Making Architectural Knowledge Sustainable – The Y-Approach
Industrial Practice Report and Outlook

Olaf Zimmermann, ABB Corporate Research, Switzerland
With contributions from Heiko Koziolek and Martin Naedele

Agenda

- Context
  - Software and software research at ABB
  - Distributed process control systems
  - Domain-specific design challenges
- Sample projects and initiatives
  - Software sustainability
  - Q-ImPrESS performance modeling
- Architectural Knowledge Management (AKM) practices
  - Decision rationale – past and present
  - Capturing advice – relevant issues, good justifications
  - Towards a sustainability guide for AKM
- Summary
Agenda

- **Context**
  - Software and software research at ABB
  - Distributed process control systems
  - Domain-specific design challenges
- **Sample projects and initiatives**
  - Software sustainability
  - Q-ImPrESS performance modeling
- **Architectural Knowledge Management (AKM) practices**
  - Decision rationale – past and present
  - Capturing advice – relevant issues, good justifications
  - Towards a sustainability guide for AKM
- **Summary**

A Global Leader in Power and Automation Technologies

- 133,600 employees in about 100 countries
- $37.990 million in revenue (2011)
- Formed in 1988 merger of Swiss and Swedish engineering companies
- Predecessors founded in 1883 and 1891
- Publicly owned company with head office in Switzerland
Power and Productivity for a Better World
ABB’s Vision

As one of the world’s leading engineering companies, we help our customers to use electrical power efficiently, to increase industrial productivity and to lower environmental impact in a sustainable way.

Software at ABB
Software – Intelligence for ABB Products

Software is part of most ABB products – from the very small to the very big

- Pressure sensor
- Industrial control system
- Power grid control center

... and all have highest requirements for
- Real-time performance
- Reliability
- Long lifetime
- Remote connectivity
Examples
- Power generation, transmission, and distribution
- Production line at car manufacturer
- Mine, tunnel, paper mill
- Building automation

Source: M. Hollender, Collaborative Process Automation Systems, ISA 2010

Collaborative Process Automation Systems
Automation Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Hardware/Software Systems</th>
<th>Typical Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Resource Planning (ERP)</td>
<td>Enterprise Resource Planning (ERP)</td>
<td>Production planning (coarse), order management, logistics, plant production and scheduling, asset management</td>
</tr>
<tr>
<td></td>
<td>Enterprise Asset Management (EAM)</td>
<td></td>
</tr>
<tr>
<td>Manufacturing &amp; Execution</td>
<td>MES, MIS, LIMS</td>
<td>Production planning (detailed), production data and gathering, KPIs, materials management, quality mgmt, Scheduling, reliability assurance</td>
</tr>
<tr>
<td>Application servers, supervision &amp; control</td>
<td>Process Control System (PCS)</td>
<td>Operate and observe, recipe management, Archiving of measurement data (historian)</td>
</tr>
<tr>
<td></td>
<td>Distributed Control System (DCS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Human Machine Interface (HMI)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supervisory Control and Data Acquisition (SCADA)</td>
<td></td>
</tr>
<tr>
<td>Automation controllers</td>
<td>SPS, control loops</td>
<td>Batch control, continuous control, discrete control</td>
</tr>
<tr>
<td>Sensors, actuators, field buses (and managed process)</td>
<td>Process signals, I/O modules, fieldbuses</td>
<td>Interface to technical production process via signals, Simple and rapid data collection, mostly binary signals</td>
</tr>
</tbody>
</table>
PCS Infrastructure (Operational Model)

- Plant / Office Network
- Remote Workplaces
- Network Isolation Device
- Servers
- Internet
- Remote Workplaces
- Firewall
- Plant / Office Network
- Workplaces
- Redundant Network
- Controllers
- Remote I/O and Field devices
- Fieldbus
Software at ABB
Solution Approach Using Enterprise Processes

Mining Ops Management
- Explore
- Design / Construct
- Mine
- Process
- Trade
- Operation Analysis

Asset Management
- Asset Policy
- Capital Work
- Register Asset
- Develop Strategy
- Deploy Strategy
- Asset Analysis

Work Management
- Create
- Plan
- Schedule
- Execute
- Complete
- Work Analysis

Materials Management
- Determine Requirements
- Source Materials
- Procure
- Use Materials
- Distribution
- Performance Analysis

People Management
- Maintain Organization
- Workforce Planning
- Organizational Development
- Time and Attendance
- Pay and Benefits
- HSE

Financial Management
- Business Config and Financial Model
- Accounts Payable
- Accounts Receivable
- Fixed Assets
- General Ledger
- Performance & Compliance

Software at ABB
Integration Platform Architecture

Oracle
SAP
Legacy Application
Ventyx Mobility
Ventyx IMS
Ventyx Ellipse
Ventyx Axis

Service Interface
Service Interface
Service Interface
Service Interface
Service Interface
Service Interface
Service Interface

Routing
Transformation
Service Orchestration

Service Registry
System Management
Security
Scheduling Services

Integration Designer IDE
Management Console

©2012 Ventyx, An ABB Company | 14

©2012 Ventyx, An ABB Company | 15
Software at ABB
Challenge and Opportunity

Well-proven technology and
15-20 years life time

Standardized
products

Build on fast
development in
IT and electronics

Customized
solutions

Design Challenges in Automation & Power Domains
(for Hardware and Software)

- Safety and security
  - E.g. Security guidelines from North American Electric
    Reliability Corporation (NERC)
  - E.g. Stuxnet threat (for entire industry)
- Remote locations
  - Unmanned plants
  - Extreme environmental conditions
- Diversity and lifetime of installed base
  - 1000s of products
  - Some of them 40+ years old; news ones to last long
- Technology evolution (and debt)
  - Operating systems, IT hardware keep on changing
  - WWW, TCP/IP, Ethernet
ABB Corporate Research Centers
Automation Research Programs

Control & Optimization
Mechatronics & Robotics Automation
Industrial Software Systems

Sensors & Signal Processing
Industrial Communication

Beijing / Shanghai
Bangalore
Krakow
Raleigh
Västerås
Dättwil
Ladenburg

ABB Corporate Research Centers
Industrial Software Systems (ISS) Program

http://www.abb.com/softwareresearch
Agenda

- Context
  - Software and software research at ABB
  - Distributed process control systems
  - Domain-specific design challenges
- Sample projects and initiatives
  - Software sustainability
  - Q-ImPrESS performance modeling
  - Architectural Knowledge Management (AKM) practices
    - Decision rationale – past and present
    - Capturing advice – relevant issues, good justifications
    - Towards a sustainability guide for AKM
- Summary

Software Architecture
Key for Sustainable Performance

**Software architectures for automation systems have to combine state-of-the-art SW technologies to build systems that are open, extendable, and predictably performant – today and tomorrow.**

An object model for the integration of the physical and the logical world

Prediction of real-time behavior
Software Sustainability

- Our definition in this context (ABB ISS program):
  
  "Sustainable software is attractive to customers and cost efficient to maintain and support over its entire lifecycle."

- Is sustainability yet another quality attribute?
  
  We see it as an orthogonal view – lifecycle and usage aspect of any other quality attribute (time dimension)

- Aren’t existing design techniques sufficient to make software sustainable?
  
  - E.g. Object-oriented Analysis and Design Techniques, Component Modeling, e.g. books by J. Cheesman/J. Daniels, C. Larman?
  
  - E.g. Quality attribute scenarios/workshops and technical debt management from the SEI?
  
  - E.g. P. Eeles/P. Cripps book and tutorial by R. Sangwhan?
How Can We Promote Software Sustainability?
Pre-Select and Assemble Proven (Public) Methods

- Iterative and incremental process
- Architecture part of design
- Functional and non-functional concerns emphasized

Software Development Improvement Initiative
Recommended Methods and Techniques

- **Requirements elicitation**
  - Quality attribute scenarios
- **Architectural design**
  - Architectural styles
  - Architectural tactics
- **Architectural documentation**
  - Architectural views
- **Architecture evaluation**
  - ARID: Active review for intermediate design
  - ATAM: Architecture trade-off analysis method
- **Architecture enforcement**
  - Dependency structure matrices
How Can We Promote Software Sustainability?
Pre-Select and Assemble Proven (Public) Methods

ATAM: Architecture Tradeoff Analysis Method

Category: Requirements Elicitation, Architecture Evaluation, Architecture Validation

Level of Effort: Medium
Quality Attributes: Arbitrary
Project Phase: Early or late

Purpose: Risk mitigation by assessing the consequences of architectural decisions by discovering trade-offs between quality requirements and sensitivity points.

Short Description: The ATAM process consists of gathering stakeholders together in two workshops to analyze business drivers and from these drivers extract quality attributes that are used to create scenarios. These scenarios are then used in conjunction with architectural approaches and architectural decisions to create an analysis of trade-offs, sensitivity points, and risks (or non-risks).

Time Involved in Implementing: Phase 1: several weeks of informal preparation, Phase 1: one day workshop, Phase 2: two day workshop 2-3 weeks after Phase 1, Phase 3: one week of informal follow-ups.

Inputs:
- Presentation of a system overview and business drivers
- A software architecture documented in several architectural views to be presented minutes.
- At least informal information about the quality attribute requirements

Outputs:

Excerpt from Sustainability Guide (DARWIN Project)

- **Motivation:** rich toolbox, but little time
  - Many methods and techniques
  - Not enough time for education, experimentation, evaluation
- Provide **guidance** in the form of structured text à la patterns
  - Motivation, short description
  - Tool pointers, literature references
- **Risks**

5.3 Refactoring

<table>
<thead>
<tr>
<th>Application effort:</th>
<th>Low (automated support for a large number of refactorings in modern IDEs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevance for evolution:</td>
<td>Keeping code readable and maintainable improves its understandability, maintainability and evolvability</td>
</tr>
<tr>
<td>Learning effort:</td>
<td>Low (although plenty of refactorings exist, most of them are easy to understand and remember)</td>
</tr>
<tr>
<td>Addressed problem:</td>
<td>Internal quality of a system and system understandability</td>
</tr>
<tr>
<td>General validation:</td>
<td>ABB internal validation: Checklists &amp; further reading: 1.19</td>
</tr>
</tbody>
</table>

© ABB Group
May 10, 2012 | Slide 26

© ABB Group
May 10, 2012 | Slide 27
Motivation
Problems of Software Evolution at ABB

- Continuous evolution of ABB software systems
  - New requirements, technologies, failure reports
  - Software maintenance and evolution are a large cost factor for ABB software development

- Current practice
  - Experience to rationalize design decisions
  - Prototyping for new technologies, performance impacts
  - Unknown change impacts on performance/reliability

Apply model-based prediction methods for systematic decision support to save costs and achieve higher quality?

Quality Impact Predictions for Evolving Service-Oriented Systems (Q-ImPreSS)
Manual Model Creation

Modelling static structure
- Analyzed architectural documentation
- Identified four key use cases
- Abstraction level: process = component

Modelling dynamic structure
- Created testbed, installed system
- Recorded component transitions
- Derived transition probabilities

Validating the model
- Created Q-ImPreSS model in workbench
- Applied Q-ImPreSS consistency checker
- Discussed the model with architects
Q-ImPrESS Model of an ABB Process Control System
Manual Model Creation

Q-ImPrESS Workbench
Performance Prediction
Sample Predictions for Different Design Alternatives

Achieved prediction error below 30 percent
Easy to analyze different evolution scenarios

Data collection consumed more time than expected
Many bottlenecks below the architectural level

Performance Prediction
Results: Measurements vs. Simulation Results

<table>
<thead>
<tr>
<th>Workload</th>
<th>PerfMon Measured</th>
<th>SimuCom Prediction</th>
<th>Error (%)</th>
<th>LONS Prediction</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>17.146</td>
<td>12.467</td>
<td>27.288</td>
<td>12.464</td>
<td>27.305</td>
</tr>
<tr>
<td>90</td>
<td>31.902</td>
<td>32.347</td>
<td>1.395</td>
<td>32.322</td>
<td>1.317</td>
</tr>
<tr>
<td>120</td>
<td>39.016</td>
<td>42.432</td>
<td>8.754</td>
<td>42.329</td>
<td>8.490</td>
</tr>
<tr>
<td>150</td>
<td>51.929</td>
<td>51.943</td>
<td>0.027</td>
<td>51.760</td>
<td>0.326</td>
</tr>
</tbody>
</table>

Pro

- Achieved prediction error below 30 percent
- Easy to analyze different evolution scenarios

Con

- Data collection consumed more time than expected
- Many bottlenecks below the architectural level
Q-Impress – Many Lessons Learned

Results Presented at ICSE 2011 (URI)

- Q-ImPrESS:
  - Provides a structured method and useful tool support
  - Is best used for evolutionary changes, not full redesigns
  - Still needs to demonstrate costs/benefits
- Future work:
  - More robust reverse engineering tools
  - Model transformations from UML to Q-ImPrESS
  - Tools and best practices for data collection

More research and tool development needed

Agenda

- Context
  - Software and software research at ABB
  - Distributed process control systems
  - Domain-specific design challenges
- Sample projects and initiatives
  - Software sustainability
  - Q-ImPrESS performance modeling
- Architectural Knowledge Management (AKM) practices
  - Decision rationale – past and present
  - Capturing advice – relevant issues, good justifications
  - Towards a sustainability guide for AKM
- Summary
Architectural Knowledge Management (AKM) (1/2)

- D. Perry/A. Wolf (1992): *Software Architecture = (Elements, Form, Rationale)*
- P. Kruchten (2004): presentation and workshop paper (QoSA 2006 update)

**Ontology of Architectural Decisions**

More than just lists of UML diagrams, architectural knowledge is embodied in the classes of architectural design decisions and their numerous relationships. This paper presents a taxonomy of such decisions.


  - An architectural (design) decision is “a description of the set of architectural additions, subtractions and modifications to the software architecture, the rationale, and the design rules, design constraints and additional requirements that (partially) realize one or more requirements on a given architecture.”
  - Rationale defined as: “The reasons behind an architectural design decision are the rationale of an architectural design decision. It describes why a change is made to the software architecture.”

Architectural Knowledge Management (AKM) (2/2)

- SHARK workshops and WICSA/QoSA conference tracks since 2006
- Special Issue: *Architectural Decisions and Rationale*, Journal of Systems and Software 82(8), 2009 (editorial and four papers, e.g., one from presenter)
  - Management strategies – explicit vs. implicit
  - Use cases, ontologies (e.g. Griffin core model), links to other design artifacts
  - Tool survey (research prototypes)
  - Case studies (e.g. SOA reference architecture with recurring architectural decisions from IBM)
- ISO/IEC/IEEE 42010 now makes decision capturing mandatory (2011)
An Example of an Architectural Decision
(Modeled in Sparx Systems Enterprise Architect)

Many Metamodels – Few Models

- Many metamodels and templates have been specified
  - IBM Unified Method Framework (since 1998), see SATURN 2010
  - Key decision template suggested by Bredemeyer Consulting
  - Capturing table by J. Tyree/A. Akerman in IEEE Software 22(2), 2005
  - arc42 initiative also suggests a template, e.g. wiki-style
  - TOGAF 9 has the notion of an architectural contract (with rationale)

- But only very few models are publicly available (confidentiality/maintenance?)
  - M. Shaw and D. Garlan (1996) feature a partial design space for user interface architectures in their book (Chapter 5, page 97)
  - 26 recurring WS-* decisions in Perspectives on Web Services (2003)
  - Guidance model for SOA partially published in PhD thesis and tutorials
SOAD (2006-2011): Generic Metamodel

· Existing metamodels and templates refactored and extended for reuse
  - *Before*: documentation – after the fact (past tense)
  - *With SOAD*: design guidance – forward looking (future tense)


http://soadecisions.org/soad.htm

Sample Model Content for SCADA/DCS Historian
Sources: Domain Patterns and Recurring Issues

· Conceptual issues (e.g. patterns):
  - Data point selection (granularity, sampling rate)?
  - Data retention policy (duration, protection)?
  - Database style (flat file/relational/document-oriented)?

· Technology issues (e.g. RFCs):
  - Query language?
  - Remoting protocol?
  - Encryption algorithm?

· Vendor asset issues (e.g. products, open source):
  - OS, MW, HW choices (make or buy)?
  - Implementation providers for selected technologies?
  - Backup and restore system?
Patterns + recurring issues yield guidance models for a domain

- Successfully applied to Service-Oriented Architecture (SOA) Design, cloud computing, strategic outsourcing

- Issue catalog organized by layer/node type, by component/connector

Recurring Issues (1/2)

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Decision Topic</th>
<th>Recurring Issues (Decisions Required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise architecture documentation [SZ92, ZTP03]</td>
<td>IT strategy</td>
<td>Buy vs. build strategy, open source policy</td>
</tr>
<tr>
<td>Governance</td>
<td>Methods, processes, notations, tools, reference architectures, coding guidelines, naming standards, asset ownership</td>
<td></td>
</tr>
<tr>
<td>System context [CCS07]</td>
<td>Projects scope</td>
<td>External interfaces, incoming and outgoing calls (protocols, formats, identifiers), service level agreements, billing</td>
</tr>
<tr>
<td>Other viewpoints [Kru95]</td>
<td>Development process</td>
<td>Configuration management, test cases, build/test/production environment staging</td>
</tr>
<tr>
<td>System context [CCS07]</td>
<td>Physical tiers</td>
<td>Locations, security zones, nodes, load balancing, failover, storage placement</td>
</tr>
<tr>
<td>Data management</td>
<td>Data model reach (enterprise-wide?), synchronization/replication, backup strategy</td>
<td></td>
</tr>
<tr>
<td>Architecture overview diagram [Fox03, CCS067]</td>
<td>Logical layers</td>
<td>Coupling and cohesion principles, functional decomposition (partitioning)</td>
</tr>
<tr>
<td>Physical tiers</td>
<td>Locations, security zones, nodes, load balancing, failover, storage placement</td>
<td></td>
</tr>
<tr>
<td>Data management</td>
<td>Data model reach (enterprise-wide?), synchronization/replication, backup strategy</td>
<td></td>
</tr>
<tr>
<td>Architecture overview diagram [EvE03, Fox03]</td>
<td>Presentation layer</td>
<td>Rich vs. thin client, multi-channel design, client conversations, session management</td>
</tr>
<tr>
<td>Domain layer (process control flow)</td>
<td>How to ensure process and resource integrity, business and system transactionality</td>
<td></td>
</tr>
<tr>
<td>Domain layer (remote interfaces)</td>
<td>Remote contract design (interfaces, protocols, formats, timeout management)</td>
<td></td>
</tr>
<tr>
<td>Domain layer (component-based development)</td>
<td>Interface contract language, parameter validation, Application Programming Interface (API) design, domain model</td>
<td></td>
</tr>
<tr>
<td>Resource (data) access layer</td>
<td>Connection pooling, concurrency (auto commit?), information integration, caching</td>
<td></td>
</tr>
<tr>
<td>Integration</td>
<td>Hub-and-spoke vs. direct, synchrony, message queuing, data formats, registration</td>
<td></td>
</tr>
</tbody>
</table>

Recurring Issues (2/2)

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Decision Topic</th>
<th>Recurring Issues (Decisions Required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical component [ZTP03]</td>
<td>Security</td>
<td>Authentication, authorization, confidentiality, integrity, non-repudiation, tenancy</td>
</tr>
<tr>
<td>Systems management</td>
<td></td>
<td>Fault, configuration, accounting, performance, and security management</td>
</tr>
<tr>
<td>Logical component [ZZG+08]</td>
<td>Lifecycle management</td>
<td>Lookup, creation, static vs. dynamic activation, instance pooling, housekeeping</td>
</tr>
<tr>
<td>Logging</td>
<td>Log source and sink, protocol, format, level of detail (verbosity levels)</td>
<td></td>
</tr>
<tr>
<td>Error handling</td>
<td>Error logging, reporting, propagation, display, analysis, recovery</td>
<td></td>
</tr>
<tr>
<td>Components and connectors [ZTP03, CCS07]</td>
<td>Implementation technology</td>
<td>Technology standard version and profile to use, deployment descriptor settings (QoS)</td>
</tr>
<tr>
<td>Deployment</td>
<td>Collocation, standalone vs. clustered</td>
<td></td>
</tr>
<tr>
<td>Physical node [YRS+09]</td>
<td>Capacity planning</td>
<td>Hardware and software sizing, topologies</td>
</tr>
<tr>
<td>Systems management</td>
<td>Monitoring concept, backup procedures, update management, disaster recovery</td>
<td></td>
</tr>
</tbody>
</table>


How Much Design Rationale is Enough?

- Little information what/how much to capture:
  - Most metamodels and templates ask for a lot of detail (cost/benefit?)
  - G. Fairbanks suggests a lean/minimalistic approach to rationale capturing in his architectural haikus (presented at WICSA 2011) and his book:
    - Requirement <driver-x> is a priority, so we chose design <alt-y>, accepting downside <consequence-z>

(WH)Y?

- My version (the Y-approach):
  - In the context of <use case/user story u>, facing <concern c>, we decided for <option o> to achieve <quality q>
  - These Y-statements yield a bullet list of open/closed (design) issues (link to project management!)
  - Can go to appendix of software architecture document, notes attached to UML model elements, spreadsheet, team space, or wiki
The Y-Approach to Decision Capturing

Skeleton

In the context of <use case/user story u>, … facing <non-functional concern c>, … we decided for <option o> … to achieve <quality q>.

Example

In the context of historian access to archive, … facing data privacy regulations, … we decided to encrypt historian database content … to achieve confidentiality.
In the context of <use case uc and/or component co>, ... facing <non-functional concern c>,

... we decided for <option o1> and neglected <options o2 to on>,

... to achieve <quality q>,

... accepting downside <consequence c>.

To satisfy quality attribute scenario <qas-n> (source, stimulus, environment, artifact)

... we decided for <option o>

... to realize response <r> and achieve <response measure m>.
Valid and Invalid Justifications
Food for Architectural Evaluations/Reviews!

- **Convincing rationale:**
  - Direct link to requirements (the “Y”)
  - Quality attributes in particular, but also functional requirements and constraints
  - Positive experience on previous project
  - Or prototype, experiment, simulation
  - Existing skills, license agreements
  - Other project management concerns

- **Poor justifications:**
  - Market momentum
  - Technology or vendor push
  - Only one alternative known/considered
  - Other killer phrases
  - Keep CVs of team members current

---

### Good and Bad Justifications, Part 1

<table>
<thead>
<tr>
<th>Decision driver type</th>
<th>Valid justification</th>
<th>Counter example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wants and needs of external stakeholders</td>
<td>Alternative A best meets user expectations and functional requirements as documented in user stories, use cases, and business process model.</td>
<td>End users want it, but no evidence for a pressing business need. Technical project team never challenged the need for this feature. Technical design is prescribed in the requirements documents.</td>
</tr>
<tr>
<td>Architecturally significant requirements</td>
<td>Nonfunctional requirement XYZ has higher weight than any other requirement and must be addressed; only alternative A meets it.</td>
<td>Do not have any strong requirements that would favor one of the design options, but alternative B is the market trend. Using it will reflect well on the team.</td>
</tr>
<tr>
<td>Conflicting decision drivers and alternatives</td>
<td>Performed a trade-off analysis, and alternative A scored best. Prototype showed that it’s good enough to solve the given design problem and has acceptable negative consequences.</td>
<td>Only had time to review two design options and did not conduct any hands-on experiments. Alternative B does not seem to perform well, according to information online. Let’s try alternative A.</td>
</tr>
</tbody>
</table>

Good and Bad Justifications, Part 2

<table>
<thead>
<tr>
<th>Decision driver type</th>
<th>Valid justification</th>
<th>Counter example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reuse of an earlier design</td>
<td>Facing the same or very similar NFRs as successfully completed project XYZ. Alternative A worked well there. A reusable asset of high quality is available to the team.</td>
<td>We’ve always done it like that. Everybody seems to go this way these days; there’s a lot of momentum for this technology.</td>
</tr>
<tr>
<td>Prefer do-it-yourself over commercial off-the-shelf (build over buy)</td>
<td>Two cornerstones of our IT strategy are to differentiate ourselves in selected application areas, and remain master of our destiny by avoiding vendor lock-in. None of the evaluated software both meets our functional requirements and fits into our application landscape. We analyzed customization and maintenance efforts and concluded that related cost will be in the same range as custom development.</td>
<td>Price of software package seems high, though we did not investigate total cost of ownership (TCO) in detail. Prefer to build our own middleware so we can use our existing application development resources.</td>
</tr>
<tr>
<td>Anticipation of future needs</td>
<td>Change case XYZ describes a feature we don’t need in the first release but is in plan for next release. Predict that concurrent requests will be x per second shortly after global rollout of the solution, planned for Q1/2009.</td>
<td>Have to be ready for any future change in technology standards and in data models. All quality attributes matter, and quality attribute XYZ is always the most important for any software-intensive system.</td>
</tr>
</tbody>
</table>


Wanted: Integrated Decision/Design Tool Chain To Create, Read, Update, Delete Y-Statements

- Tool builders should justify capture their design decisions (like any architect)… and share them with their collaborators!
From Sustainable Software to Sustainable Architectural Knowledge

- Wanted: Timeless advice, but still concrete and grounded
  - Pattern harvesting books vs. abstract principles in design-by-committee standard/specification
  - Easy to maintain, easy to follow

- The sustainability guide from DARWIN project is very much in line with SOAD vision of capturing guidance models compiling recurring issues and related advice:
  - Structured text, blending facts with opinions
  - Complementary to component-and-connector views

⇒ Stay tuned!

Agenda

- Context
  - Software and software research at ABB
  - Distributed process control systems
  - Domain-specific design challenges
- Sample projects and initiatives
  - Software sustainability
  - Q-ImPrESS performance modeling
- Architectural Knowledge Management (AKM) practices
  - Decision rationale – past and present
  - Capturing advice – relevant issues, good justifications
  - Towards a sustainability guide for AKM
- Summary
Summary and Conclusions

- Software and software architecture play a key role at ABB
  - Projects, programs, initiatives are in place
  - Key themes: modeling, reuse, rationale, sustainability
- Architecture design is driven both by functional and by non-functional requirements – and constraints of both kinds
  - Design techniques and modeling tools should combine these
  - Hard to see the forest for the trees – guidance required
- Answers to why questions matter – and often are more sustainable than most component-and-connector diagrams (reuse of know how!)
  - Explicit knowledge management does not imply big design upfront (evolutionary architectures and designs should be justified!)
    - See recently released IEEE 42010 standard
  - Try an architectural haiku and/or the “Y”-approach to capture the essence of a decision (outcome and rationale)

IEEE Software Update

- Digital edition (PDF, enhanced PDF)
  - A free sample is here (IEEE Software, Jan/Feb 2011).
- Upcoming special issue submission opportunities:
  - Bridging Software Communities through Social Networking, Due June 1.
    http://www.computer.org/portal/web/computingnow/swcfp1
  - The Twin Peaks of Requirements and Architecture, Due August 1.
    http://www.computer.org/portal/web/computingnow/swcfp2
Recommended Reading

  - Motivation for decision capturing, basic definitions
  - Rationale – what to capture, how to capture
  - From documentation to design guidance, SOA examples
  - Domain-specific quality attributes and decisions