Software architectures that last

Intelligent software architectures create value and safeguard product investments in the short, medium and long term

ALDO DAGNINO, PIA STOLL, ROLAND WEISS – **The proportion and complexity of software in almost all ABB products is now higher than at any time in the past – a trend that is accelerating. Indeed, some products are purely software. As such industrial software has become ever more sophisticated, and critical, its long-term maintainability and sustainability have become very important factors for a good return on investment over its entire lifetime. Therefore, it is essential that it is founded on an appropriate and durable architecture.**

The world in which we live is based on an incredibly intricate spider's web of technology: Our power grids delicately balance a huge range of generation and distribution assets and effortlessly provide us with guaranteed power at the flick of a switch; a fantastically complex chain of technological wonders transports an oil molecule from a subsea reservoir to the nozzle of the pump at our local gas station; every item we buy and consume reaches us through a amazingly complex series of co-ordinated actions, mostly hidden to us. As individuals, we directly interact with only the tip of this technology iceberg.

ABB supplies many of the products which, unseen, provide this material structure of our society. At the heart of much of this, by now often essential, infrastructure is software. And the proportion and complexity of this software is now higher than ever before – a trend that shows no signs of abating, quite the reverse, in fact.

Some ABB products are purely software. In others the software and hardware components work together intimately and in yet others the software is embedded in the product hardware itself. They are found in almost all applications in the industrial world: in utilities, in process industries (such as pulp and paper, oil and gas, petrochemical, pharmaceutical, chemical, etc.) and in all kinds of manufacturing plants.

Such a high degree of software content makes products very adaptable, imbues them with powerful decision-making capabilities and fosters a higher degree of system autonomy. This has, in turn, shifted the operators' role from one in which they use their expertise to manually set control values, to one of supervision, fine-tuning and fault finding. A modern industrial system can nowadays control a process with minimum operator inter-

> To obtain a compelling return on investment, an industrial software-intensive system must be fully sustainable over decades.

Title Picture
Sophisticated software lies at the heart of much of the seen and unseen technology which supports our daily lives. Under the surface of this cityscape, for instance, is an entire world of complex and indispensible software systems. But whether the system is a building management software controlling a 100-storey building or a global share trading package running in one of the companies housed there, they all have one essential quality in common: a solid and robust software architecture.

The architects need to have an understanding of how the stakeholders' evolving business environment can influence software architectural requirements.

vention and autonomously interact with a multitude of other systems in the plant.

Further functional synergy is created when software components interact with each other in a way the hardware parts could not. In short, all this software content creates significant additional value for ABB's customers.

However, there is one very critical aspect of such a sophisticated software system: its maintainability and sustainability. To obtain a compelling return on investment for both the customers and the development organizations, an industrial software-intensive system must be able to be maintained in a cost-effective way and stay operational, in a fully-supportable way, for decades, ie the system must be fully sustainable.

Over such a long time, this sustainability will face challenges: new, and perhaps radically changed, technologies; new stakeholder requirements; new organizations and re-organizations; key expertise emigration; and changing business goals. In addition, software-intensive systems often have an inherent legacy heritage that significantly impacts software architecture and design going forward. If the organization in the past accurately predicted today's stakeholders' needs and adapted the development to suit, the incorporation of today's concerns in the system should be fairly straightforward. In the same fashion, today's organization should predict future stakeholders' needs and select the most important concerns to address.

To do this, the architects need to have an understanding of how the stakeholders' evolving business environment can influence software architectural requirements. For example, industrial software-intensive systems are often impacted by company mergers and acquisitions, where two or more systems have to be consolidated into one or perhaps share a core part.

Furthermore, stakeholders can include customers, end users, developers, project managers, product managers, maintainers and others, each with different and often conflicting expectations. The architect must reconcile these and balance them with technical and economic constraints.

Sustainability is, therefore, related not purely to software structures and their interactions but also to their environment in terms of enterprise aspects such as organization, business, tactics and scope [1].

To meet all the challenges described above and thus preserve the integrity of sophisticated software systems over, potentially, many decades, one very important prerequisite must be fulfilled: the systems must sit on a very solid software foundation. And this is where the role of the software architect becomes crucial.

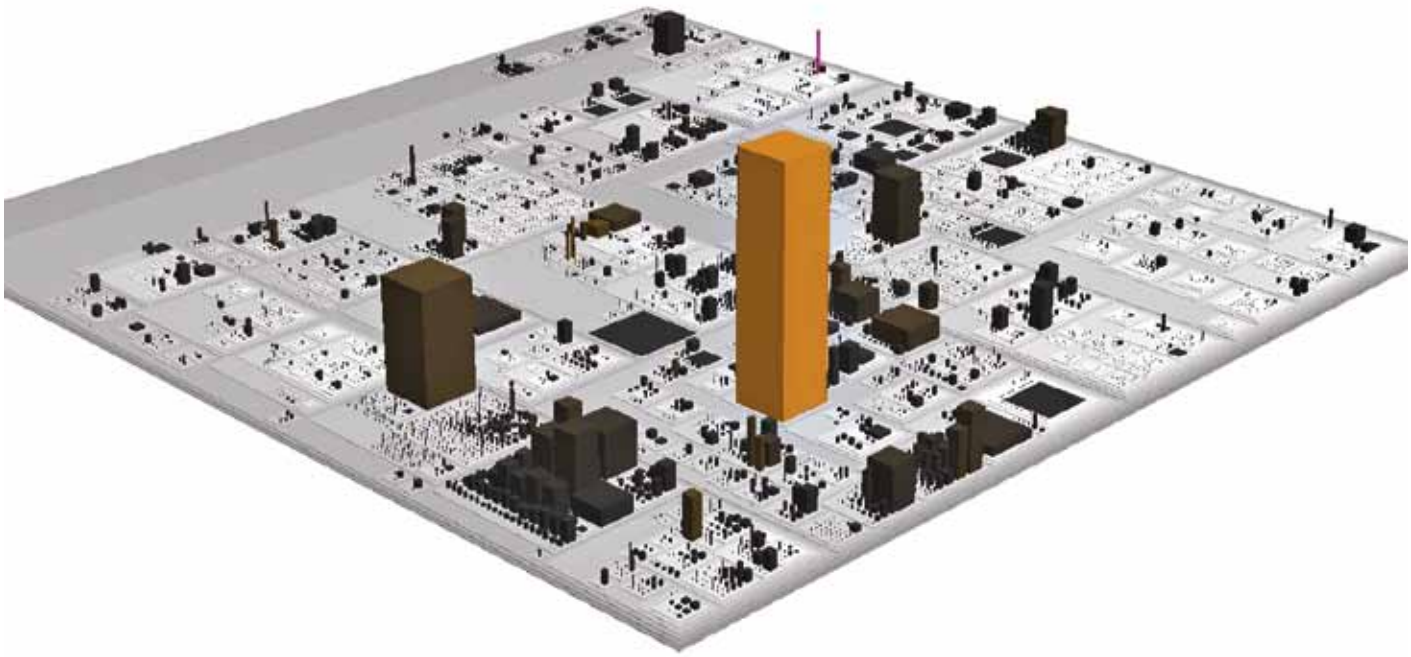
Software architecture

The study of software architecture is, in large part, a study of software structures and their interactions. This began in 1968, the year in which the term "Software Engineering" was introduced when Dijkstra presented his work with the THE-multiprogramming system. Dijkstra showed a layered software structure that supported the testability quality of the system, thereby connecting the software quality "testability" to software architecture structures [2]. Twenty years later, Shaw described different architecture styles [3]. She wrote:
"... important decisions are concerned with the kinds of modules and subsystems to use and the way these modules and subsystems are organized. This level of organization, the software architecture level, requires new kinds of abstractions that capture essential properties of major subsystems and the ways they interact". Shaw describes common ways to solve specific problems and concepts to solve a particular problem. An example of the latter is the "Blackboard" architectural model, where a common knowledge base, the "blackboard", is iteratively updated by a diverse group of specialists, starting with a problem specification and ending with a solution. This was applied, for example, to solve early software problems in speech recognition.

"We shape our buildings; thereafter they shape us".

WINSTON CHURCHILL, TIME, SEPTEMBER 12, 1960

1 System architecture and code visualization using the city analogy



On the formal side, the ISO/IEC 42010:2007 standard defines system architecture as: “The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution”.

Software architecture can be visualized as if the constituent components were buildings in a city. In the physical world, a secure building could, for example, be realized by allowing only one road, guarded by a watchman requiring a password, to lead to it. The software corollary would allow only one access possibility, from secure, authorized sources, to a software component. Software architecture researchers are constantly seeking innovative ways to design their “city plans” in order to positively influence software usability, security, performance, reliability or energy efficiency.

This “city” analogy has indeed been used in architecture visualization, where components/packages are represented by districts and classes by buildings whose sizes are determined by code metrics, eg code size or cyclomatic complexity (codecity.inf.usi.ch) ➔ 1.

Architecting a system is a process because a sequence of steps is prescribed to produce or change the architecture within a set of constraints. Architecting a system is a discipline, too, because a body of knowledge is used to inform practitioners as to the most effective way to design within a set of constraints. System architecture is primarily concerned with the internal interfaces between the system’s components or subsystems and the interface between the system and its external environment, especially the user.

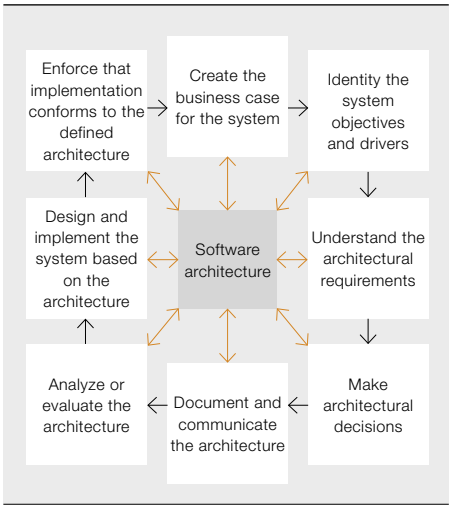
Architecture patterns of ABB’s industrial software systems

Christopher Alexander is a building architect researcher. In the book “The Timeless Way of Building”, published 1979, he describes common architectural patterns in space, events and human existence, at all levels of granularity. According to Alexander, “each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution”.

Alexander’s thinking regarding building patterns has inspired many of the software community’s architects. Software architecture patterns describe the core of a solution to software problems that occur over and over again. While Alexander focuses on the usability quality, namely the user’s experience of the building, the software architecture patterns address software qualities, such as

Software architecture can be visualised as if the constituent components were buildings in a city.

2 Software architecture methodology.



3 Graphical user-interface

security, performance, reliability, availability, maintainability and so on.

ABB's industrial software systems exhibit different types of architecture patterns. Some commonly observed ones include: client-server, event-driven, multi-tier and data-centric. These are briefly explained below.

Client-server

Client-server computing is a distributed application architecture that partitions tasks or workloads between service providers (servers) and service requesters, called clients. Often clients and servers operate over a computer network on separate hardware components. A server computer is a high-performance host that runs one or more server programs that share its resources with its multiple clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await the incoming requests from the clients [4].

Event-driven architecture

An event is defined as a significant change in a particular system state, eg incoming information from I/O components. Event-driven architecture may be applied by the design and implementation of systems that transmit events among loosely-coupled software/hardware components and services. An event-driven system typically consists of event generators and event consumers. Event consumers have the responsibility of instigating a reaction as soon as an event is presented. Such an architecture facilitates more responsiveness because event-driven systems are, by design, more normalized to unpredictable and asynchronous environments [5]. Many ABB systems operate in such a way that external input is continuously received, processed and appropriate actions are taken, eg process control or manufacturing.

Multi-tier architecture

Multi-tier architecture, or n-tier architecture, is a client-server architecture in which the user interface, the system processing capability and the data management are logically separate processes. For example, middleware which services data requests between a user and a database employs multi-tier architecture. The most common is three-tier architecture. The concepts of layer and tier are often used interchangeably, though many subscribe to the view that a layer is a logical structuring mechanism for the elements that make up the software solution, while a tier is a physical structuring mechanism for the system infrastructure [6].

> Software architecture evaluation and development at ABB is framed by important principles constituting an established methodology.

Data-centric architecture

Here, databases play a central role as such systems typically use a Database Management System (DBMS) as a major system engine. These contain a set of stored procedures that run on the database servers and have table-driven logic. The database-centric approach primarily leverages the indexing, transaction processing, integrity, recovery and security capabilities provided by high-end database systems [7].

Software architecture principles employed at ABB

Software architecture evaluation and development at ABB is framed by important principles which constitute an established methodology [8] → 2:

Create the business case for the system

The business case constrains requirements and provides a guide for determining the software qualities.

Identify system objectives and drivers

Guided by the business case, a system's objectives and primary drivers have to be identified, eg in a Quality Attribute Workshop. These drivers have to be taken into account when analyzing the system requirements and when making architectural design decisions.

Understand the architectural requirements

These have typically two components: the functional and the non-functional (or quality) elements. Architectural functional requirements define the basic functionality of

4 Mission critical system

the system and the architectural non-functional requirements, or quality attributes, define the behavioral and quality requirements, eg usability or performance.

Make architectural decisions

The desired quality attributes of a system determine the shape of its architecture. Specific tactics that address these are embedded in the system.

Document and communicate the architecture

To be an effective element of the software design, the architecture needs to be clearly documented and efficiently communicated to all relevant stakeholders, bearing in mind the diversity of their backgrounds (developers, testers, customers, managers, etc.) This documentation should also illuminate the decision-making process which leads to the target architecture.

Analyze or evaluate the architecture

The software architecture must be evaluated for the qualities that it supports to ensure the system satisfies the needs of the relevant stakeholders. Scenario-based techniques are effective tools to evaluate software architectures.

Design and implement the system based on the architecture

Having a well-documented and clear set of architecture documents is imperative for software designers and developers to remain faithful to the defined architecture.

Ensure implementation conforms to the defined architecture

The culture of the organization should support the maintenance of both the code and the architecture, especially once the system is in maintenance mode.

Use of software architectures methodology at ABB

The methodology described in → 2 is used at ABB in different ways. Firstly, to evaluate if the architecture of a current product still meets the quality attributes that the market expects, especially as customers' expectations evolve over time. Secondly, to evaluate new and emerging technologies that could be employed to re-develop or enhance an existing product. Thirdly, to develop a new or revised product architecture to meet the quality attributes and functionality expected by the customer. Finally, the architecture methodology can be employed to verify and validate a newly-created product architecture by evaluating the architectural scenarios generated. Examples of these four cases are provided below, based on projects conducted by ABB Corporate Research together with various ABB Business Units.

Evaluate architecture of existing product

The Architecture Tradeoff Analysis Method was developed by the Software Engineering Institute (SEI) in Pittsburgh, USA. ABB employs the method to evaluate architectures of both new and existing software products. The strength of the method lies in the analysis result, which shows how different quality attributes trade-off with each other and what business case they support. In the case described here, the ATAM review's customers had questions related to the usage of a code-generating tool for embedded code modules. It was not clear if

The culture of the organization should support the maintenance of both the code and the architecture.

The business goals and benefits to the customer of replacing the GUI included reduction in maintenance costs, enhancing the system's scalability and improving the system's performance.

the tool produced code modules that were optimized for performance, since the developers of the tool's generation engine had focused on the portability of the code modules.

However, by using the ATAM it was possible to demonstrate to the customer that the tool's generation engine produced code with an architecture that could be slightly more performance-efficient, at the cost of being less portable. The ATAM review of the customer's business case showed that the portability was no longer prioritized in the same way as it was at the time of the tool's development. This showed the customer that they could target optimization of the software's performance instead of software portability without losing business.

Evaluate emerging software technologies

Yet another project evaluated emerging software technologies that could be used to create a new, replacement Graphical User Interface (GUI) ➔ 3 for an operations management software system. The business goals and benefits to the customer of replacing the GUI included reduction in maintenance costs, enhancing the system's scalability and improving the system's performance. All these directly translated into the software qualities employed to create architecture and technology options and evaluate them. An analysis of the system architecture associated with the selected technologies was conducted. A set of architectural requirements was elicited in conjunction with the business unit that guided the evaluation of these technologies. A subset of the requirements identified was selected together with the business unit and this subset was used to create a scenario which was, in turn, employed to evaluate the technologies through the development of prototypes. Based on the results obtained in the creation of the prototypes, two competing technologies and corresponding architecture options emerged, one of which was eventually chosen after a subsequent prototype stage.

The customer saw that they could target optimization of the software's performance instead of software portability without losing business.

Develop new architecture

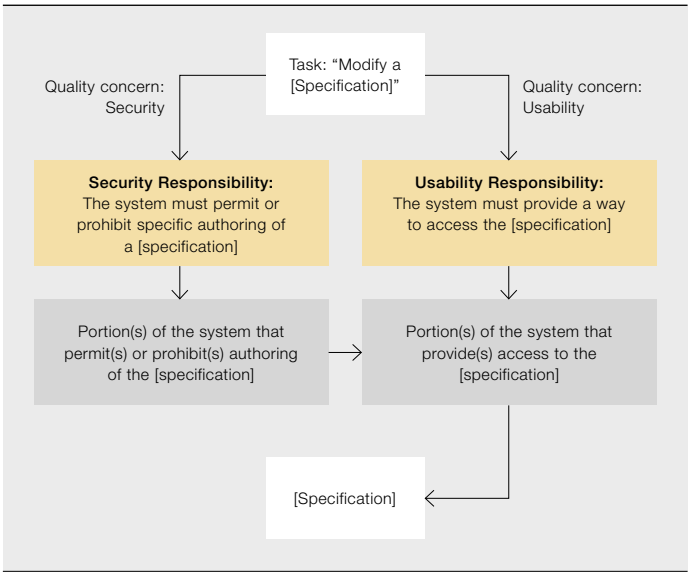
Attribute-driven Design Methodology

A system that integrated a mission-critical ABB product ➔ 4 with a wide array of third-party applications used at customers' sites, and that extracts data from these applications for later use, was architected using the attribute-driven design methodology [9]. Several drivers guided the development of the system's architecture: First, the system should integrate with a wide range of third-party applications seamlessly. Second, the system was to have the capability to collect large amounts of data from the third-party applications. Third, it was necessary that the system was perceived by its users as being very fast. These drivers were used to define the primary quality attributes for this system as integrability, scalability, performance and security. The quality requirements were then utilized to create the scenarios needed to build and evaluate the architecture options for the system and select the best one. Once the system architecture was selected, a system prototype was built and demonstrated to customers. This served as an excellent way to obtain their input so the final system could be developed.

Usability-supporting architecture patterns

The next example of new architecture development deals with supporting usability. One user task in a software system can have multiple quality concerns ➔ 5. Often security and usability have to be traded off. Security is all about preventing inappropriate user access and usability is all about facilitating appropriate user access. In the Usability-Supporting Architecture Patterns, USAP, the term "responsibility" is used for the general sub-tasks the software system has to support to ensure the usability quality of the main task. For each responsibility the USAP provides architectural implementation instructions ➔ 6.

5 Multiple quality concerns of one task [1]



6 Screen shot of the USAP tool

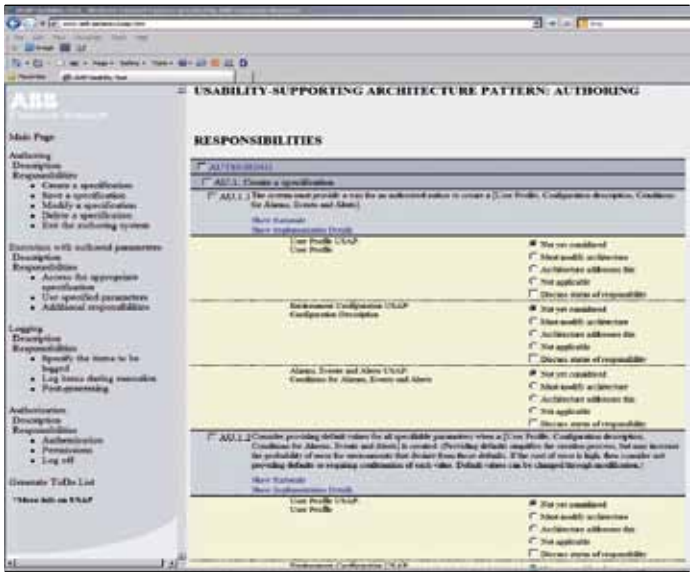


ABB undertook a USAP study in the domain of sustainable industrial software systems and contributed a description of an enhanced research method and a software tool that visualizes the method's constructed responsibilities.

The tool, visualizing the responsibilities, acts as an experience factory [10] housing reusable architectural knowledge for a set of system environment interaction scenarios in the form of a check-list. Three scenarios were hosted with a check-list of forty-two architectural responsibilities describing how the architecture can be revised to accommodate the usability requirements. One of the scenarios was the "Alarm & Event" interaction between the system and its environment. Two ABB architects who used the tool for six hours estimated that this time spent saved them five weeks' effort by allowing them to understand the usability requirements early on [11]. Three aspects of this study are significant:

- The usability-supporting patterns are primarily described at the level of responsibilities. These are independent of implementation and lead the architects to think about how a particular responsibility relates to their current system design
- Using textual descriptions for implementation instructions rather than diagrams was well received by the architects at ABB. In the study's first group, the architects showed some reluctance with respect to diagrammatic instructions. Using textual instructions, the tool enables the architects to investigate one aspect of the pattern at a time instead of forcing them to overlay an entire visual diagram of a pattern on their design to identify gaps.
- Using a tool to encourage the architects to examine all of the items in the checklist makes the architect go through all aspects of the patterns.

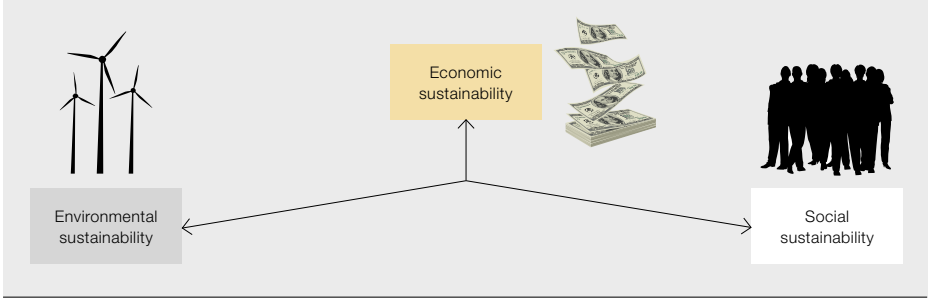
In addition, there is nothing in the USAP delivery tool that is specific to usability patterns. Any quality attribute where the requirements can be expressed as a set of responsibilities, eg security, could probably be included in the tool. The same portions of a system could then be represented from both the security and the usability responsibilities implementation points of view.

Verify and validate new architecture

A development team responsible for a major update of an ABB software system spent significant effort creating a new architecture for this next system version. As ABB Corporate Research was part of the architecture creation process, a neutral entity was asked to conduct an architecture evaluation. This external company used the architecture documentation from the project team and the results of a Quality Attributes Workshop to baseline the demands on the system's qualities. They then interviewed all relevant stakeholders, including Business Unit (BU) management, product management and system architects. After

Two ABB architects who used the tool for six hours estimated that this time spent saved them five weeks' effort by allowing them to understand the usability requirements early on.

7 Triple bottom line of corporate sustainability



> "When we build, let us think that we build forever."
>
> JOHN RUSKIN 1849

analysis of the material and interviews, the external reviewers presented their findings. The key results were:

- The main architectural decisions around the new architecture were sound and address the project's primary objectives.
- The architecture documentation was not precise enough in some areas and needed further refinement to avoid erroneous design decisions.
- The reasoning behind design decisions was not part of the architecture documentation. This makes future evolution more difficult and leaves room for deviations from the target architecture.

Overall, the workshop and the subsequent external review provided the BU stakeholders with the necessary confidence in the proposed new architecture and also pointed to topics that needed more attention and further elaboration.

Focusing on sustainable software architectures

It is evident from the preceding discussion of software architecture that the discipline has expended much effort to lay sustainable architectural foundations. But what particular aspects demand attention so that sustainability is maximized?

In 1849, John Ruskin wrote, "when we build, let us think that we build forever." Ruskin, then, was referring to the architecture of buildings, but the saying is as relevant today in the software world.

Within a sustainable architecture there is a focus on the process as well as the end product and while the product may "wear out" over time, the process remains. This process can then be repeated without resort to major external inputs. In the building industry, sustainable architecture brings together at least five key characteristics:

- Technical sustainability: can skills be introduced and passed on to others, and are the required tools accessible?
- Organizational sustainability: is there a structure that allows one to bring together the different stakeholders without, for example, needing to call on outside expertise on each occasion?
- Financial sustainability: can money or service exchange be accessed to pay for the work that needs to be done?
- Environmental sustainability – does the approach avoid depleting natural resource bases and contaminating the environment?
- Social sustainability: does the overall process and the product fit within, and satisfy the needs of, society?

Economic sustainability represents one of the "triple-bottom-lines" [12] of corporate sustainability → 7. From the economic sustainability point of view, three of the characteristics above are important in software-intensive systems: technical, organizational and financial sustainability.

Technical sustainability in a software-intensive system is achieved by selecting a technology that not only provides the required qualities but also provides a platform for future maintainability and evolution of long-lived systems. Issues such as developers' skills and compatibility with other company's products are important factors.

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. It is impor-

tant to ensure that the right processes are implemented and followed to reduce non-value added costs such as re-work, cost of poor quality, etc.

Software architectures can also contribute to the environmental sustainability axis → 7. This is impacted by the software system's structures and inter-operations. Software architecture designed to limit the energy consumption of the product increases environmental capital. Social Sustainability can be increased if the architecture is structured in a way that simplifies the developers' daily work and stimulates and motivates them.

Outlook

The importance of systematic software architecting has been identified within ABB development organizations. Most development units have established the software architect's role and increasingly adopt software architecture methodologies such as attribute-driven design.

At the same time, ABB continues to investigate ways to improve the architecture discipline in areas with potential for ABB, by, for example:

- Identifying and cataloging best practices for developing systems with sustainability as a high-importance quality attribute, like ABB's distributed control systems.
- Evaluating the benefits and applicability of software product line architectures as a basis for software development within ABB, as well as fostering systematic and coarse-grained software reuse.
- Developing methods for making design decisions early in the development process, eg not relying on extensive prototyping. In this regard, ABB has participated in the publicly funded research project Q-ImPrESS (www.q-impress.eu) that targeted predictions of changes to the performance, reliability and maintainability quality attributes.
- Deriving concepts for future automation systems for increased modularity, maintainability, scalability and portability.

Most development units have established the software architect's role and increasingly adopt software architecture methodologies such as attribute-driven design.

Aldo Dagnino
ABB Corporate Research
Raleigh, NC, United States
aldo.dagnino@us.abb.com

Pia Stoll
ABB Corporate Research Center
Västerås, Sweden
pia.stoll@se.abb.com

Roland Weiss
ABB Corporate Research Center
Ladenburg, Germany
roland.weiss@de.abb.com

References

[1] Stoll, P. (2009). Exploring Sustainable Industrial Software System Development within the Software Architecture Environment. Malardalen University Press, Vasteras, Sweden.
[2] Dijkkstra, E. (1968). The Structure of the T.H.E.-Multiprogramming System. Communications of the ACM 11, 5:341–46.
[3] M. Shaw. Larger scale systems require higher-level abstractions. Proceedings of the Fifth International Workshop on Software Specification and Design 1989.
[4] Berson, A. (1996). Client/server Architecture, McGraw-Hill, Inc., New York, NY, Second Edition.
[5] Hanson, J. (2005). Event-driven Services in SOA. Javaworld. January 31st. http://www.javaworld.com/javaworld/jw-01-2005/jw-0131-soa.html (Retrieved 2009/9/16)
[6] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M. and Tantawi, A. (2005). An Analytical Model for Muliti-tier internet services and its applications. Proceedings of the 2005 SIGMETRICS.
[7] Manuel, P. D. and AlGhamdi, J. (2003). A data-centric Design for n-tier Architecture. Information Sciences, vol. 150, issues 3–, April, pp. 195–06.
[8] Bass, L., Clements, P., Kazman, R. (2003). Software Architecture in Practice. Second Edition. Addison-Wesley. Pearson Education Inc. Boston, MA.
[9] Shaw, M. and Garlan, D. (1996). Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall.
[10] Basili, V. R., Caldeira, G. and Rombach, H. D. (1994). Encyclopedia of Software Engineering, chapter: The Experience Factory. Wiley.
[11] Stoll, P., Bass, L., John, B. E. and Golden, E. (2009). Supporting Usability in Product Line Architectures. Proceedings of the 13th International Software Product Line Conference (SPLC). San Francisco, USA. August 2009.
[12] Dyllick, T. and Hockerts, K. (2002). Beyond the business case for corporate sustainability. Business Strategy and the Environment, 11:130–41, 2002.