Perseverance in Sustainable Software Architecting

Roland J. Weiss
ABB Power Generation
Via Enrico Albareto, 35
16153 Genova, Italy
roland.weiss@it.abb.com

Daniele Repetto
ABB Power Generation
Via Enrico Albareto, 35
16153 Genova, Italy
daniele.repetto@it.abb.com

Heiko Koziolek
Industrial Software Systems
ABB Corporate Research
Ladenburg, Germany
heiko.koziolek@de.abb.com

ABSTRACT
In the recent past, there has been an increased interest in better managing the evolution of existing software systems and improving the software engineering practices for this now common task. In this paper, we take a look at the efforts at ABB to advance in this area, with special emphasis on architectures of long-living systems. The review consists of detailing the introduced methods and tools, as well as sharing experiences from applying them. In addition, we present two current case studies from the industrial automation domain that will be used as additional test fields for the developed methods.

Categories and Subject Descriptors
D.2.11 [**Software Architectures**]: Domain-specific architectures – *industrial automation, long-living software systems.*

General Terms
Management, Measurement, Design, Experimentation.

Keywords
Sustainability, long-lived software systems, validation.

1. INTRODUCTION
Software is controlling many aspects of our daily life. Every day new software is written and existing software is being updated and evolved. For long-living industrial software systems, the majority of investments flow into maintenance and evolution of the systems instead of greenfield development. These software systems with a life-span of more than 10 years have to be constructed according to special requirements to their design, structure, and internal quality. During their life-span, such systems have to be able to evolve in response to changes in the environment (i.e., hardware and software), usage profile (i.e., changed or increasing workload), and business demands (i.e., new features, changed business processes). Because these requirements may require expensive changes to a system, it is necessary to keep efforts and costs within acceptable limits during maintenance and evolution.

To address this pressing need, software evolution has become a topic of active research recently again. Within ABB, we have monitored these activities, as well as started several initiatives, either on our own or with partners, in order to improve our practices to sustainably evolve the software systems that we own.

From the economic sustainability point of view, we see three characteristics as important in industrial software-intensive systems: technical, organizational and financial sustainability.

Technical sustainability in a software-intensive system is achieved by selecting technologies that not only provide the required qualities but also provide a platform for future maintainability and evolution of long-lived systems.

Organizational sustainability ensures the right resources (people and tools) will be available to ensure development is conducted in the most efficient way.

Financial sustainability ensures the organization meets its expected revenues from the developed software.

In this paper we focus mainly on approaches - and experiences with them - to ensure technical sustainability.

2. A Toolkit for Sustainable Software Architecting and Engineering
In this section we will present the approaches we have investigated in order to improve sustainable software development within ABB. These approaches both consist of existing methods and tools, adaptations, and unique work.

2.1 Sustainability Guidelines
After an extensive literature survey [9] we have distilled the most important practices for sustainable software development into a small handbook called “Software Sustainability Guidelines”. The document provides a guide for explicit consideration of sustainability during system design, development, operation, and maintenance. Its objective is to support problem analysis and decision making for active incorporation of sustainability aspects. The structure of the document is aligned with the software lifecycle phases: Requirements, Architecture, Design, Implementation, Validation and Verification, and Maintenance. It also conveys some general aspects that are relevant for multiple lifecycle phases.

Each section provides an essential selection of approaches recommended for improving sustainability in the respective phase. Each approach and each section is accompanied with a list of risks that have to be kept in mind when applying the respective approach and a checklist that serves as a quick reference covering the most important sustainability questions of the respective phase.

In order to quickly grasp the essence of each approach, the descriptions follow a common description template. In the header, we summarise the name, relevance for evolution, addressed

Industry Day'12, June 27, 2012, Bertinoro, Italy.

problem of the approach, application, and learning efforts. The body section provides a short description, names supporting tools, explains usefulness, and enumerates risks and pitfalls.

The main audience for the guideline are development teams within ABB. There are at two possibilities how to select methods from this guide: look up per development phase or look up per evolution scenario. The latter one was introduced in order to have a fast way of checking only sections that are relevant to a specific evolution scenario that a development team faces. We have defined and grouped a number of generic evolution scenarios for software systems in the industrial automation domain based on a number of interviews with researchers in this area.

In order to give a taste of the content of the guidelines, we list both some of the present scenarios as well as approaches they relate to.

Evolutions scenarios (examples):

- Replacement of the user interface technology
- Change of the operating system
- Enhance or add in-house components

Sustainability aspects (examples):

- Architecture-Level Modifiability Analysis
- Design: bad smells, anti-patterns
- Maintenance: Architecture consistency checking

At the moment, the document is only available for internal use.

2.2 TechSuRe

Over the lifetime of a long lived software intensive system the (software) technologies used in the system will change. Changing technologies in these systems is typically extremely costly and painful for the organizations that have to perform these changes. Hence, wrong technology choices can jeopardize the economic viability of such systems in the long run. With TechSuRe [3] we introduced a method for assessing technology sustainability in long lived software intensive systems. The method makes sustainability issues an explicit part of a technology assessment and offers guidance on how to gauge the associated sustainability. Together, this supports making appropriate technology choices for long lived software intensive systems.

The TechSuRe method uses a bottom-up approach to establish expected sustainability risk and sustainability. The method starts with quantifying different indicators in a three-level discrete scale (e.g. high, medium, low). Next a reasoning framework is provided that, given the quantification of the individual indicators, guides one to reach conclusions.

The reasoning framework is represented as a set of tables where the left column holds the results of the reasoning given the indicators. We have not ranked the importance of each indicator, this is implicitly handled in the resulting estimate.

The result from applying our reasoning framework is a statement about the sustainability and associated sustainability risk (given as number) for using a particular software technology in a specific software intensive product. Consequently, the sustainability is only relevant in the context of a particular use.

Note, that the precise numbers are not the main contribution of the method. It is the process of analyzing a software technology from a sustainability point of view that really is the strength of the method.

2.3 Q-ImPrESS

Model-driven development holds the promise of smooth evolution once the models are defined and all relevant meta-data is in place. As part of the EU-funded FP7 research project Q-ImPrESS [5] we have investigated usage of model-driven development for enabling software architects to predict the impact of architectural design decisions on performance, reliability, and maintainability of a service-oriented software system. With such tooling in place, evolution of our software systems can be improved, resulting in shorter update cycles and higher conformance to the quality goals.

To evaluate the various features of the Q-ImPrESS method, we selected a process control system (PCS) from the automation domain. A PCS manages time-dependent industrial processes, e. g. power generation, pulp and paper handling, and oil and gas processing. It periodically collects sensor data like temperature, flow, and pressure from various field devices and visualizes it for human operators.

The operators use the system to manipulate actuators in the process, e. g. pumps, valves, and heaters. The system can automatically execute predefined actions and informs operators of irregular conditions using alarms.

After the EU project finished, we have applied part of the Q-ImPrESS method for guiding a redesign of a Robotics infrastructure system [2].

2.4 Tracking Sustainability Indicators

For the same kind of process control system that we applied the Q-ImPrESS method, we have also introduced a reporting framework for several novel architecture-level code metrics from recent literature [6].

We found more than 40 architecture-level code metrics in literature. They measure different aspects of sustainability, but most of them are related to the modularization quality of the system. It is argued that a clean modularization of a large system benefits maintainability, as it reduces system complexity, allows faster system understanding, and enables more easy replacement of modules during system evolution. Such modularization metrics concern for example, similarity of purpose within modules, encapsulation, compilability, extendability, testability, and module size. In order to track sustainability characteristics over time, the tool plots the evolution of user-selected metrics for selectable time intervals. It is also possible to directly compare only two builds. The type of graph can be chosen according to the capabilities of the spreadsheet tool. This typically includes the linear plots of Fig. 10 as well as Kiviat diagrams.

In addition, we added tooling to the build process for architecture enforcement. This is based on the face that over time the initial system structure may erode and layering rules may be broken. Violated layering rules severely impact maintenance costs, as they negate the benefits of modularization and complicate independent module compilability, extendibility, and testability [8].

This decay is often introduced unintentionally because of missing enforcement. Either the design documents become out of sync with the system or developers neglect to check for layering rule violations in subsequent builds. Two reasons are that layering violations do not have an immediate effect on functionality and that the architectural design documentation is often merely used for documentation purposes.

The tools NDepend (for C#) and CppDepend (for C++) check generic dependency rules out-of-the-box, such as disallowed dependency cycles between .NET assemblies. Based on fact

databases extracted from the source code, they produce design structure matrices (DSMs) for easily analyzing module dependencies and identifying cyclic dependencies. Additionally, developers can specify custom queries to the fact databases using the declarative Code Query Language (CQL). We used this extension mechanism to define CQL queries checking allowed dependencies from the architecture model.

2.5 Sustainability Evaluation

Finally, as part of several projects we have conducted architecture evaluations according to established methods that are used in industry [1][4]. We emphasized quality attributes related to sustainability, e.g. maintainability, adaptability etc.

One case study compared two versions of a system with respect to sustainability scenarios. The first version of the system reflects the architecture of currently available products, and the second version reflects the future architecture to address changing customer and market requirements. We have chosen ALMA as it focuses on modifiability as a single quality attribute which is also specific to our case study. It does not perform a more general trade off analysis between multiple quality attributes such as the Architecture Trade-off Analysis Method (ATAM). Second, it is less depending on the experience of the assessors than others such as SAAM. For example, it provides more guidance by the applications of the definition of the goals, the parts of the architecture to be considered as well as the steps to be performed in general. All of these aspects support a more repeatable method.

The following table provides a summary of the analysis of the change scenarios selected for the ALMA analysis. Each change scenario was analysed for both the current and future system, and the resulting sustainability rating is listed in the following table for the comparison.

The "sustainability rating" documents how the analysis rates the support for the individual scenario according to the available documentation. Possible ratings are:

- High: The scenario is explicitly considered in the architecture.
- Medium: The scenario is not explicitly considered but no blocking issues have been identified.
- Low: Issues that might block this scenario have been identified in the architecture and documentation.

Table 1. Overview of the sustainability ratings for the change scenarios analyzed with ALMA

Scenario	Current	Future
Third Party Component	Medium	Medium
UI Changes: Replacement of the User Interface Technology	High	High
Functionality level changes: Enhance in-house components	Low	Medium
Data level changes: Support for processing larger amounts of data	Low	Medium
Application level Changes: Change of the Operating System	Medium	High
Hardware Level Changes: Exploit Multi-Core Processors	Low	High

3. Experiences from Industrial Applications

In the previous section we have given an overview of activities executed at ABB (often with partners) in order to improve sustainable software architecting and engineering. In this section we summarize the key experiences from these activities.

3.1 Sustainability Guidelines

The guidelines have so far been applied to only one internal case study. Interestingly enough, we identified two key areas to improve the investigated software architecture and related processes:

- Traceability is not considered for requirements and architecture design.
- The framework is planned with a Copy-Paste-Instantiation for individual products, which might be a treat to the systems' sustainability.

Also, the results of this first case study triggered several requests for the document. Application of the guidelines document is now active in a few development projects.

3.2 TechSuRe

Applying TechSuRe typically does not require huge efforts from the development teams. Thus, it has been well perceived and applied in the last two years in roughly a dozen projects, both research and product development.

Overall, the feedback is positive. With some support from the researchers that created the method, it is easy enough to apply and provides the right amount of input for making key technology decisions.

3.3 Q-ImPrESS

Q-ImPrESS has of course been applied to the demonstrator that was part of the EU project. Beyond that we have used the performance prediction capabilities in another project [2].

On the one hand are the results promising, and development teams express interest in the results. However, the learning curve is steep, and the up-front investment in model creation and data-collection is high [5].

3.4 Tracking Sustainability Indicators

The framework for sustainability tracking and architecture enforcement has been integrated into the development process of one major software product unit within ABB. Both aspects have been well perceived as they are part of the build process and carry low overhead. Violations from architecture conformance checks have been added to the following sprint backlogs. The same is true for corrective actions to keep the sustainability metrics within the defined thresholds. Another major rollout within ABB of the framework is planned this year.

3.5 Sustainability Evaluation

Architecture evaluations are well accepted practice within major software development activities within ABB. They are either executed by internal or external review teams. We have even introduced a lightweight method for fast internal reviews. The one application of ALMA was positive, but more experience is required to make a final assessment with respect to cost/benefit of this approach.

3.6 Summary

The previous paragraphs summarized our experience with various approaches to improve sustainability. At this point we collect this qualitative information in a table for quick reference.

Table 2. Overview of experience with sustainability methods

Method	Ease-of-use	Expe-rience	Impact	Active
Guidelines	Medium	Low	Medium	Medium
TechSuRe	High	High	Medium	High
Q-ImPrESS	Low	Medium	High	Low
Sustainability Indicators	High	Low	Medium	Medium
Evaluation	Low	Medium	Medium	Medium

4. Perseverance and Future Work

In the previous sections we have seen which approaches we have selected and developed to further sustainable software systems within our organization. There are two main challenges that we still have to address to make these efforts sustainable themselves:

1. The methods have to be anchored in product development units and relying on experts from research units has to be reduced to a minimum.
2. A quantitative validation of the benefits of applying these methods has to be done. This is a long term effort and requires support from the whole organization.

As can be seen from Table 2, point 1 is on a good way. However, without proving the benefits also quantitatively, there is a high risk that they are getting used less and get obsolete. And we still have to define a good framework for this validation, thus we are at the research level with this point. At a high level, the key performance indicators are:

- Time between releases with certain added functionality
- Budget dedicated for maintenance/evolution activities

In the automation domain, and related to process control systems specifically, there are two interesting areas that we want to further study in the context of sustainability:

Connectivities: PCS have to be able to connect to controllers and other devices from legacy systems, competitor systems, and of course the current line-up. As most PCS vendors offer several such systems, a good sharing approach is of interest, at the same time increasing the sustainability demands.

Process graphics: PCS present the current system state from process graphics. However, significant changes in customer demands (portable devices, levels of details, pan-and-zoom, portable web access) are asking for sustainable solutions.

5. Related Work

The first systematic taxonomy of software evolution is typically attributed to Lehman with the notorious Lehman's Laws. This is a good starting point for classifying work related to evolution [7], which is the starting point for sustainability goals.

For a detailed survey of related work we recommend the extensive collection in [9].

6. Conclusions

We have presented the current practices and related experiences regarding sustainable software development at ABB. While we have made progress in recent years, we also have identified significant challenges in order to make sustainable software development sustainable itself.

7. ACKNOWLEDGMENTS

Our thanks go to the numerous colleagues that worked with us in the last years to establish the practice of sustainable software architecting at ABB, both cooperation partners and colleagues: Aldo Dagnino, Pia Stoll, Anton Janssen, Tommy Kettu, Magnus Larsson, Martin Naedele, Eric Harper, Ralf Reussner, Steffen Becker, Johannes Stammel, Zoya Durdik, Benjamin Klatt, Klaus Krogmann, Mircea Trifu, Raffaela Mirandola, Anne Martens, and the whole Q-ImPrESS consortium.

8. REFERENCES

[1] Ali Babar, M., and Gorton, I, 2004. Comparison of Scenario-Based Software Architecture Evaluation Methods. 11th Asia-Pacific Software Engineering Conference (APSEC'04), 2004.

[2] De Gooijer, T., Jansen, A., Koziolek, H., and Koziolek, A., 2012. An industrial case study of performance and cost design space exploration. In *Proc. 3rd Int. Conf. on Performance Engineering (ICPE'12),* ACM.

[3] Jansen, A., Wall, A., Weiss, R., 2011. TechSuRe - A Method for Assessing Technology Sustainability in Long Lived Software Intensive Systems. In *Proc. 37th EUROMICRO Conference on Software Engineering and Advanced Applications,* pp. 426-434.

[4] Kazman, R., Klein, M., Clements, P., 200. *ATAM: Method for Architecture Evaluation.* Carnegie Mellon University CMU/SEI-2000-TR-004.

[5] Koziolek, H., Schlich, B., Bilich, C., Weiss, R., Becker, S., Krogmann, K., Trifu, M., Mirandola, R., and Martens, A., 2011. An industrial case study on quality impact prediction for evolving service-oriented software. In *Proc. 33rd ACM/IEEE Int. Conf. on Software Engineering (ICSE'11)* Software Engineering in Practice Track, 776-785. ACM.

[6] Koziolek, H., 2011. Sustainability evaluation of software architectures: A systematic review. In *Proc. 7th Int. ACM/SIGSOFT Conf. on the Quality of Software Architectures (QoSA'11)*, 3–12, ACM.

[7] Cook, S., Harrison, R., Lehman, M.M., and Wernick, P., 2006. Evolution in Software Systems: Foundations of the SPE Classification Scheme. *J. Softw. Maintenance & Evolution: Res. & Pract. 18 (1): 1–35.* doi:10.1002/smr.314

[8] Sarkar, S., Rama, G. M., and Kak, A. C., 2007. API-based and information-theoretic metrics for measuring the quality of software modularization. *IEEE Trans. Softw. Eng., vol. 33, pp. 14–32*, January 2007.

[9] Stammel, J., Durdik, Z., Krogmann, K., Koziolek, H., and Weiss, R., 2011. *Software evolution for industrial automation systems - literature overview.* Technical Report 2011-2, Karlsruhe Institute of Technology.