Sustainable Software Development

Definitions

- ***Sustainability***
  - Capacity to endure

- ***Sustainable Development***
  - Meeting the needs of the present without compromising the ability of future generations to meet their own needs

Question

What is sustainability in the context of Software Engineering?

Sustainability in the Context of SW Engineering …1

- Mahaux et al. [1] claim that
  - IT changes behavior
  - Therefore IT has considerable effect on society and environment, which is supported by
    - ***greenIT*** concepts and
    - analysis of the usage processes of a software system.

Sustainability in the Context of SW Engineering …2

- Naumann et al. [2] define
  - *green and sustainable software* such that direct and indirect negative impacts on economy, society, human beings, and environment that result from development, deployment and usage of the software have a positive effect on sustainable development
  - *Green and Sustainable Software Engineering*, such that the negative and positive impacts on sustainable development are continuously assessed, documented, and used for a further optimization of the software product

Three Parameters for Scope and Context

- Which system shall be preserved in
- Which function over
- Which time horizon?

Four Dimensions of Sustainability in SW Engineering

1. Development Process Dimension
2. Maintenance Process Dimension
3. System Production Dimension
4. System Use Dimension

Development Process Dimension

- Sustainability during the initial software development process means *development with responsible use of ecological, human, and financial resources*.
- *System*: a software system development company (including its staff as well as its operational environment with equipment)
- *Function*: to perform software development *with minimized environmental impact and a sufficient economic balance*
- *Time horizon*: dependent on the company size and the general duration of projects, can be estimated with the time horizon of a *long-term business plan*

Maintenance Process Dimension

- Sustainability of the software system during its maintenance period until replacement by a new system includes *continuous monitoring of quality and knowledge management*.
- *System*: a software system development company
- *Function*: to maintain and evolve a software system *with minimized environmental impact, a sufficient economic balance, and well-managed knowledge*
- *Time horizon*: dependent on the company size and the general maintenance period of projects, can be estimated with the time horizon of a *long-term business plan*

System Production Dimension

- Sustainability of the software system as product with respect to its use of resources for production is achieved, for example, *by using green IT principles, sustainably produced hardware components, and optimising the required logistics for assembly, etc.*
- *System*: a *conjoint of developing and supplier companies*
- *Function*: to *produce (assemble) a system with minimized environmental impact and a sufficient economic balance*
- *Time horizon*: *dependent on the project plan for production, according to the system's size and complexity*

System Utilization Aspect

- Sustainability in the use of processes within the application domain triggered by the software system as product takes into account *responsibility for the environmental impact and designing green business processes*.
- *System*: a system with its operational environment and business context in its application domain
- *Function*: to maintain and evolve a software system with minimized environmental impact, a sufficient economic balance, and social responsibility
- *Time horizon*: dependent on the system’s size and complexity

How to Improve Sustainability in Software Process

- Use of a common artificial model for documentation during development, facilitating collaboration in the initial development process and improving reuse or refactoring in the maintenance process.
- Knowledge management, for example, by company internal training and workshops to share best practises, improving both the initial development process and the maintenance process aspect listed in Sec. 3.
- Continuous quality assessment with feedback into the company's developer handbook, for example, Albertao measures sustainability performance of a software project according to standard quality properties [1].
- Optimization of resource usage during software development tasks, taking into account the energy consumption of used equipment (hardware and software).
- Consideration of working offline for certain tasks, for example, requirements elicitation with stakeholders and early interface design activities — naturally that saves only little resources, but it certainly improves communication and, potentially, creativity.
- Producing energy-aware software design as, for example, proposed by Naumann et al. in their guidelines for environmentally sustainable web development [11].
- Assessment of the project management using a Sustainability Maturity Model, for example according to Silvius et al. [10], then definition of goals to be achieved in further optimization activities.

References

[1] Martin Mahaux et al. Discovering Sustainability Requirements. In 17th Intl. Working Conf. on Requirements Engineering: Foundation for Software Quality, 2011.

[2] Stefan Naumann et al. The GREENSOFT Model. Sustainable Computing: Informatics and Systems, 1(1), 2011.

[3] Birgit Penzenstadler, What does sustainability mean in and for software engineering?, ICT4S, 2013

Maintainable Software Development

Importance of Maintainable SW Development

- A typical enterprise business application spends at least 4/5 of its time in maintenance.
- SW is intangible, i.e., no wear and tear. Why, then, does it ever need maintenance?
- Three reasons
  - Defect fixing: especially right after delivery
  - Changed or new business requirements: the less customer involvment the more change customer demands
  - Changes to software's execution context: SW's execution context is the required hardware and other SW (e.g., DB, O/S, libraries) for the relevant SW to run. Once they undergo a change, SW product requires maintenance.

10 Guidelines… first three

1. ***Use small modules*** (i.e., shorter pieces of code)
   - smaller methods and constructors are easier to analyze, test, and reuse.
2. ***Write simple units of code***
   - Units with fewer decision points are easier to analyze and test.
   - Do not use deep nesting since that increases complexity.
   - Closely related to guideline 1.
3. ***Write code once***
   - Duplication of source code should be avoided at all times, since changes will need to be made in each copy.
   - Duplication is also a source of regression bugs (bugs that occur whenever software functionality that previously worked as desired, stops working or no longer works in the same way that was previously planned).

10 Guidelines... 4-7

4. ***Keep unit interfaces small***
   - Units (methods and constructors) with fewer parameters (arguments) are easier to test and reuse. A valuable lesson here is to keep list of parameters to a method small.
5. ***Separate concerns in modules***
   - Modules (classes) that are loosely coupled are easier to modify and lead to a more modular system.
6. ***Couple architecture components loosely or use less dependent modules!***
   - Top-level components of a system that are more loosely coupled are easier to modify and lead to a more modular system.
7. ***Keep architecture components balanced***
   - A well-balanced architecture, with neither too many nor too few components, of uniform size, is the most modular and enables easy modification through separation of concerns.

10 Guidelines... Last three

8. ***Keep your codebase small***

- A large system is difficult to maintain, because more code needs to be analyzed, changed and tested.
- Also maintenance productivity per line of code is lower in a large system than in a small system.
- Refactor existing code... do not copy and paste code, and
- Use third-party libraries and frameworks to avoid necessary over-engineering.

9. ***Automate development pipeline and tests*** .

- Automated tests (i.e., test executed with no manual intervention) enable near-instantaneous feedback on the effectiveness of modifications. Manual tests do not scale.
- Design and add tests ideally when writing the code or even drive your design by testing, because you get in the habit of thinking about how your code can be tested.
- The immediate feedback and safety net of a regression suite, makes you less afraid to make changes which is an important requirement with ever-changing business rules.

10. ***Write clean code***

- Dead code in your codebase makes others inproductive and thus maintenance less efficient.
- Code requiring comments, shouldn’t it be refactored / redesigned? Especially commented code can be very confusing.
- No worries: git still has copies of everything!
- Long comments don’t tend to stay up2date with code changes turning well-intended comments into lies.