite for sustainable software is a **sustainable development process**, which refers to considering environmental and other impacts during the software life cycle and the pursuit of the goals of sustainable development.
- Building on this, we can define **Sustainable Software Engineering** as the art of developing sustainable software through a sustainable software engineering process.

Note that sustainable software engineering can be seen as an emerging discipline at the intersection of climate science, software, hardware, electricity markets, and data center design

Dimensions of software sustainability

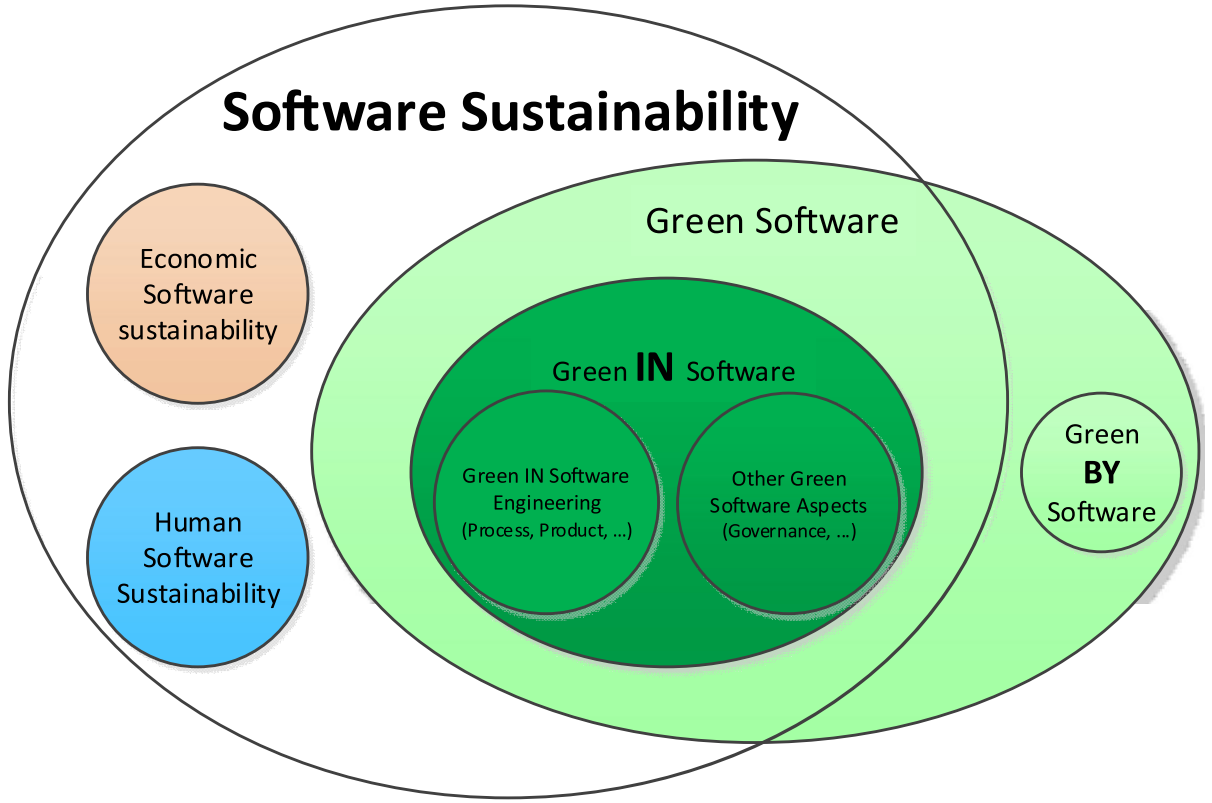
From: C. Calero, M. Piattini: Sustainable Computing: Informatics and Systems 16 (2017) 117–124. https://doi.org/10.1016/j.suscom.2017.10.011

- **Human sustainability**: how software development and maintenance affect the sociological and psychological aspects of the software development community and its individuals. This encompasses topics such as: Labor rights, psychological health, social support, social equity and livability.
- **Economic sustainability**: how the software lifecycle processes protect stakeholders' investments, ensure benefits, reduce risks, and maintain assets.
- **Environmental sustainability**: how software product development, maintenance and use affect energy consumption and the usage of other resources. Environmental sustainability is directly related to a software product characteristic that we call "software green ability."



Software sustainability

- The main goal of Green in Software Engineering is to include green practices in both software development and the other activities that are part of Software Engineering.
- ISO/IEC/IEEE defines software engineering as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software”



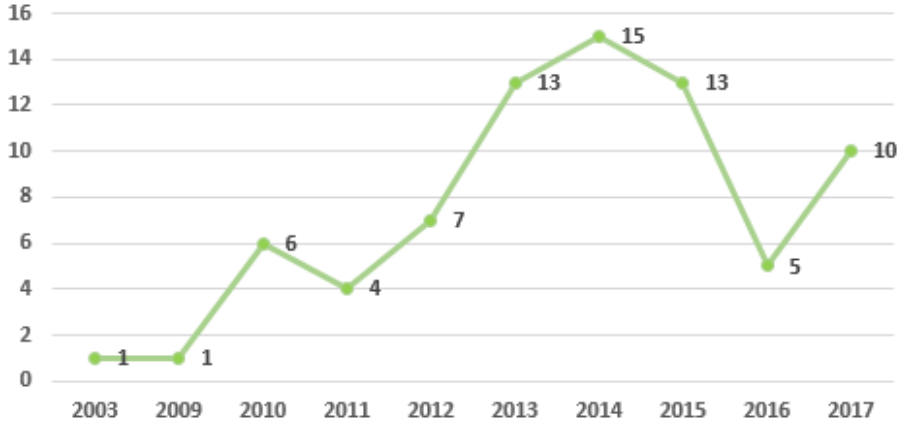
Summary of the discussion so far

- In a closer sense, software sustainability captures ”Green (in) Software” and focuses on the development and maintenance of software that contributes to the reduction of GHG emissions!

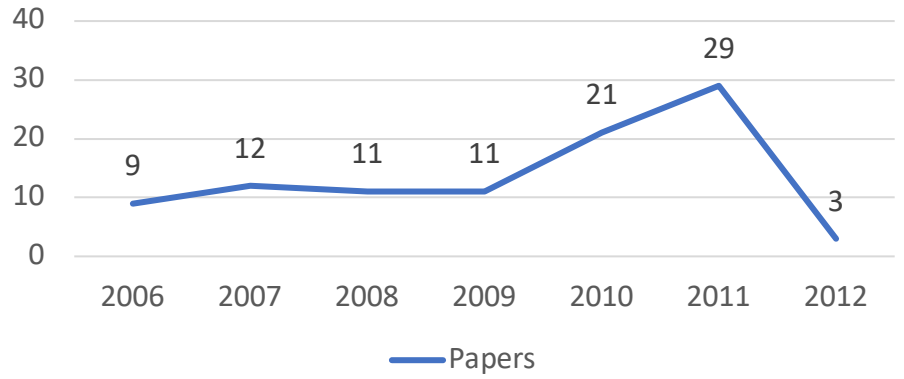
- Reduction during development: e.g., reduce compilations, and minimize relearning in case of machine learning-based software applications
- Reduction during use: Develop programs that have a minimized GHG footprint!
  - Might be more important because of multiple program executions (1 development but *n* executions)

Research impact / Mapping studies and reviews

From: Brunna C. Mourão, Leila Karita, Ivan do Carmo Machado, Green and Sustainable Software Engineering - a Systematic Mapping Study, SBQS, October 17–19, 2018, Curitiba, Brazil, https://doi.org/10.1145/3275245.3275258



From: Birgit Penzenstadler, Veronika Bauer, Coral Calero, Xavier Franch, Sustainability in Software Engineering: A Systematic Literature Review, in Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE), Jan 2012, Review, DOI: 10.1049/ic.2012.0004

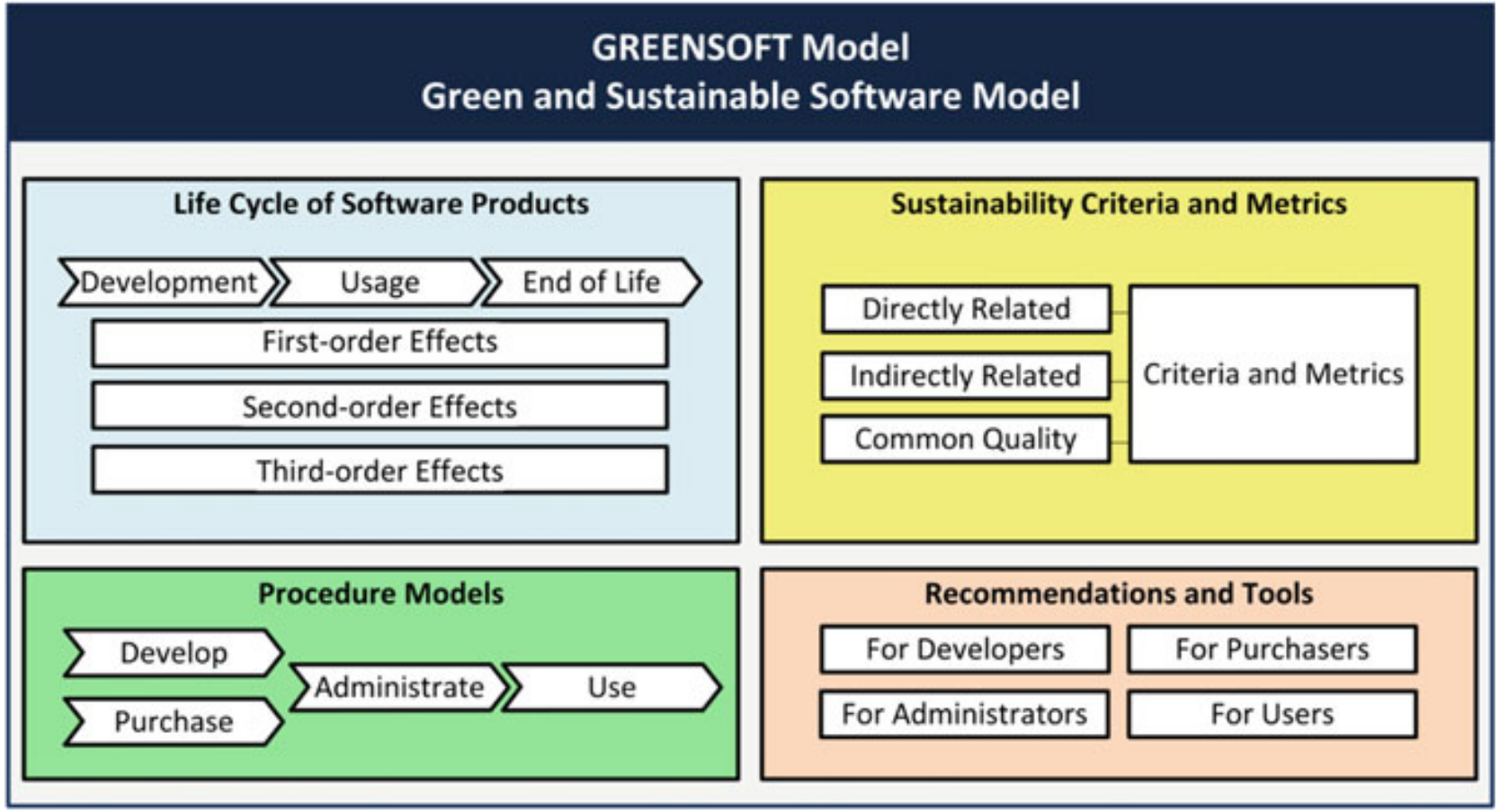


The GREENSOFT Model

From: Stefan Naumann, Eva Kern, Markus Dick and Timo Johann, Sustainable Software Engineering: Process and Quality Models, Life Cycle, and Social Aspects, in L.M. Hilty and B. Aebischer (eds.), ICT Innovations for Sustainability, Advances in Intelligent Systems and Computing 310, DOI 10.1007/978-3-319-09228-7_11

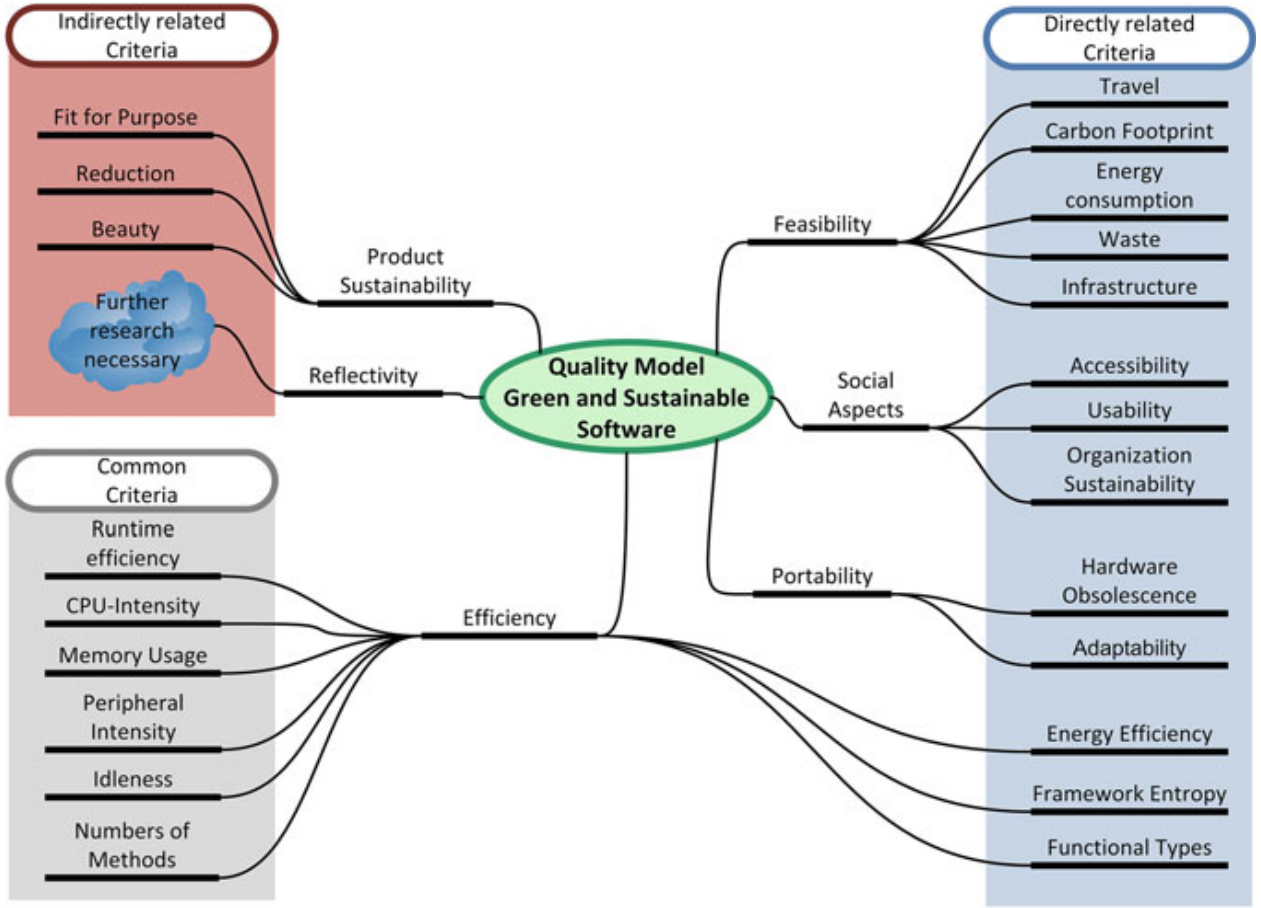
- A sustainable software product ideally meets three conditions:
  - The software is produced in a way that meets sustainability objectives.
  - The software has minimal negative social and environmental impacts during its usage (first-order effects).
  - The software functionality reinforces sustainable development or at least has no negative impacts on society or the environment (second-order and systemic effects).

The GREENSOFT Model (cont.)



A quality model for sustainable software

- To decide whether or not software is sustainable, appropriate criteria are required
- Quality aspects: Efficiency, Reusability, Modifiability, and Usability (extended to sustainability)



Process Models of Sustainable Software Engineering (1) Process-Centric Software Sustainability

- **Sustainability Management Process**. The management process includes a preliminary phase, a planning phase, a monitoring phase, and a supplier sustainability control. In the preliminary phase, the principles and criteria for sustainability are established. In the planning phase, sustainability activities of the development process are indicated, and the corresponding requirements and necessary resources are planned. Afterward, the sustainability of the deployed activities and their conformity with the requirements are monitored. The last part of the management process (supplier sustainability control) deals with sustainability policies and supply and service requirements. Here, an agreement must be reached, and the supplier's sustainability needs to be monitored.
- **Sustainability Engineering Process**. The engineering process concerns suitable tools and methods to enable and support a sustainable development process. In this context, sustainable issues and green principles for development are defined, applied, and analyzed. Energy and resource consumption are factors that impact sustainability and, thus, should be identified at the start of the engineering process. In the next step, the impacts of these effects should be analyzed in order to set sustainability objectives for the development process subsequently. In addition to the impacts of the process itself, the impacts of change requests on sustainability should be determined.
- **Sustainability Qualification Process**. The qualification process applies to external resources such as engineering and management support tools. Aimed at sustainable products, these external resources need to be sustainable as well. In order to ensure their quality, a qualification strategy, an implementation plan for the strategy, documentation of the outcomes of the qualification, and a qualification report are required.

Process Models of Sustainable Software Engineering (2) Agile and Sustainable Software Engineering

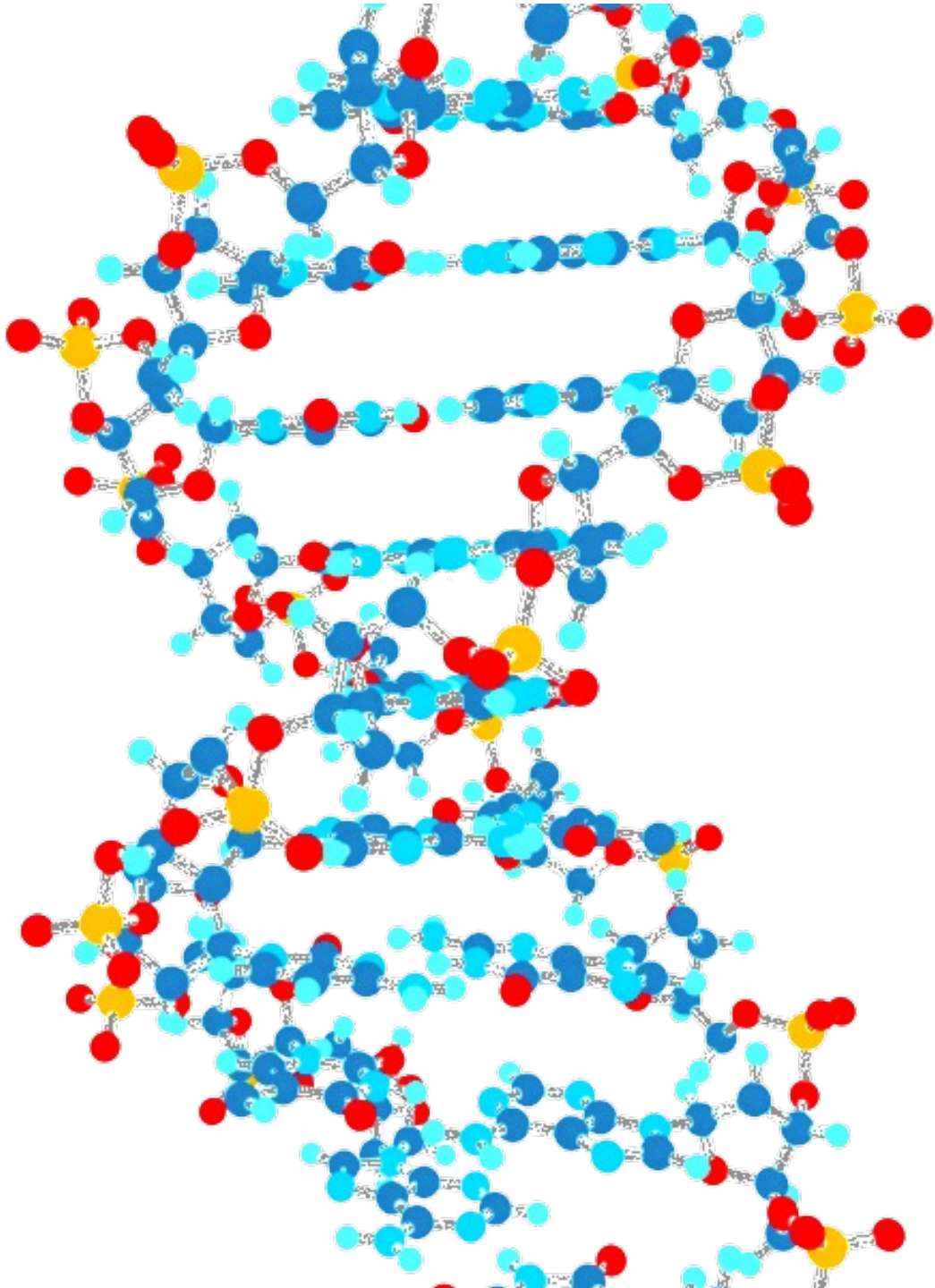
- Not a whole process but an extension
- **Two main activities**:
  - Process Assessment (Focus on the sustainability of the software development process)
    - Process Assessment is a continuous activity alongside the software development process. It is meant to collect and edit data from the process that can be used for a carbon footprint calculation or even for a life cycle assessment
  - Sustainability Reviews and Previews (Focus on the sustainability of the software product)
    - Reviews and Preview meetings are conducted regularly
- **Three roles**: the Customer Representative, the Development Team, and the Sustainability Executive

However, can we do better?

- Focus on the automation of producing sustainable software!

Genetic improvement (GI)

Genetic programming/algorithms applied to sustainable SE

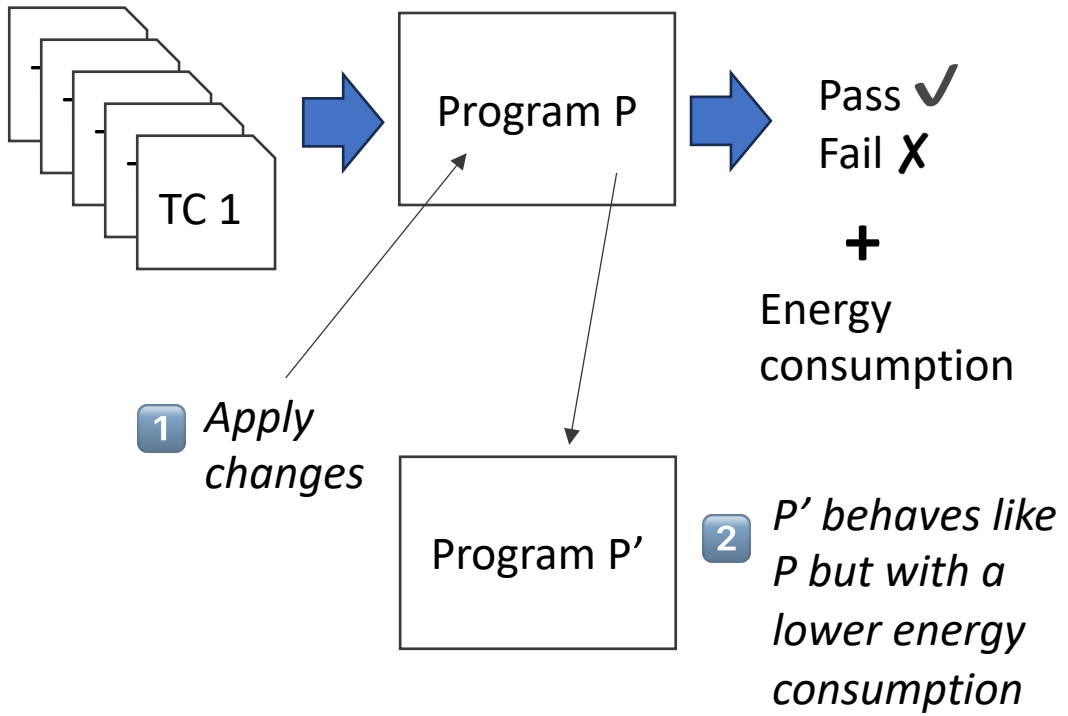


Motivation

- The largest hurdle in producing energy-efficient software is the developer's disconnect between the source code they write and the energy that will be consumed from the compiled product they deliver.
- Without a deep understanding of how a particular compiler works, along with an equally deep understanding of how much energy a given instruction will consume, the problem remains difficult for many developers.
- It has been found that metrics previously believed to guide developers to more energy efficient solutions could be better at doing so.
- Subtle changes, such as introducing inline methods, swapping API implementations, and constructing semantically equivalent (but structurally inequivalent) algorithms, have all been shown to influence energy consumption.
    - However, this influence is difficult to determine outside of the ad hoc and inefficient process of trial and error.
- Tools have to be developed to guide users to energy-inefficient areas of their software.
- Hence, a method of decreasing software's energy consumption lies in automated processes.

Using genetic improvement in sustainable software engineering

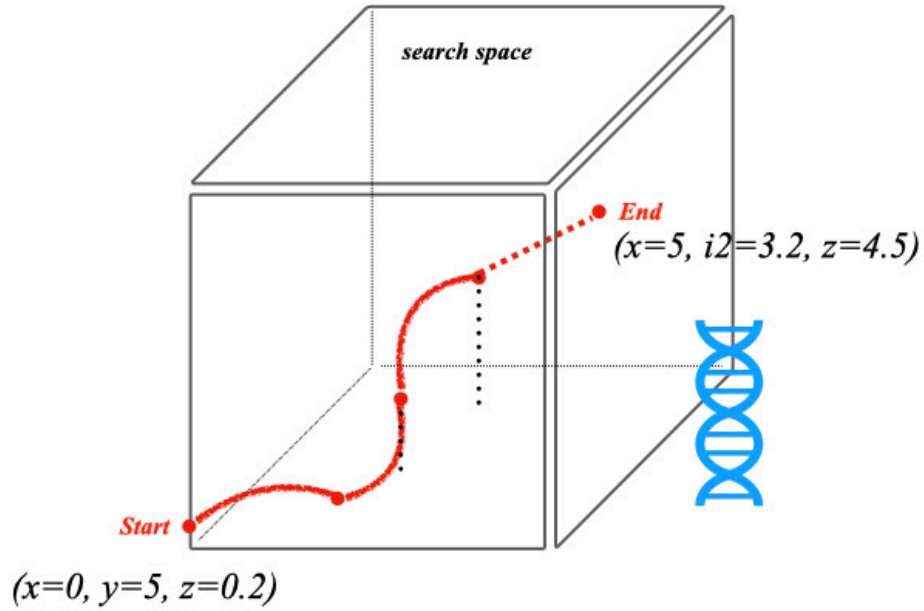
- **Objective**: Optimize the energy consumption of programs but still keep the required functionality
- **Idee**: Change the program until the energy consumption is lowest
- **Potential solution**: Use genetic algorithms/programs for optimization (ako Search-Based Software Engineering)



Genetic programming

- Population where each element has a chromosome
- Apply operators like
  - Selection based on a fitness function
  - Crossover
  - Mutation

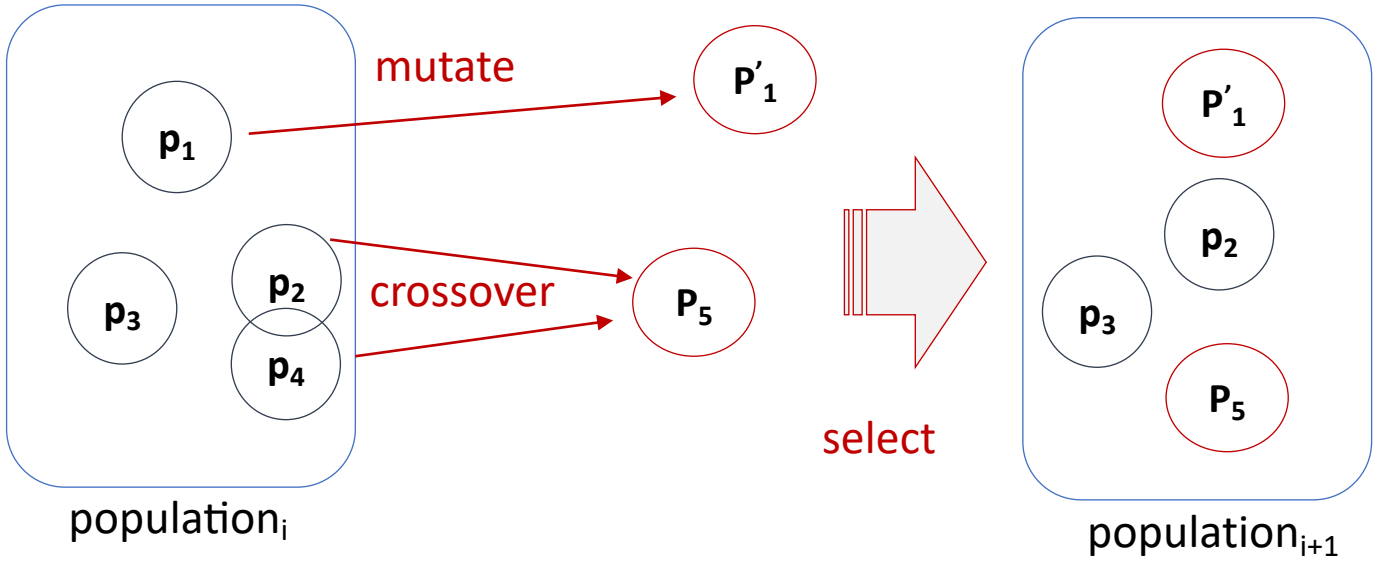
to generate new populations



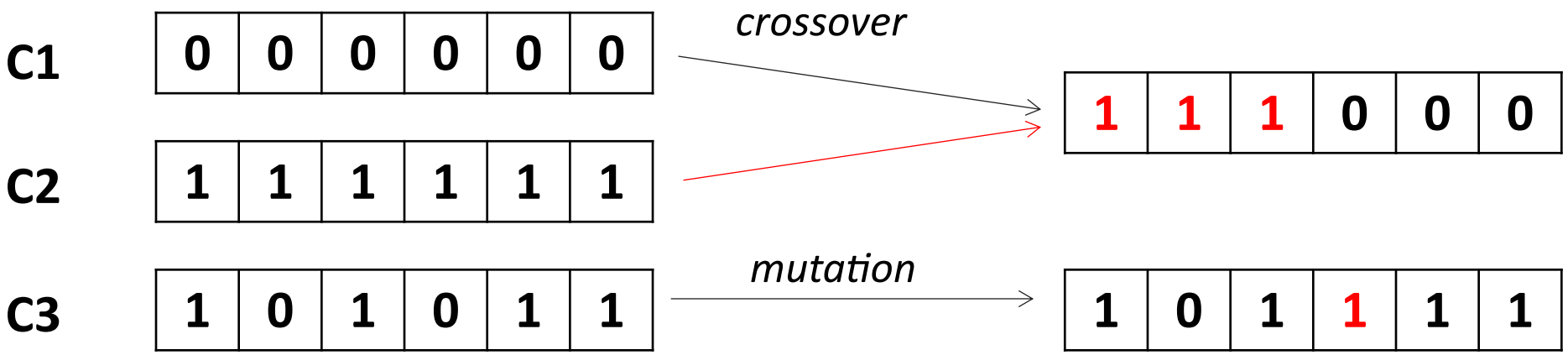
Genetic Search: Using genetic principles to guide search

Questions:
- What are genes?
- What are the operations?

Genetic programming



- Represent problem as chromosomes comprising genes. A set of chromosomes is called a population
- Chromosomes can be stated as strings (or any other collection



Genetic algorithm (GA)

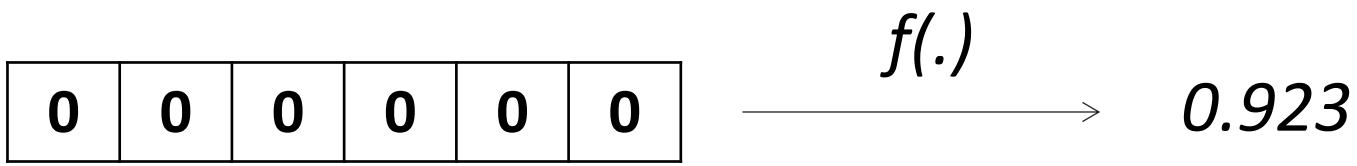
- Crossover:
  - Selection of 2 arbitrary chromosomes
  - Take genes from both to generate a new chromosome

- Mutation:
  - Select 1 arbitrary chromosome
  - Change 1 or more genes for generating a new chromosome

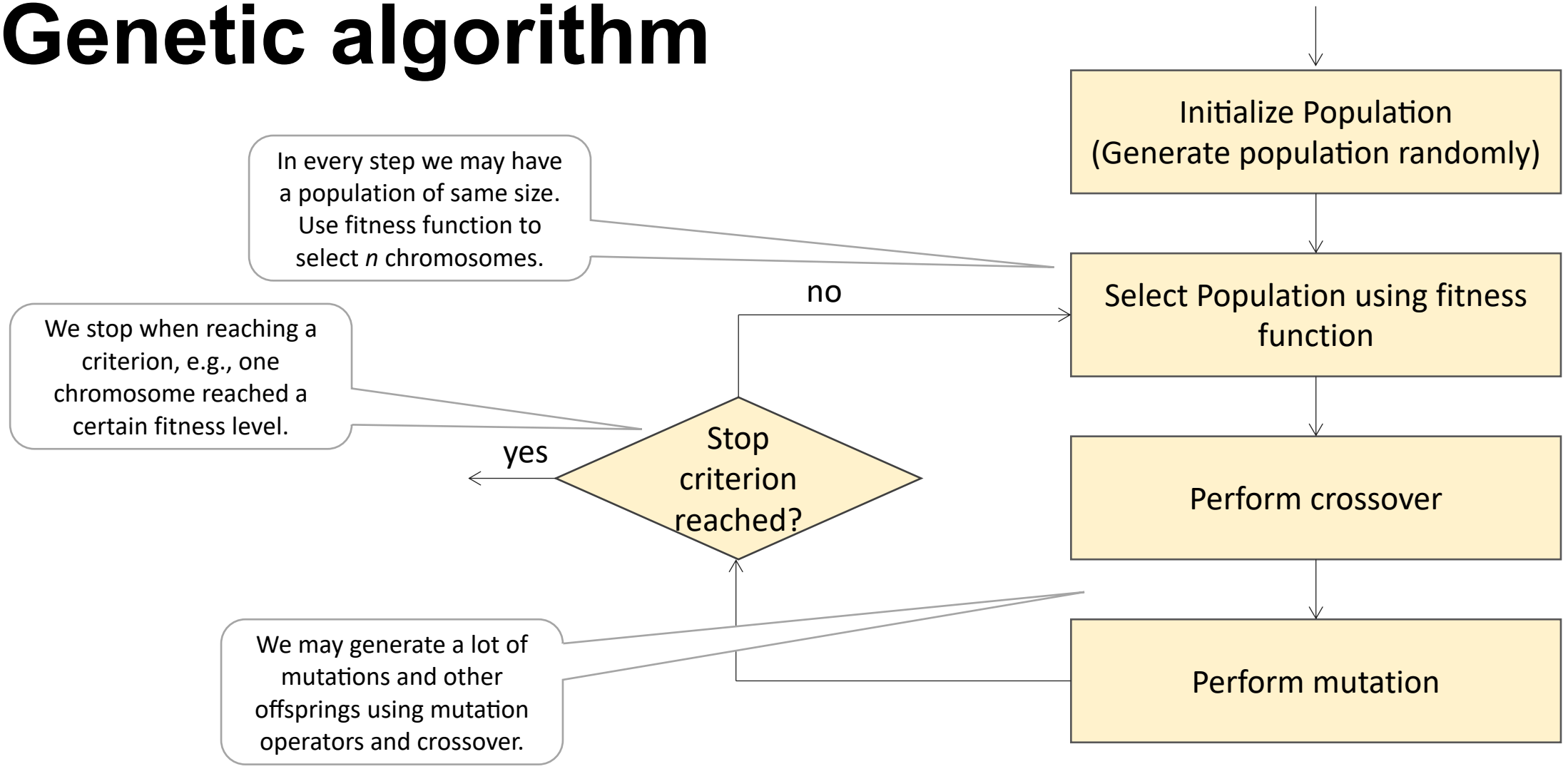
- Selection of chromosomes:
  - Make use of a fitness function

Fitness function

- Maps chromosomes to a particular fitness value



Genetic algorithm



How to use Genetic Algorithms / Programming for reducing energy consumption?

Reducing Energy Consumption Using Genetic Improvement

Bobby R. Bruce
University College London
London
United Kingdom
r.bruce@cs.ucl.ac.uk

Justyna Petke
University College London
London
United Kingdom
j.petke@ucl.ac.uk

Mark Harman
University College London
London
United Kingdom
mark.harman@ucl.ac.uk

ABSTRACT

Genetic Improvement (GI) is an area of Search Based Software Engineering which seeks to improve software's non-functional properties by treating program code as if it were genetic material which is then evolved to produce more optimal solutions. Hitherto, the majority of focus has been on optimising program's execution time which, though important, is only one of many non-functional targets. The growth in mobile computing, cloud computing infrastructure, and ecological concerns are forcing developers to focus on the energy their software consumes. We report on investigations into using GI to automatically find more energy efficient versions of the MiniSAT Boolean satisfiability solver when specialising for three downstream applications. Our results find that GI can successfully be used to reduce energy consumption by up to 25%.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

Keywords

Search based software engineering, SBSE, genetic improvement, GI, optimisation, energy optimisation, energy efficiency, energy consumption, Boolean satisfiability

1. INTRODUCTION

Less than a decade ago the quality of software (outside of end-user design preferences) could broadly be described as the extent to which software met its specification while minimising the prevalence of bugs and usage of traditional computer resources such as CPU time and memory allocation. The growth in two new technologies, mobile computing devices and cloud services, has led to a new environment for software engineers where they must now consider the energy an application consumes; the quality of software is now measured in Joules, as well as bug counts, seconds, and megabytes. At present there are more smartphones in the world than personal computers [22], each containing a limited store of energy between charges that must be used efficiently. The energy required to run large server clusters has grown considerably in the last decade, estimated to be between 1.1% to 1.5% of global electricity consumption in 2010 [26], putting strain on energy suppliers and the budgets of those responsible for purchasing this energy [7]. The total ICT infrastructure generated 1.9% of global CO_2 emissions in 2011 [5] (larger than the entire United Kingdom estimated at 1.47% for the 2010-2014 period [42]) indicating that computer science has a role to play in mitigating climate change.

Thus we believe it important that software engineers find ways of programming computers with energy efficiency in mind to appease the demands from consumers for longer battery life, from companies to reduce their energy bills, and from society's desire to minimise humanity's impact on the environment.

One of the largest hurdles in producing energy-efficient software is the developer's disconnect between the source code they write and the energy that will be consumed from the compiled product they deliver [33]. Without a deep understanding of how a particular compiler works, along with an equally deep understanding of how much energy a given instruction will consume, the problem remains difficult for many developers. It has been found that metrics previously believed to guide developers to more energy efficient solutions are, in reality, poor at doing so [38]. Subtle changes, such as introducing inline methods [41], swapping API implementations [33], and constructing semantically equivalent (but structurally inequivalent) algorithms [8] have all been shown to influence energy consumption. However this influence is difficult to determine outside of the ad hoc and inefficient process of trial-and-error. Tools have been developed to guide users to energy-inefficient areas of their software [2, 11, 30, 19] though the developer retains responsibility for rectifying these inefficiencies.

We suggest that the most under explored method of decreasing software's energy consumption lies in automated processes. Such processes would allow developers to focus solely on meeting the specification requirements with worries about non-functional attributes like energy consumption left to an algorithm capable of refactoring software to a more optimal state.

Genetic Improvement (GI) [20, 25, 27, 28, 29, 36, 37, 45, 44] is a Search Based Software Engineering (SBSE) technique [21] which treats program code as if it were genetic material that can then be evolved to produce optimised solutions. GI has previously been found effective at optimis-

GECCO '15, July 11 - 16, 2015, Madrid, Spain

DOI: http://dx.doi.org/10.1145/2739480.2754752

- Program and its Genotype Representation:
  - Source code conversion
  - Select lines that might be changed
  - A gene represent sequences of selected lines
  - Introduce source code change operations (for mutation)
    - DELETE (delete a line of code)
    - REPLACE (replace one line with another)
    - COPY (copy a line to another location)
    - Assumption: All required information is already in the available source code! Consider similar statements only.
    - **Example**: a while loop (e.g., `x>5`) can be replaced with the condition of another while loop (e.g., `y==2`)

Genetic improvement (cont.)

- Fitness Function: Energy consumption

The fitness of a candidate solution is determined by measuring the total energy consumed (see Section 2.4) across all tests selected from the training set (see Section 2.3) when using the original unmodified software divided by the energy consumed by the phenotype across the selected tests. Thus, a fitness greater than 1 indicates a solution that consumes less energy, while a fitness less than 1 indicates a solution that consumes more energy.

- Selection:

Each selected test case can either be passed or failed. A test is deemed to have passed when the modified version categorizes a test as satisfiable or unsatisfiable, with that categorization equal to the categorization produced by the original MiniSAT. We use the original code as an oracle to guide the GI to functionally correct solutions. When a test is found to have failed, the energy consumption or that test case is not included in the fitness evaluation, and instead, an appropriate penalty is applied. To be selected for the next generation, a solution must have a fitness of above 0.95, have passed an appropriate number of the selected test cases, and be in the top 50% of the population.

Genetic improvement (cont.)

- Crossover:

Crossover is carried out by selecting one parent based on fitness and another chosen randomly from the selected individuals. Due to the simplicity of the genotype representation, crossover consists of appending one genotype to another, producing a new individual. Crossover is carried out until the population size, after selection, has doubled.

- Mutation:

After crossover, mutations are applied to the selected genotypes. Prior investigations have shown GI frameworks such as this can lead to bloat, resulting in effective solutions being encumbered with ineffective mutations. For this reason, elitism has been implemented so that the top 5 solutions in each generation move forward to the next without mutation. The remaining selected individuals have a 50% chance of having a mutation applied. Mutations consist of adding a random DELETE, REPLACE, or COPY modification to the genotype. If the population after crossover has not met the preset population size, then single, random mutations are added as entirely new genotypes until the population size is met.

Genetic improvement (cont.)

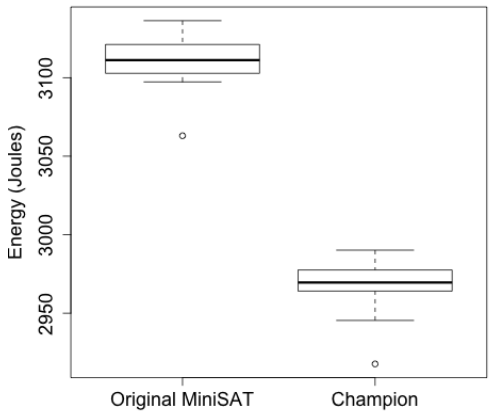
- Carrying out the experiments
  - Program to be optimized: MiniSAT (MiniSAT2-070721)
  - MiniSAT applications: CIT, Ensemble, AProVE
  - Test cases during the search are selected depending on the application

- Estimation of the energy consumption:
  - Intel Power Gadget API for Mac OS X2, which estimates the energy consumption of 2nd Generation and higher Intel Core processors
  - Intel Power Gadget uses drivers and libraries to read the processor's special energy model-specific registers (MSRs) over a specified time period. These register readings are then used to calculate the total energy consumed.

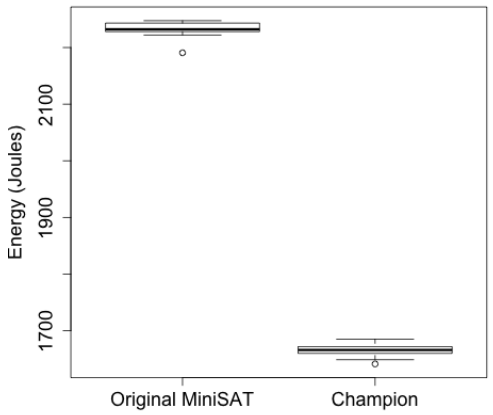
Experimental results

- Energy consumption reduction using GI

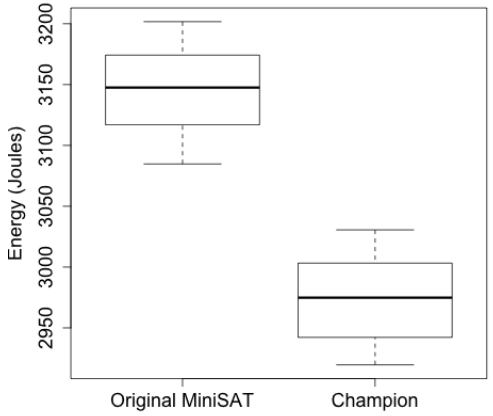
Application	Original(J)	Champ(J)	Reduction(%)
CIT	3111	2969	4.58
Ensemble	2232	1665	25.39
AProVE	3145	2973	5.44



(a) CIT



(b) Ensemble



(c) AProVE

Experimental results (cont.)

- Can we optimize for one SAT application and use it in another?

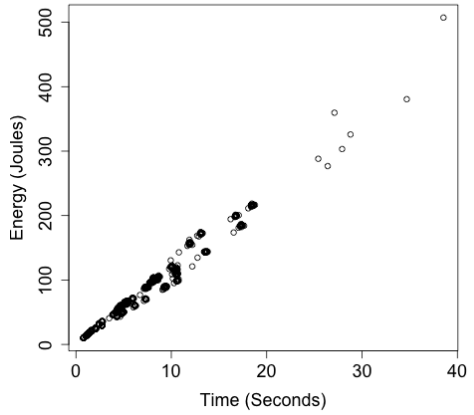
-	On CIT	On Ensemble	On AProVE
CIT	-	X	X
Ensemble	X	-	X
AProVE	3.56%	3.86%	-

An X indicates a time out

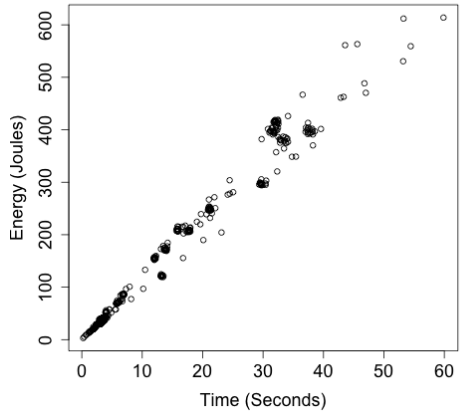
- Most likely no (or at least not always)

Energy consumption vs. runtime

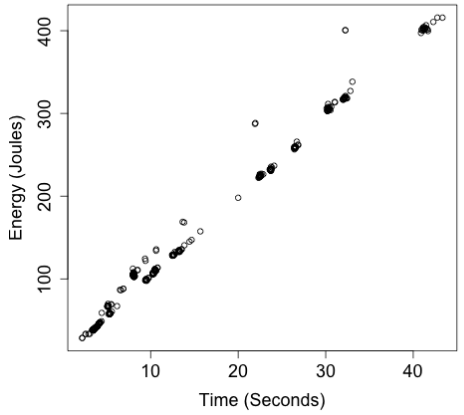
Application	Unmodified(s)	Champion(s)	Reduction
CIT	268	261	2.58%
Ensemble	219	162	25.89%
AProVE	280	261	6.69%



(a) CIT



(b) AProVE



(c) Ensemble

These findings show that for CPU-bound processes, such as MiniSAT, optimizing execution time exclusively produces more energy-efficient solutions (and vice-versa).

Summary & Conclusions

Summary

- Discussed and defined the term “Sustainable Software Engineering” / ”Green Software”
- Discussed influencing factors of sustainability of Green in Software
- Introduced a method for converting programs into ones that require less energy resources (Genetic Improvement / Genetic algorithms, Search-based Software Engineering)

Other conclusions

- The term “sustainable software engineering” is not commonly agreed on.
- Finding better (= less resource requiring) programs is valuable, leading to green software
  - Less runtime seems to lead to less energy consumption
- AI approaches like GA can be used to adapt programs for resource efficiency
  - Other ideas include using Expert Systems / Decision Support Systems to come up with programs requiring fewer resources…
  - We may use LLMs or other deep-learning models for this purpose
  - …

For further information on current research in this area …

- International Workshop on Green and Sustainable Software (GREENS) as part of ICSE
  - 2012-2018, 2023-2025
  - 1-day workshop
  - See https://greensworkshop.github.io
  - Objectives:

Engineering green software-intensive systems is critical in our drive towards a sustainable, smarter planet. ***The goal of green software engineering is to apply green principles to the design and operation of software-intensive systems****.* ***Green and self-greening software systems have tremendous potential to decrease energy consumption****. Moreover, software can and should be rethought to address sustainability issues, for instance, innovative business models, new processes, and incentives. Monitoring and measuring the greenness of software is critical to the notion of sustainable software.*