Software (engineering) and energy efficiency (and other effects) – the big picture

Presentation for LVM Working group on climate and environment effects of ICT - applications

Prof. Jari Porras (LUT University) and

KARLSKRONA MANIFESTO
FOR SUSTAINABILITY DESIGN

Aim of this study

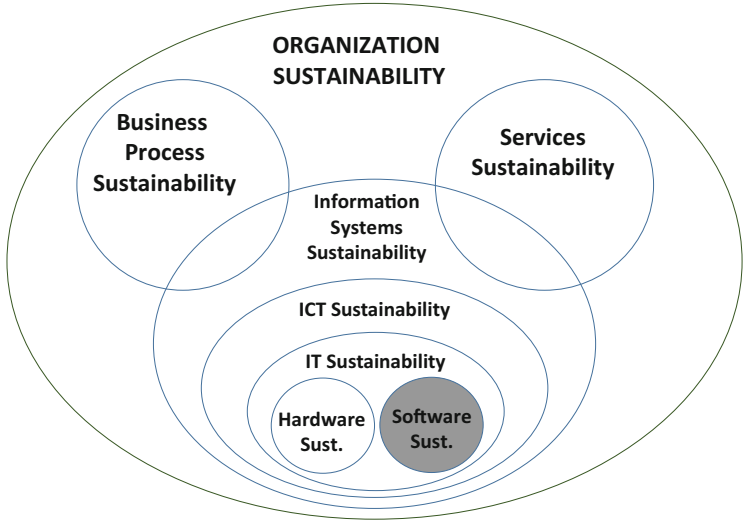
- This study is performed by people working on software (engineering) and sustainability related issues within Karlskrona Sustainability Design Alliance
- This study is prepared for the Finnish Ministry of Transportation and Communication working group on Climate and Environment effects of ICT (#ICTClimate)
- The aim of this study is to collect information from literature concerning the ways software and systems, as well as processes of software engineering, can have an effect on environment and thus climate.

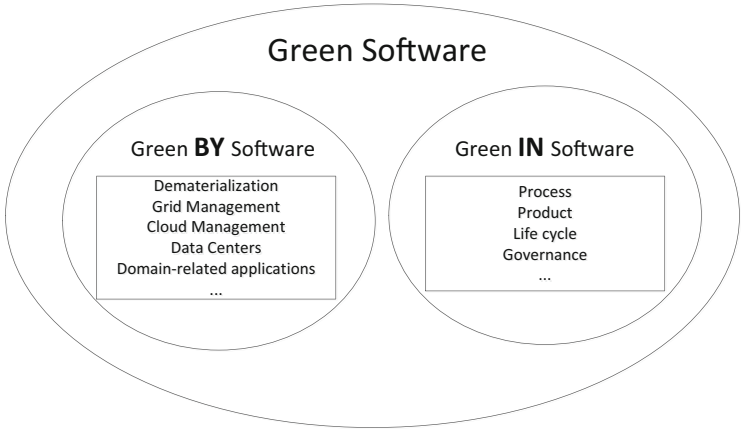
Categories to be considered

- Footprint of software and systems
  - Energy efficiency issues as a way to decrease the footprint
- Sustainability within software engineering process
  - Sustainability as a quality attribute
- Handprint of software and systems
- Increasing the awareness of the effects of software and systems

Calero & Piattini: Introduction to Green in Software Engineering, Green in Software Engineering, Springer, 2015

- Software sustainability
  - the way to achieve sustainable software is principally by improving power consumption but rarely if ever the objective
- Software engineering sustainability
  - Sustainability should generally be taken into account from the very first stages of software development
- Green software
  - Produces as little waste as possible during its development and operation
- Green in Software Engineering
  - include green practices as part of the software development process





Footprint of software and systems (energy efficiency)

Taina: How green is your software?, ICSB, 2010

- Since a software has a life cycle, it creates direct and indirect carbon emissions: it has a carbon footprint
- Software has indirect resource requirements
- 1) Development: The software is implemented,2)Beta testing: Potential customers test the software, 3) Delivery and re-delivery: The software is delivered to customers, 4) Usage: The software is used and 5) Maintenance: The software is updated.

$$cf_s = (d_s + b_s + dr_s + m_s)/n_s + u_s$$

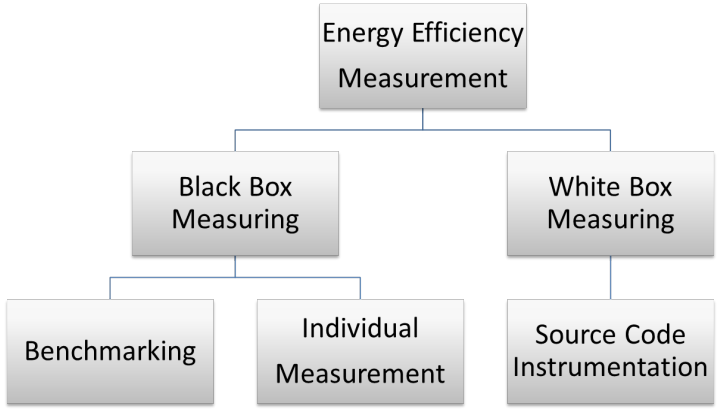
$$u_s = (\sum_j t_j) * p * F$$

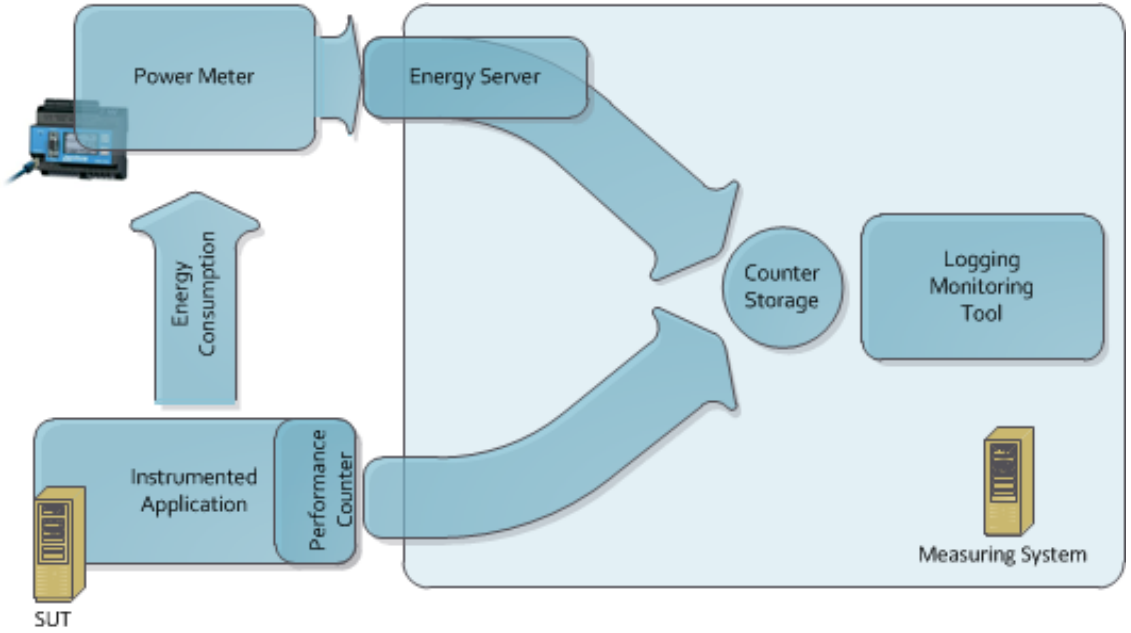
$$d_s = dg_s + dd_s + diu_s + di_s + dv_s$$

Johann et al.: How to Measure Energy-Efficiency of Software: Metrics and Measurement Results, GREENS, 2012

- Generic metric to measure software and a method to apply it in a software engineering process

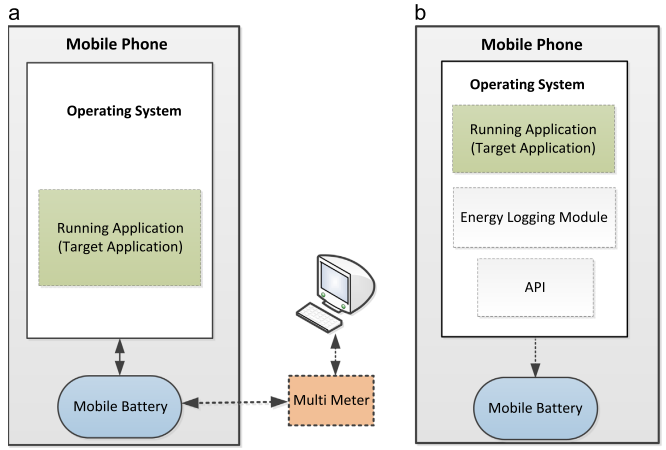
$$EnergyEfficiency = \frac{UsefulWorkDone}{UsedEnergy}.$$

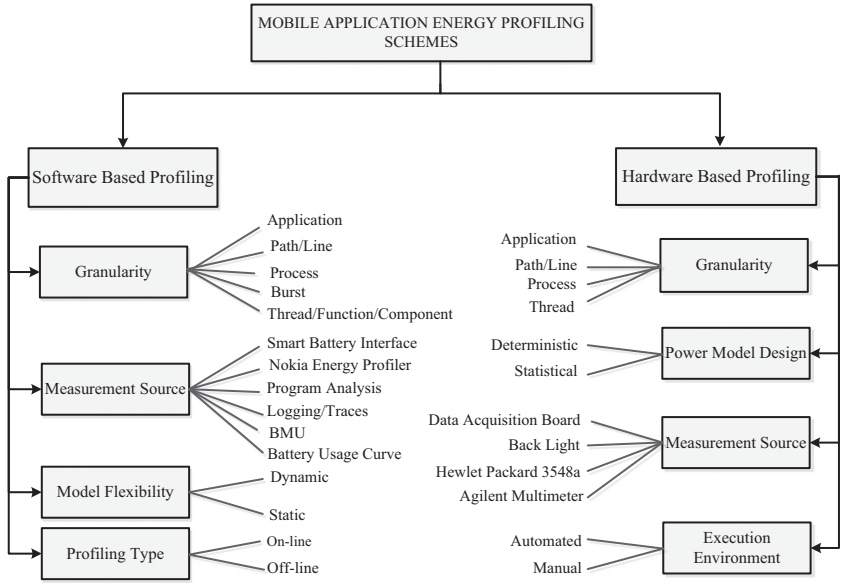




Ahmad et al.: A Review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues, JNCA, 2015

- Hardware-based energy pro- filing schemes are expensive, labor-intensive, and non-scalable compared with software-based solutions
- Hardware-based energy profiling schemes are highly accurate for the specific mobile device for which they are developed, but worst accurate for other device models
- Software-based energy profiling estimates battery consumption at diverse granularities, such as process, thread, function, line, or path, by maximizing numerous power tracking resources
- The correctness of software-based energy profiling designs is affected by the accuracy level offered by the voltage and current sensors of the smart battery





Energy efficiency of various software elements ... examples

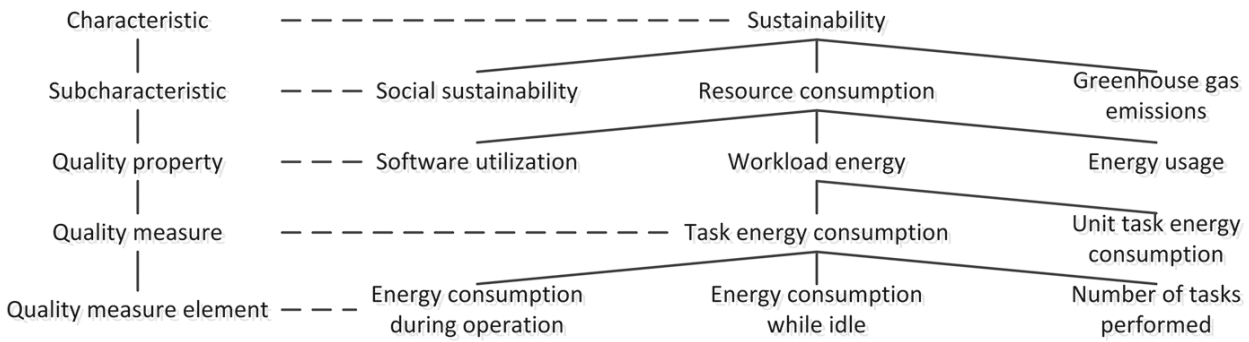
- Vlad Christea: Energy Consumption of Applications on Mobile Phones, M.Sc. Thesis, 2017 – energy usage of screen elements
- Dagnachew Temesgene: Cyber foraging for green computing, improving performance and prolonging battery life of mobile devices, M.Sc. Thesis, 2016 – energy usage of different functions of mobile phones
- Mustaqim Rahman: Analysing API Calls to Reduce Energy Consumption of Apps in Idle States, M.Sc. Thesis, 2017 – benchmarking API performance

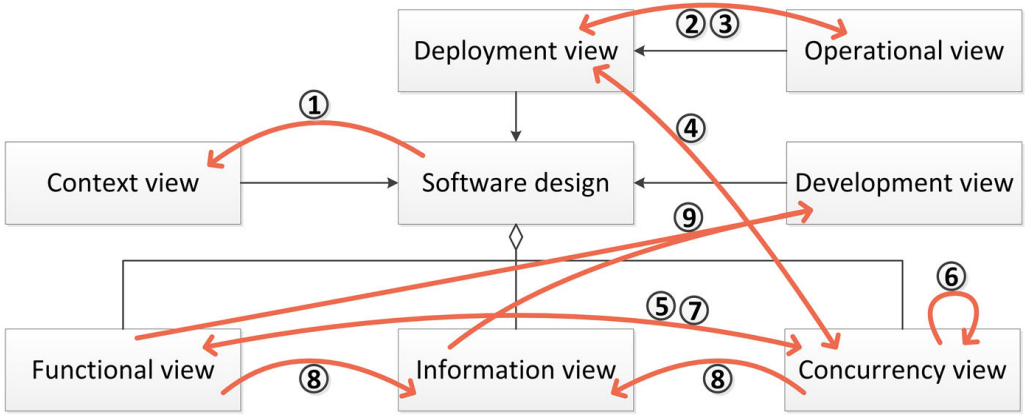
Interim conclusion

- Energy efficiency of software in different environments can be measured BUT
  - Results are highly contextual (environment)
  - Results are hard to compare
  - Energy efficiency is only one part of the picture
    - Software is always tied to hardware
    - Optimising energy efficiency now may have an effect on other sustainability effects

Jagroep et al.: Extending software architecture views with an energy consumption perspective, Computing, 2016

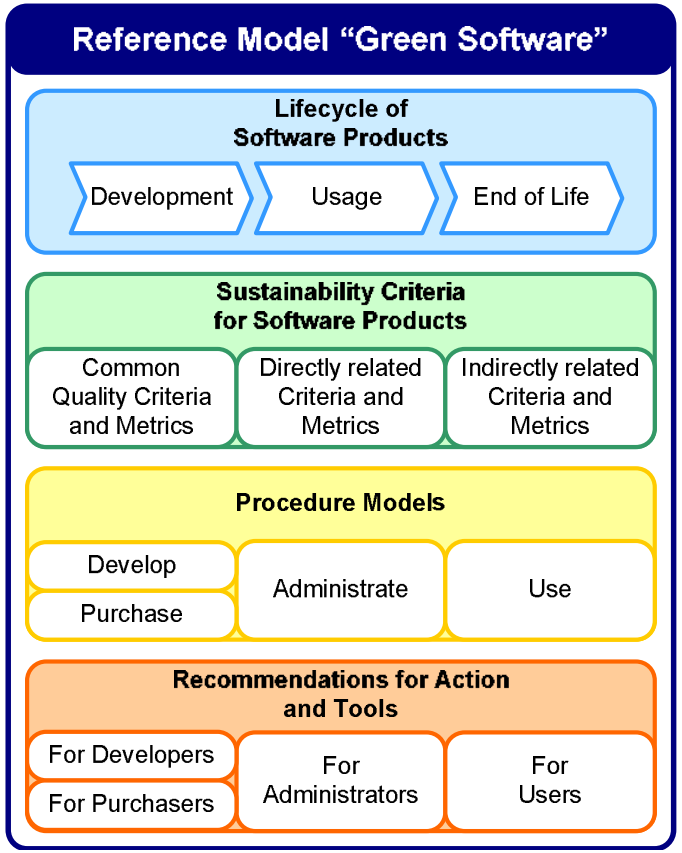
- Software vendors are not able to address energy consumption on software level
  - Energy consumption of software is measured by relating the **energy consumption of hardware** to **computational resource usage** on behalf of the software and, consequently, energy efficiency refers to the efficient use of computational resources
  - Use energy profilers; software tools including a power model with the ability to estimate the software EC on different levels of granularity, not always efficient
- Architecture Description (AD) of product software complemented with Energy Consumption measurements, could help to direct green computing efforts (i.e. Energy Efficient (EE) algorithms) and determine appropriate adjustments on the right locations
  - creation of Energy Efficient software starts with the design of the software, i.e. with its architecture





Kern et al.: Green software and green software engineering–definitions, measurements, and quality aspects, 2013

- Measurement of software
  - Causes of energy consumption, complexities, dependencies
- Energy efficient software
  - **Methodologies**, designs, tools for improving energy efficiency
- Green software and its engineering
  - Software bloat vs. **efficient code**
- Green software reference model
  - **Metrics**, Measurement challenges



Procaccianti et al.: The Green Lab:Experimentation in Software Energy Efficiency, 2015

- Software Energy Efficiency is a research area that lacks well-defined, validated methods
  - *Chaotic behavior*
  - *Complexity of measurement*
  - *Anecdotal and contradictory evidence*
  - *Lack of a unified approach*
- How to combine the traditional hypothesis-driven (top-down) approach with a bottom-up discovery approach
  - Energy hotspots
  - Energy bugs
  - Energy smells

Capra et al.: Is software “Green”?, Information and software technology, 2012

- A higher use of application development environments has a detrimental effect on software energy efficiency
  - small to average size applications the use of application development environments is associated with greater software energy efficiency, but that for larger applications the opposite is true
- The detrimental effect of using application development environments on energy efficiency is more pronounced for larger than for smaller applications
  - large application typically oc- curs by embedding initial modules inside other larger modules. This may unnecessarily increase the number of layers that must be crossed to execute a single operation

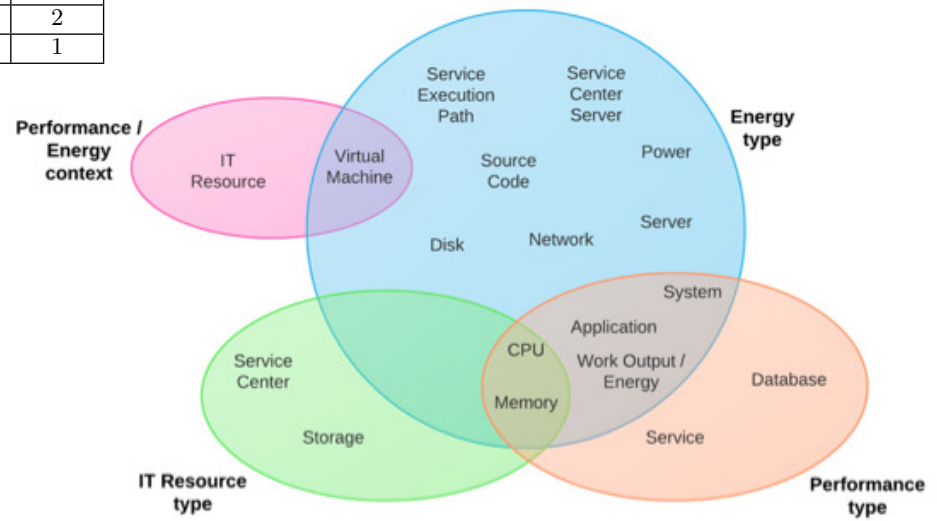
Bozzelli et al.: A SLR on green software metrics, 2014

- What green metrics have been proposed in the Software Engineering literature?
- How green metrics can be classified?

Measured resource	Total
Architecture Elements	17
Application	13
Service	11
Financial Impact	9
Memory	9
CPU	7
Storage	5
Virtual Machines	5
Source Code	3
Data Center	2
Power	2
Process	2
Service Center	2
Service Execution Path	2
DBMS	1
IT Resource	1
JVM	1
Network	1
Pollution	1
Server	1
System	1

Metrics Type	Total	Measurement Unit(s)
Energy	48	Joule (J), Index, Watt (W), Ampere (A) Kilowatt-hour (kWh), Number, byte/kWh
Performance	19	GFLOPS/kWh, Computing Unit/kWh, Percentage (%), Seconds (s), Index, Number
Utilization	17	Percentage (%), Megabyte (MB), Megahertz (MHz), GB/s
Economic	9	Dollars ($)
Performance / Energy	2	GFLOPS/Watt, Index
Pollution	1	CO_2 units

	Years							
Metrics Type	2001	2002	2008	2009	2010	2011	2012	Total
Energy	2	3	21	4	8	5	5	48
Performance	0	0	0	0	2	10	7	19
Utilization	0	0	0	0	3	10	4	17
Economic	0	0	0	0	0	4	5	9
Performance / Energy	0	0	0	0	0	1	1	2
Pollution	0	0	0	0	0	1	0	1



Ardito & Morisio: Available data and guidelines for reducing energy consumption in IT systems, Sustainable computing: Informatics and Systems, 2014

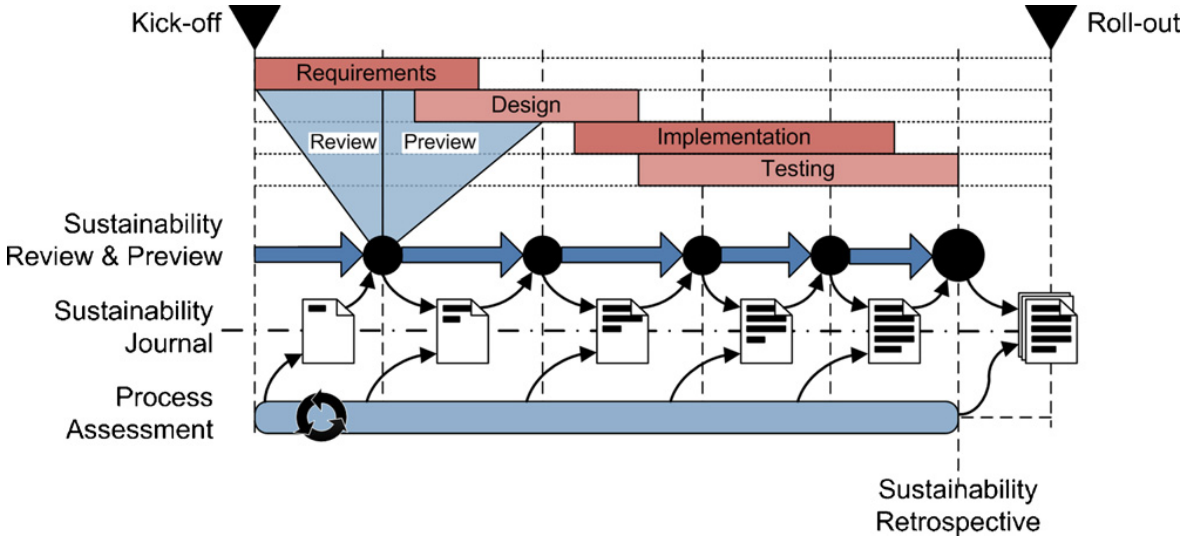
- Energy efficiency guidelines (complex)
  - Infrastructure …
  - Application
    - Design efficient UI - simplified interaction
    - Event based programming – sleeping
    - Low level programming – use of system optimization
    - Batch II/O – sleeping
    - Code migration – optimized execution
    - Reduce data redundancy – storage and transfer optimization
    - QoS scaling – behavior change
    - Profiling tools – optimization
  - Operating systems
    - Power management
    - Optimal use of peripherals
    - Compiler optimization
    - Background process optimization
  - Hardware …

Sustainability within software engineering process

Naumann et al.: The GREENSOFT Model, Sustainable computing: Informatics and Systems, 2011

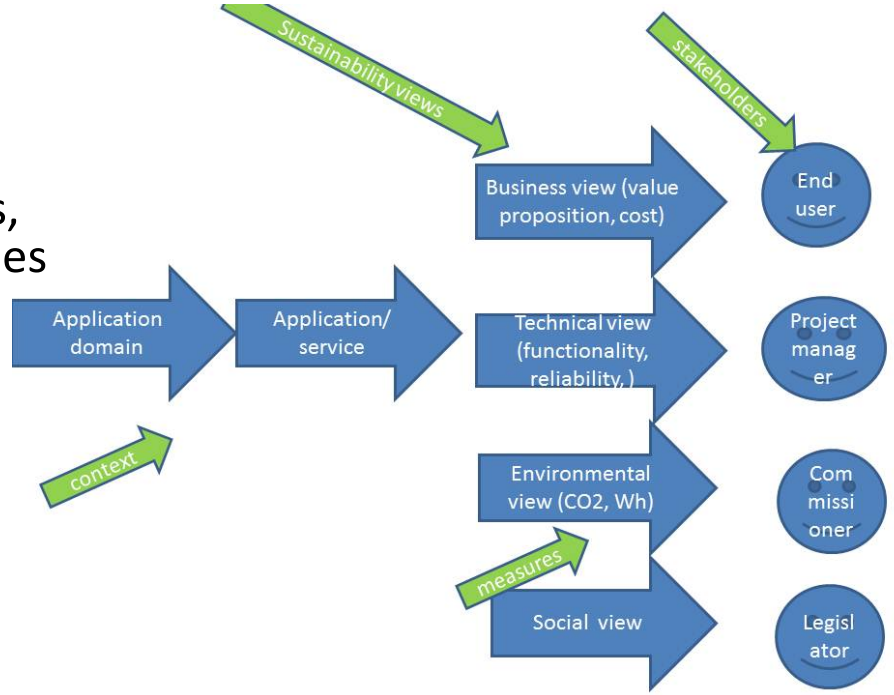
- “Green and Sustainable Software” and “Green and Sustainable Software Engineering”

	Development	Usage	End of Life
Third-order Effects	- ... - Changes in software development methods - Changes in corporate organizations - Changes in life style	- ... - Rebound effects - Changes of business processes	- ... - Demand for new software products
Second-order Effects	- ... - Globally distributed development - Telework - Higher motivation of team members	- ... - Smart grids - Smart metering - Smart buildings - Smart logistics - Dematerialization	- ... - Media disruptions
First-order Effects	- ... - Daily way to work - Working conditions - Business trips - Energy for ICT - Office HVAC - Office lighting - … - Manuals - Transportation - Packaging - Data medium - Download size	- … - Accessibility - Hardware requirements - Software induced resource consumption - Software induced energy consumption	- … - Backup size - Long term storage of data (due to legal issues) - Data conversion (for future use) - ... - Manuals - Data medium - Packaging
	Development, Distribution	Usage	Deactivation, Disposal



Lago et al.: Leveraging “Energy Efficiency to Software Users”, GREENS/ICSE, 2013

- Social, environmental, economic, technical sustainability dimensions
- A framework framing green software quality
  - green concerns - potentially impact all other system qualities, demanding for an additional dimension that frames both types of qualities for trade-offs analysis and decision making
- The role of stakeholders – what stakeholders matter in software sustainability
- Green trade-off analysis - spans all four sustainability dimensions
- Sustainability goals and design concerns
  - all sustainability dimensions should be made explicit in the same way as they are for (technical) design and architectural concerns
- Environmental sustainability needs context



Lago et al.: Framing sustainability as a property of software quality, CACM, 2015

- Framework by extending an existing model, the *Third Working Draft of ISO/IEC 42030 Architecture Evaluation*
- Traditional **software decision making** considers trade-offs either between different technical sustainability criteria (such as performance versus availability) or between technical sustainability criteria and economic sustainability criteria (such as performance versus costs)
- Sustainability-related software decision making involves trade-offs between environmental sustainability criteria (such as energy efficiency) and social, economic, and technical sustainability criteria

Figure 1. Framework for sustainability software-quality requirements.

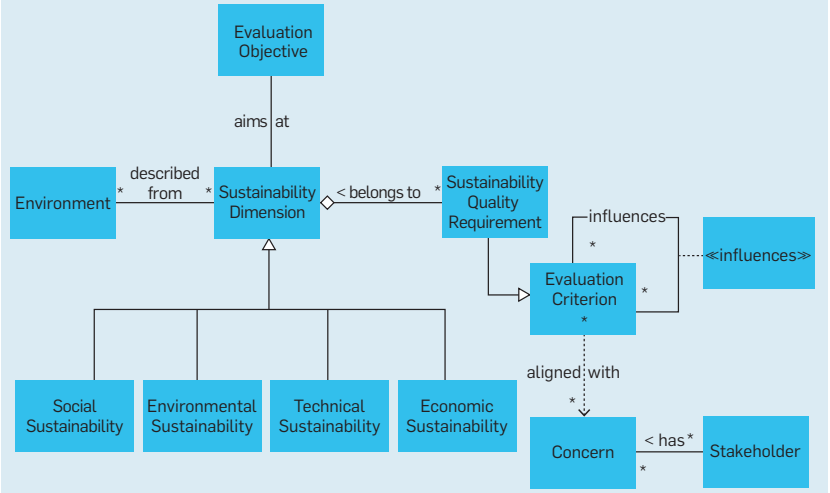
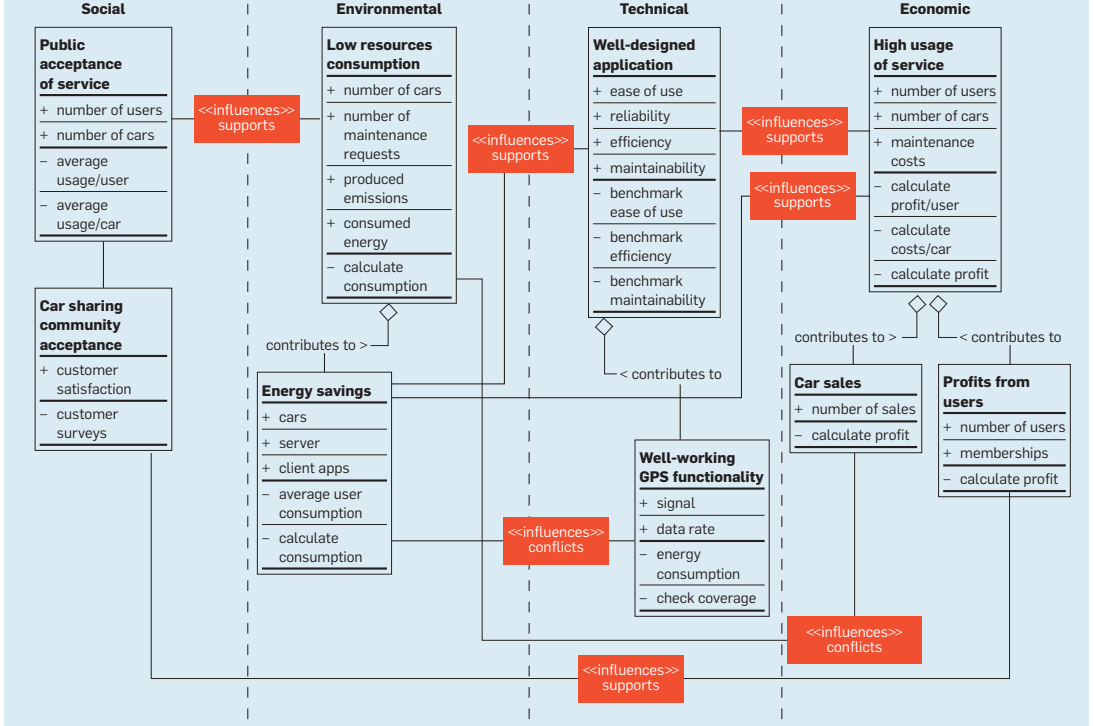
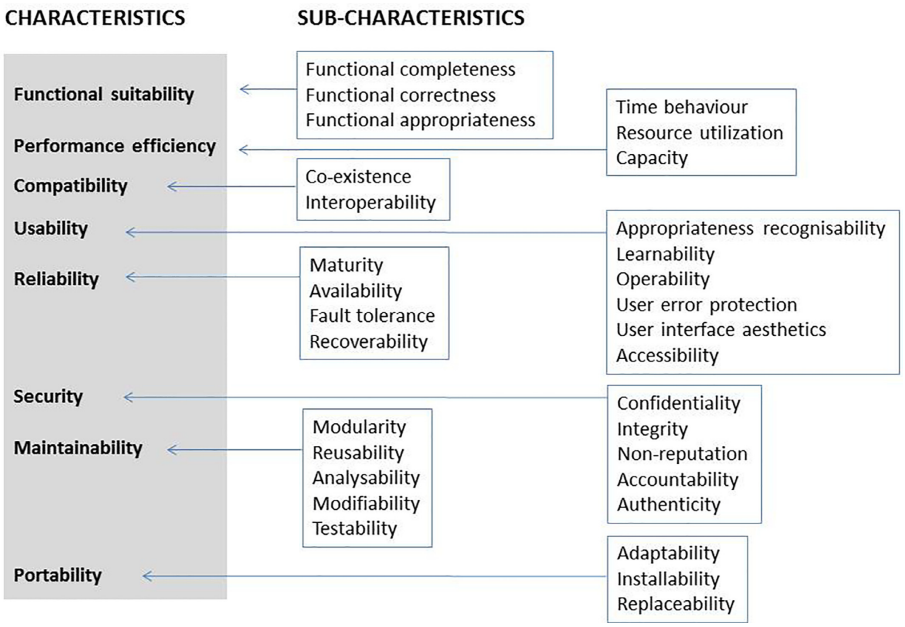


Figure 3. Sustainability quality requirements: car-sharing platform.



Condori-Fernandez & Lago: Characterizing the contribution of quality requirements to software sustainability, JSS, 2018

- Key challenge for software sustainability is its characterization as a software quality requirement
- Software quality characteristics defined by the ISO/IEC 25010 standard quality models
- Software architects, Project managers, Sustainable ICT experts, Requirement engineers



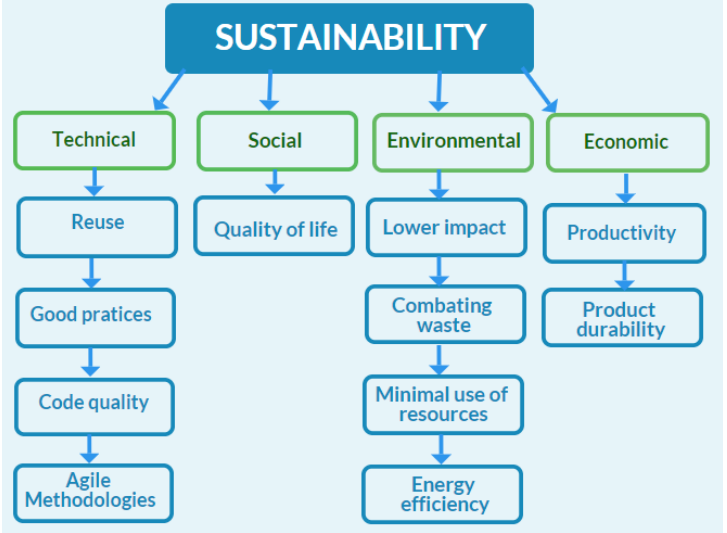
	ISO/IEC 25010:2011 Quality model Qualities	Technical	Economic	Social	Environmental
Product quality model	**Compatibility**				
	Co-existence	x	x	x	x
	Interoperability	x	x	x	x
	Functional suitability				
	Functional appropriateness	x	x		x
	Functional corrected	x	x		x
	Functional completeness	x	x		x
	Maintainability				
	Analysability	x	x		
	Modifiability	x	x		x
	Modularity	x	x		
	Reusability	x	x		x
	Testability	x	x		
	Performance efficiency				
	Capacity	x	x		x
	Resource utilization	x	x		x
	Time behaviour	x	x		x
	Portability				
	Adaptability	x	x		
	Installability	x	x		
	Replaceability	x	x		
	Reliability				
	Availability	x	x		x
	Fault tolerance	x	x		x
	Maturity	x	x		x
	Recoverability	x	x		x
	Security				
	Accountability			x	
	Authenticity			x	
	Confidentiality			x	
	Integrity			x	
	Non-repudiation			x	
	Usability				
	Accessibility		x	x	
	Appropriateness recognizability		x	x	
	Learnability		x		
	Operability		x		
	User error protection		x		
	User interface aesthetics		x		
	Context coverage				
	Context completeness	x	x		x
	Flexibility	x	x		x
Quality in use model	**Effectiveness**	x	x	x	x
	Efficiency	x	x	x	x
	Freedom from risk				
	Economic risk mitigation		x		
	Environmental risk mitigation			x	x
	Health and safety risk mitigation			x	
	Satisfaction				
	Comfort	x	x	x	
	Pleasure	x	x	x	
	Trust	x	x	x	
	Usefulness	x	x	x	

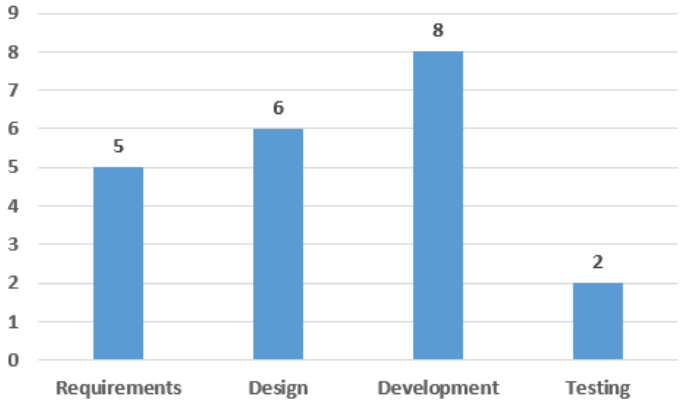
- Qualities identified as good contributors to technical sustainability (Functional correctness, functional appropriateness, availability, modifiability, interoperability and recoverability) favor positively the endurability of software systems
- Some of maintainability requirements (in terms of reusability and modifiability) as relevant for addressing environmental sustainability

	Characteristics	Quality attributes	High	Medium	Sum
A38	Maintainability	Reusability	11/16	2/16	13/16
A39	Performance efficiency	Resource utilization	11/16	4/16	15/16
A31	Efficiency	Efficiency	9/16	6/16	15/16
A22	Maintainability	Modifiability	8/16	4/16	12/16
A16	Compatibility	Co-existence	8/16	7/16	15/16
A19	Reliability	Availability	7/16	4/16	11/16
A11	Freedom from risk	Environmental risk mitigation	6/16	6/16	12/16
A26	Performance efficiency	Time behaviour	6/16	6/16	12/16

Karita et al.: Software industry awareness on green and sustainable software engineering: a state-of-the-practice survey, SBES, 2019

- RQ3: What phases of the software development life cycle (SDLC) do sustainable practices apply?
- RQ4: What dimensions of sustainability have been explored in practice (technical, environmental, social and eco- nomic) of software development?
- RQ5: What models for sustainable software development have been adopted by the software industry?
- RQ6: What tools have been used to support sustainability in the software development process?





Dimensions	Sustainability concern	Irrelevant (1)	Less important (2)	Neutral (3)	Important (4)	Very important (5)
Technical	Longevity		1		**12**	**12**
Technical	Resilience to uncertainty				11	**14**
Technical	Performance			1	9	**15**
Technical	Software Evolution		1	1	**13**	10
Technical	Reusability				7	**18**
Technical	System Quality				5	**20**
Social	Product Roadmap		1	3	**13**	8
Social	Awareness		1	1	**13**	10
Social	Ethics			3	**11**	**11**
Environmental	Energy consumption	1	1	5	**10**	8
Environmental	Environmental concern		1	6	**13**	5
Economic	Time to Market	1		3	**11**	10
Economic	Development effort			3	**12**	10

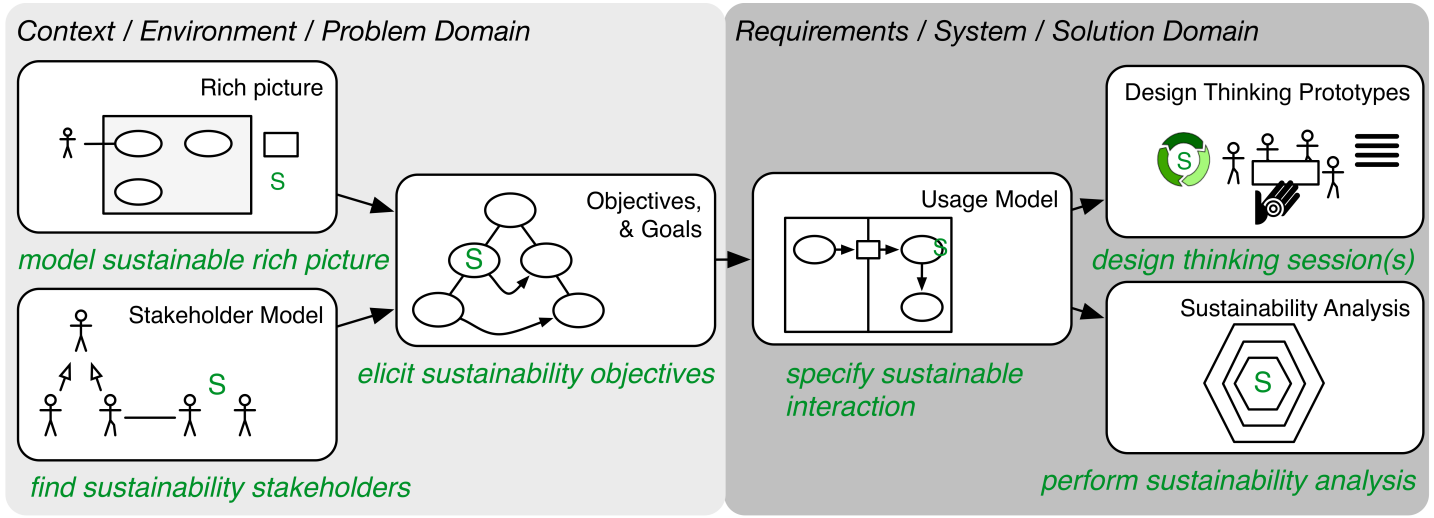
Becker at al.: Requirements: The key to sustainability, IEEE Software, 2016

- Sustainability has often been equated with environmental issues, but it requires simultaneous consideration of environmental resources, societal and individual well-being, economic prosperity, and the long-term viability of technical infrastructure
  - trade-offs occur across other dimensions
- A software system's impact on its environment is often determined by how the software engineers understand its requirements
  - *sustainability debt*: decisions made for the present situation have invisible effects that accumulate over time in each of the five dimensions
- A series of decision points occurs during system design. Many of them are requirements-engineering activities that occur repeatedly in all iterations throughout the projects

Task	Standard current practice	Focus of future practice
Mind-setting	The world is a puzzle, and we should solve the problem.	The world is complex, and we should first understand the dilemmas.
Determination of the project objective and the system purpose, boundary, and scope	Focus on the immediate business need and key system features. Don't question the project's or system's purpose.	Emphasize how the project can affect sustainability in all dimensions. Strive to advance sustainability in multiple dimensions simultaneously. Experiment with different system boundaries to understand the alternative impacts.
External constraint identification	See constraints as imposed by the direct environment of the system and its technical interfaces. Minimize the constraints considered, but include legal, safety, security, technical, and business resources.	See constraints in each dimension as opportunities. Look for constraints from additional sources, starting with company corporate-social-responsibility policies, legislation, and sustainability standards.
Stakeholder identification	Minimize the number of stakeholders involved, and focus on those who have influence. Focus on internal stakeholders, and exclude unreachable stakeholders.	Maximize stakeholder involvement in an inclusive perspective integrating external stakeholders, and involve those who are affected. Assign a dedicated role to be responsible for sustainability, and introduce surrogate stakeholders to represent outside interests.
Success criteria definition	Focus on the financial bottom line at project completion. Measure the business outcome and financial return on investment.	Focus on advancing multiple dimensions simultaneously, including financial aspects, and take into account that most effects occur after project completion.
Requirements elicitation	Focus on the features and immediate effects the stakeholders want.	Help the stakeholders understand the system's enabling effects. Use creativity techniques and long-term scenarios to forecast the potential structural impact.
Risk identification	Identify risks that threaten timely project completion within the budget.	Include the effects on the system's wider environment. Include enabling and structural effects and risks that can develop over time.
Tradeoff analysis	View tradeoff analysis as a prioritization and selection problem, and let the key stakeholders decide.	Strive to transform sustainability tradeoffs into mutually beneficial situations. Ensure that a wider range of stakeholders (or their surrogates) discuss sustainability tradeoffs.
Go/no-go decision	Base the decision on feasibility, financial costs and benefits, and risk exposure to project participants—that is, internal stakeholders.	This continues to be an internal business decision but is documented to show to external audiences that it took into account sustainability indicators and enabling effects. The decision is based on a consideration of positive and negative effects in all five dimensions.
Requirements validation	Let key stakeholders verify that their interests are captured.	Ensure broad community involvement focused on understanding effects.
Project completion	Verify whether success criteria are met on the completion date. After that, focus on maintenance and evolution.	Evaluate the effects in all five dimensions over a certain time frame after completion, aligned with the expected timescale of effects.
Requirements documentation	Current templates ignore long-term effects and sustainability considerations.	Templates require information about sustainability as a design concern and support analysts with checklists.

Penzenstadler et al.: Everything is INTERRELATED:Teaching Software Engineering for Sustainability, SEET/ICSE, 2018

- Keeping the sustainability in the process from the beginning
  - From problem domain to solution domain
- Getting all relevant stakeholders involved
- Having clear sustainability objectives



Venters at al.: Characterising Sustainability Requirements, SEIS/ICSE, 2017

- Term ‘sustainability requirement’ in software and requirements engineering
  - constructed in a way that suggests it is different in the way from how we understand requirements in general

Area	Key concepts	Motivation	Main actors	Sustainability requirement context
IS	Cost effectiveness Process improvement Process structuring	Improve cost effectiveness of process, aiming for cost reduction.	Business, Regulators, Customers	Metrics and controls context, “such as operating and capital cost, safety, energy cons., waste gen., efficiency”
ICT	Optimisation of IT infrastructure, Green computing, Environmental sustainability, Sustainability of IT services, Longevity of energy systems	Improved resource and energy efficiency of ICT	Customers, employees, business partners, NGOs	Environmental sustainability related to energy consumption and performance
SW Eng	Software development process models	Environmental impacts of ICT	Software developers, administrators, users	Implicit non-functional qualities
Sys Eng	Optimize systems considering sustainability issues	Economic expectations and environmental consciousness	All stakeholders in context, noting they have varying background	Sustainability requirements have to be communicated
Ergonomics	Multi-dimensional understanding with economic, social, and environmental	Economic and business-strategic aspects, human factors	Wide range of stakeholders, including all designers	Environmental context and long life cycles
RE	Multi-dimensionality of sustainability, Interdependence of dimensions, Trade-offs, General models of sustainability	Make sustainability more tangible, Make related goals explicit, Assess sustainability	Decision making households and/or software professionals, regulators	Multiple dimensions and trade-offs: ‘Achieve acceptable level of service (...), have min. impact on natural env., be socially and economically acceptable’

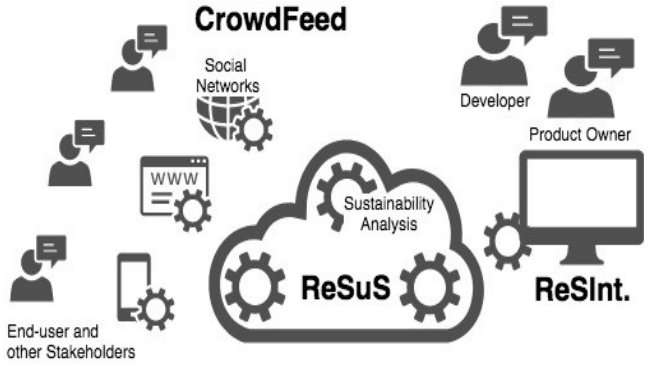
Seyff et al.: Tailoring Requirements Negotiation to Sustainability, RE, 2018

- Existing RE methods and tools do not explicitly facilitate the discussion and negotiation of sustainability-related concerns
  - leads to insufficient or one-dimensional perceptions of sustainability
- Adapted EasyWinWin approach

Idenfiy Issues, Options and Agreements		Sustainabilty Dimensions					Orders of Effects		
Identifer	**Description**	**Individ.**	**Social**	**Economic**	**Environ.**	**Technical**	**Immediate**	**Enabling**	**Structural**
Win Condition 1	The webshop shall notfify user of all new products	neg		x	x	x	x	x	
Issue 1	If too many notifications are sent this could lead to spam	neg		neg	neg	neg	Environ.	Individ. Economic Technical	
Issue 2	What if the users do not have an interest in these products?	neg		neg				Individual Economic	
Option 1 for Issue 1	There is a maximum number of messages sent to a customer per a predefined period of time	1		1	-1	-1	Environ.	Individ. Economic Technical	
Option 2 for Issue 1	Messages are only sent for highly attractive products	2		2	-1	-1	Environ.	Individ. Economic Technical	
Option 1 for Issue 2	We just send info for products the customer has declared interest	3		2	-1	-1		Individ. Economic Technical	
Agreement for Win Condition 1	The webshop shall notfify user of new key products which they have declared an interest in.								

Seyff et al.: Crowd-Focused Semi-Automated Requirements Engineering for Evolution Towards Sustainability, RE, 2018

- No established means to analyse the impact of a given requirement on sustainability
- Concept of *sustainability requirements*
  - non-functional requirement or software quality aligned with one or more of the sustainability dimensions
- Expect that requirements positively affecting sustainability ideally have an overall long- term positive effect on one or more sustainability dimensions



- Can also be used to support the elicitation, analysis, and negotiation of user concerns regarding other issues (e.g., usability, accessibility, or performance)

Venters et al.: Software sustainability: Research and practice from a software architecture viewpoint, JSS, 2018

- Software systems are sustainable if they can be cost-efficiently maintained and evolved over their entire life-cycle, which is arguably determined by the software architecture
  - mechanism for reasoning about key software qualities (e.g. **maintainability**, **extendability**, scalability, security, performance, reliability, portability etc.)

- Software sustainability, Software architecture sustainability, Sustainable software architecture decisions
  - Architecture drift, erosion, Sustainability debt
- loss of quality of a system must be estimated using appropriate indicators and metrics that can smell that the quality is decreasing during evolution cycles
- key issue in assessing the value of software metrics is whether they support decision- making.

Overview of software metrics that can be used to estimate architecture sustainability.

Architecture level metrics			
	Smells	Metrics	Quality attributes
Maintenance	Smells about ambiguous and unused interfaces, when functionality of modules are rather small or big and those smells concerning delegation of functionality	Module interaction index, Attribute hiding factor, API function usage index, Module Size Uniformity Index, Module Size Boundedness Index	Complexity, Modularity (Mitchell, 2006), Analyzability, Effectiveness, Understandability
	Smells that effect to duplicate functionality and coupling between components	Clone detection, Coupling between object, Ratio of cohesive interaction, Modularization Quality	Reusability, Complexity, Modifiability, Modularity
	Smells where multiple components realise the same concern or a component implements an excessive number of concerns. Therefore, we can identify components with a suitable percentage of methods	Concern diffusion over architectural components, Component-level interlacing between concerns, Number of concerns per component, Well-sized Methods Index	Reusability, Modifiability, Understandability, Modularity
	We identify components with an excessive number of dependencies, cyclic dependencies and dependencies that crosscut layers	Cyclic dependency index, API function usage index, Layer Organization Index, Cumulative component dependency, Excessive structural complexity	Modularity, Understandability, Changeability Modifiability
	Other cross-cutting smells affecting any part of the architecture	Architectural smell coverage Architectural smell density	Cost
Evolution	Elements that change too often, Number of elements impacted by a change	Instability, Ripple effect, Distance from Main Sequence, Module Interaction Stability Index	Stability, Evolvability
	Likelihood of components that evolve together	Bi-directional coupling component	Complexity, Evolvability
Architecture knowledge level metrics			
Maintenance	Excessive number of decisions and trace links	NodeCount, EdgeCount	Complexity, Stability
	Too many AK ítems and decision alternatives	Cost of AK capturing effort	Cost
Evolution	A change impact on many decisions	Ripple effect, instability, change proneness	Changeability, Stability
	Obsolete decisions and frequent changes	Decision volatility	Timeliness

Blevis: Sustainable Interaction Design: Invention & Disposal, Renewal & Reuse, CHI, 2007

- Focus is primarily on environmental sustainability and the link between interactive technologies and the use of resources
- Software and hardware are intimately connected to a cycle of mutual obsolescence
- If we agree that fundamental change is needed and it might be the change that users don't want, who gets to decide what change should happen and how?
- Linking invention & disposal, promoting renewal & reuse, promoting quality & equality, de-coupling ownership & identity, using natural models & reflection

10. **active repair of misuse**—is the design specifically targeted at repairing the harmful effects of unsustainable use, substituting sustainable use in its place?

1. **disposal**—does the design cause the disposal of physical material, directly or indirectly and even if the primary material of the design is digital material?
2. **salvage**—does the design enable the recovery of previously discarded physical material, directly or indirectly and even if the primary material of the design is digital material?
3. **recycling**—does the design make use of recycled physical materials or provide for the future recycling of physical materials, directly or indirectly and even if the primary material of the design is digital material?
4. **remanufacturing for reuse**—does the design provide for the renewal of physical material for reuse or updated use, directly or indirectly and even if the primary material of the design is digital material?
5. **reuse as is**—does the design provide for transfer of ownership, directly or indirectly and even if the primary material of the design is digital material?
6. **achieving longevity of use**—does the design allow for long term use of physical materials by a single owner without transfer of ownership, directly or indirectly and even if the primary material of the design is digital material?
7. **sharing for maximal use**—does the design allow for use of physical materials by many people as a construct of dynamic ownership, directly or indirectly and even if the primary material of the design is digital material?
8. **achieving heirloom status**—does the design create artifice of long-lived appeal that motivates preservation such that transfer of ownership preserves quality of experience, directly or indirectly and even if the primary material of the design is digital material? This notion of heirloom status is similar to Nelson & Stolterman's [30] description of "ensoulment".
9. **finding wholesome alternatives to use**—does the design eliminate the need for the use of physical resources, while still preserving or even ameliorating qualities of life in a manner that is sensitive to and scaffolds human motivations and desires?

Becker at al.: Requirements: The key to sustainability, IEEE Software, 2016

KARLSKRONA MANIFESTO
FOR SUSTAINABILITY DESIGN

SUSTAINABILITY PRINCIPLES FOR SOFTWARE ENGINEERING

The following principles are based on "Sustainability Design and Software: The Karlskrona Manifesto."[1]

- Sustainability is systemic; a system can never be treated in isolation from its environment.
- Sustainability is multidimensional; the five key dimensions are economic, social, environmental, technical, and individual.
- Sustainability is interdisciplinary; sustainability design in software engineering requires an appreciation of concepts from other disciplines and must work across disciplines.
- Sustainability transcends the software's purpose; any software can impact the sustainability of its socioeconomic, sociotechnical, cultural, and natural environments.
- Sustainability is multilevel; it requires us to consider at least two spheres during system design: the system under design and its sustainability, and the wider system of which it will be part.
- Sustainability is multi-opportunity; it requires us to seek interventions that have the most leverage on a system[2] and to consider the opportunity costs.
- Sustainability involves multiple timescales; it requires long-term thinking to address the timescales on which sustainability effects occur.
- Sustainability isn't zero-sum; changing a system's design to consider the long-term effects doesn't automatically imply making sacrifices now.
- System visibility is a necessary precondition and enabler for sustainability design. This is because only a transparent status of the system and its context, made visible at different abstraction levels and perspectives, can enable system designers to make informed responsible choices.

For more on this, see www.sustainabilitydesign.org.

Handprint of software and systems

Hilty & Ruddy: Towards a Sustainable Information Society, Informatik, 2000

- *Interrelation between the emerging information society and the goal of sustainability*
- *Environmental information processing and the impacts of Information Society Technologies*
- *Rebound effects*
- *Eco-efficiency to New lifestyles*
  - *Dematarialization vs. Immaterialization*
- Virtual substitutes for physical processes will never be functionally equivalent to the physical processes, but will always have some advantages and disadvantages as compared with physical processes
- People may not change their lifestyles for environmental savings alone, but might be more strongly motivated by some functional advantage in a cyberworld

Environmental information processing (EIP)	Public sector: Environmental Information Systems (EIS) operated by public authorities	Public awareness about condition of public goods
		Prerequisites for political decisions
		Executing instruments of environmental policy
	Private sector: Environmental Management Information Systems (EMIS)	Legal compliance
		Environmental reporting to stakeholders
		Eco-efficiency and material flow management
Information Society Technologies (IST)	Direct impact on material intensity of economy	Material intensity of ISTs' product life cycles
	Indirect impact on material intensity of economy	Substitution potential
		Optimization potential
		Induction potential

Baumer & Silberman: When the Implication is not to Design (Technology), CHI, 2011

- It is not obvious that the complex conditions associated with unsustainability—including environmental, political, social, historical, economic, and other factors—are best addressed with computing technology
  - *Could the technology be replaced by an equally viable low-tech or non-technological approach to the situation?*
  - *Does a technological intervention result in more trouble or harm than the situation it's meant to address?*
  - *Does a technology solve a computationally tractable transformation of a problem rather than the problem itself?*
- No single, simple solution will enable us to live sustainably
- Encourages attending to the complex ways technological interventions reconfigure the situations into which they are introduced

DiSalvo et al.: Navigating the Terrain of Sustainable HCI, Interactions, 2010

- A dominant genre in sustainable HCI is persuasive technology: systems that attempt to convince users to behave in a more sustainable way.
- Designers usually determine what constitutes “sustainable behavior,”
  - What counts as success is behavior change
    or decision making that aligns with the predetermined desired behaviors, although many papers in this genre do not evaluate sustainability
- How to approach users and their lifestyles: Individuals vs. groups/society
- Question whether a solution for sustainability can be achieved through technology alone, or perhaps at all
  - If technology is not the point, then what becomes the work of sustainable HCI?
- If we agree that fundamental change is needed and it might be change that users don’t want, who gets to decide what change should happen and how?

Preist et al.: Evaluating Sustainable Interaction Design of Digital Services: The Case of YouTube, CHI, 2019

- Application of Sustainable Interaction Design (SID) to understand and reduce the environmental impact of digital services
- Energy use and associated GHG emissions of systems of products and infrastructure associated with service use
- Identifying and eliminating digital waste
  - Evaluation should consider goal, mechanism, metric, method, and scope
- Corporate GHG strategy

Hindle: If you bill it, they will pay: Energy consumption in the cloud will be irrelevant until directly billed for, RE4SuSy, 2018

- We have limited motivation to investigate energy consumption in the cloud because cloud customers cannot necessarily realize savings
- Much like carbon-taxes, end-user billing of software energy consumption will promote reduced energy consumption or at least sustainable energy consumption
- If system operators are choosing software packages and services for their sustainability footprint then there will be significant pressure on software developers to address energy consumption as a first-class non-functional requirement in their software systems

Coroama & Mattern: Digital Rebound – Why Digitalization Will Not Redeem Us Our Environmental Sins, ICT4S, 2019

- Types of rebound effect
  - Direct - Jevon's paradox or Backfire
  - Indirect – Induction effect, Income and substitution effect, Producer rebound
  - Time rebound
  - General equilibrium effects and other macro level rebound
- Digitalization and its rebound
- Digitalization without rebound

Increasing the awareness of the effects of software and systems

Duboc et al.: Do we really know what we are building? Raising awareness of potential Sustainability Effects of Software Systems in Requirements Engineering, RE, 2019

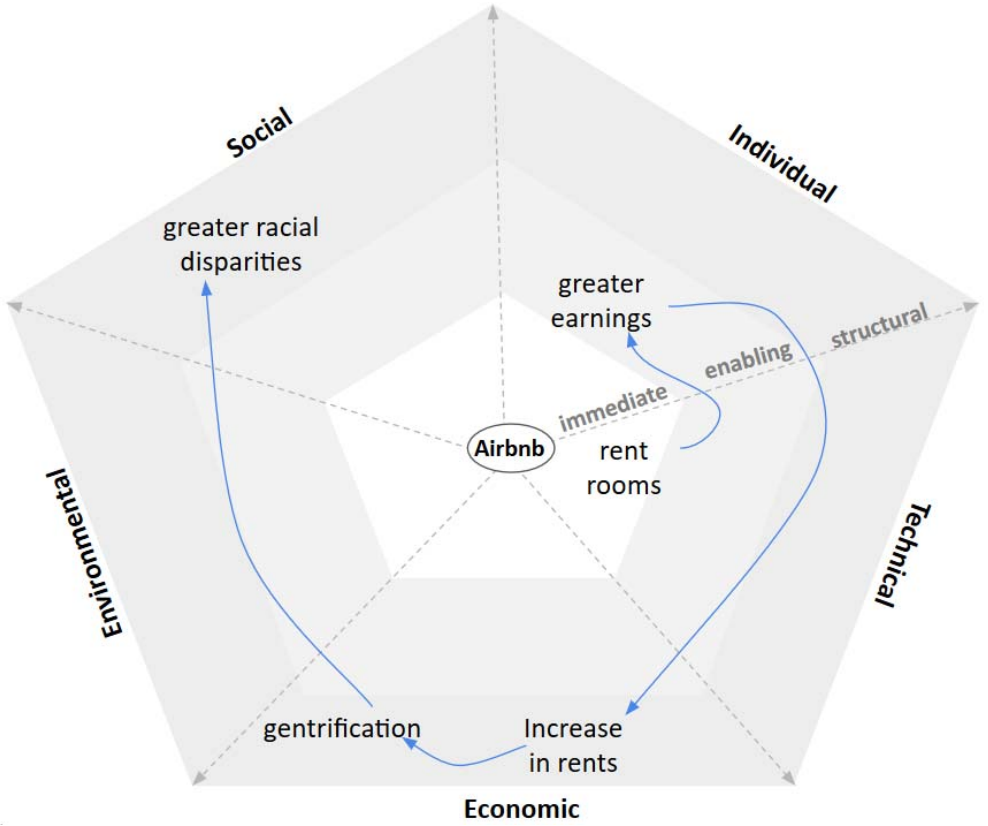
- question-based framework for raising awareness of the po-tential effects of software systems on sustainability

Social	(1) Sense of Community; (2) Trust; (3) Inclusiveness and Diversity; (4) Equality; (5) Participation and Communication;
Individual	(1) Health; (2) Lifelong learning; (3) Privacy; (4) Safety; (5) Agency;
Environmental	(1) Material and Resources; (2) Soil, Atmospheric and Water Pollution; (3) Energy; (4) Biodiversity and Land Use; (5) Logistics and Transportation;
Economic	(1) Value; (2) Customer Relationship Management (CRM); (3) Supply chain; (4) Governance and Processes; (5) Innovation and R&D;
Technical	(1) Maintainability; (2) Usability; (3) Extensibility and Adaptability; (4) Security; (5) Scalability;

Topic	Key Points - Social Dimension
SENSE OF COMMUNITY	rent rooms → personal contact → sense of community
	rating system → welcome and he
	high use → change house dynam
	high use → door codes → less p
	structural changes to properties
	high use → long-term renters for

SOCIAL DIMENSION (Interviewer copy. Tick questions as you advance in the interview.)

Specific Questions	Remind participants to consider...
SENSE OF COMMUNITY [ ] Normally people belong to an **organization**, to an **area** or to a **group of like-minded people**. Can the system affect a person's **sense of belonging** to these groups?	[ ] the **user community** and the **local community.** Say, for example: *you mentioned an effect on the sense of commun* *What about the people in the local community?*



Condori-Fernandez et al.: Using Participatory Technical-action-research to validate a Software Sustainability Model, ICT4S, 2019

- Sustainability Assessment Framework (SAF)
  - Validation of sustainability-quality model
  - Practitioners
- Confirms the multi-dimensional nature of sustainability

Characteristics	Attributes	Definition according to [9]	TECH	SOC	ENV	ECON
Compatibility	Co-existence	product can perform its functions efficiently while sharing environment and resources with other products.				
	Interoperability	a system can exchange information with other systems and use the information that has been exchanged.				
Context coverage	Context completeness	system can be used in all the specified contexts of use				
	Flexibility	system can be used in contexts beyond those initially specified in the requirements.				
Effectiveness	Effectiveness	accuracy and completeness with which users achieve specified goals.				
Efficiency	Efficiency	resources expended in relation to the accuracy and completeness with which users achieve goals.				
Freedom from risk	Economic risk mitigation	system mitigates the potential risk to financial status in the intended contexts of use.				
	Environmental risk mitigation	system mitigates the potential risk to property or the environment in the intended contexts of use.				
	Health and safety risk mitigation	system mitigates the potential risk to people in the intended contexts of use.				
Functional suitability	Functional appropriateness	the functions facilitate the accomplishment of specified tasks and objectives.				
	Functional correctness	system provides the correct results with the needed degree of precision.				
	Functional completeness	degree to which the set of functions covers all the specified tasks and user objectives.				
Maintainability	Modifiability	system can be effectively and efficiently modified without introducing defects or degrading existing product quality				+
	Modularity	system is composed of components such that a change to one component has minimal impact on other components.				
	Reusability	an asset can be used in more than one system, or in building other assets				+
	Testability	effectiveness and efficiency with which test criteria can be established for a system.				
Performance efficiency	Capacity	the maximum limits of a product or system parameter meet requirements.				
	Resource utilization	the amounts and types of resources used by a system, when performing its functions, meet requirements.				
	Time behaviour	response, processing times and throughput rates of a system, when performing its functions, meet requirements.				

Characteristics	Attributes	Definitions	TECH	SOC	ENV	ECON
Data Privacy	Data Privacy	privacy concerns arise wherever personally identifiable information is collected, stored, or used.				
Timeliness	Timeliness	the fact or quality of being done or occurring at a favourable or useful time.				
Regulation compliance	Regulation compliance	allows to draw conclusions about how well software adheres to application related regulations in laws.				
Scalability	Scalability	the ability of a computing process to be used or produced in a range of capabilities				
Tailorability	Tailorability	system's capability to allow users to create or enable new configuration of functionality as well as control information provision.				

Penzenstadler B. et al.: Software Engineering for Sustainability - Find the Leverage Points!, IEEE Software, 2018

- “Leverage points are places within a complex system (a corporation, an economy, a living body, a city, an ecosystem) where a small shift in one thing can produce big changes in everything.” Meadows D.H., 1999
- Software engineers need to be aware of the power of software systems as a transformational force in society and the significant impact that their designs can have.

Table A. Leverage points.[3]

Leverage point	Description
LP 12	Constants, parameters, and numbers. Tweaking parameters allows change to the intensity of the flows in systems but rarely alters the underlying dynamics.
LP 11	The sizes of buffers and other stabilizing stocks, relative to their flows. Stabilize a system by adjusting the capacity of its buffers, and make it more efficient by optimizing the flow.
LP 10	The structure of material stocks and flows (such as transportation networks and population age structures). Physical structure is crucial in a system but often hard to change; therefore, the leverage point is in proper initial design.
LP 9	The lengths of delays, relative to the rate of system change. A system cannot respond to short-term changes when it has long-term delays.
LP 8	The strength of balancing feedback loops, relative to the impacts they respond to. Balancing feedback loops help systems to self-correct by monitoring and adjusting according to the system goal.
LP 7	The gain around reinforcing feedback loops. Reinforcing feedback loops can be sources of system instability or mechanisms to amplify desired change, so adjusting their strength affects how the system responds to change.
LP 6	The structure of information flows. This can create a new feedback loop that was not there before. Altering the structure of information flows enables more agency by users.
LP 5	The rules of the system, including incentives, punishments, and constraints. Social rules include constitutions, laws, standards, policies, and incentives. Changing the rules of a system can change the behavior of the society under them.
LP 4	The power to add, change, evolve, or self-organize system structure. In biology, this is called evolution; in society, we call it empowerment. In systems terms, it is called self-organization, the strongest form of system resilience.
LP 3	The goals of the system. Changing the goal of a system is a powerful strategy to effect change but can be hard to achieve.
LP 2	The mind-set or paradigm out of which the system arises. Paradigms are a shared set of deep beliefs about how the world works. They are the hardest to change in a system, as society will fiercely resist any challenges to its paradigms.
LP 1	The power to transcend paradigms. This final and most effective LP is about being unattached to existing paradigms; there is no certainty in any particular worldview.