

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

# Making Architectural Knowledge Sustainable – The Y-Approach

## Industrial Practice Report and Outlook

Power and productivity  
for a better world™ **ABB**

## 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

© ABB Group  
May 10, 2012 | Slide 3

**ABB**

## 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

© ABB Group  
May 10, 2012 | Slide 4



## A Global Leader in Power and Automation Technologies

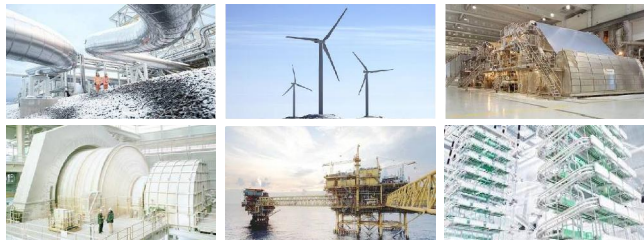


- 133,600 employees in about 100 countries
- \$37.990 million in revenue (2011)
- Formed in 1988 merger of Swiss and Swedish engineering companies
- Predecessors founded in 1883 and 1891
- Publicly owned company with head office in Switzerland

© ABB Group  
May 10, 2012 | Slide 5



## 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.

© ABB Group  
May 10, 2012 | Slide 6



## 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

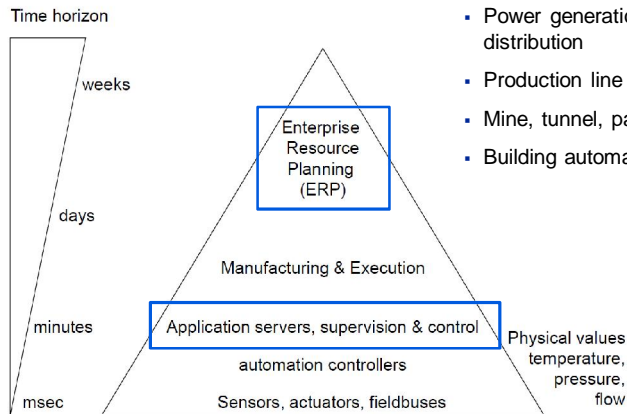
... and all have highest requirements for

- Real-time performance
- Reliability
- Long lifetime
- Remote connectivity

© ABB Group  
May 10, 2012 | Slide 8



## Collaborative Process Automation Systems Automation Pyramid



### 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

© ABB Group  
May 10, 2012 | Slide 9



## Collaborative Process Automation Systems Automation Levels



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

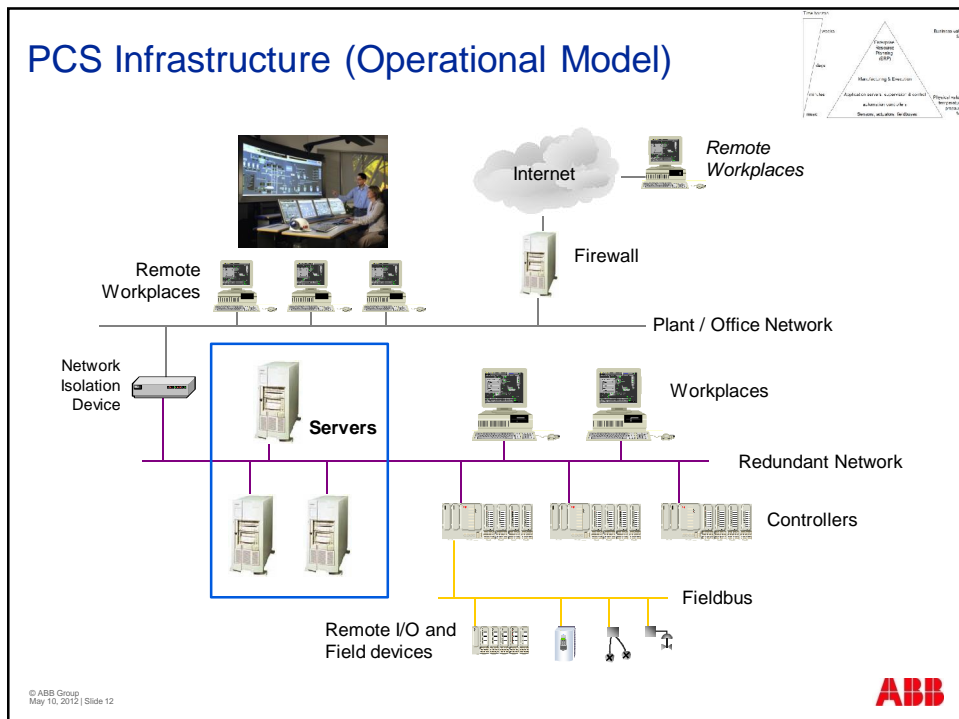
© ABB Group  
May 10, 2012 | Slide 10



## Industrial Automation: Process Control Systems (PCS)

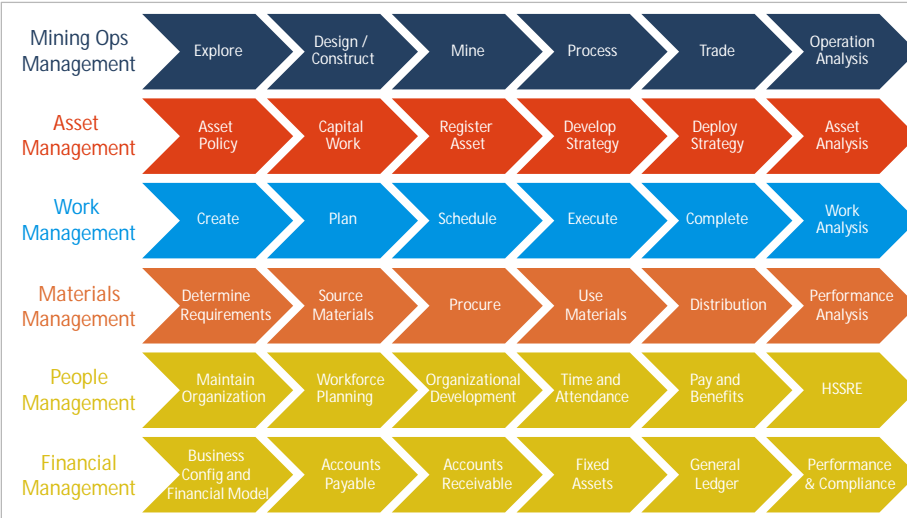


## PCS Infrastructure (Operational Model)



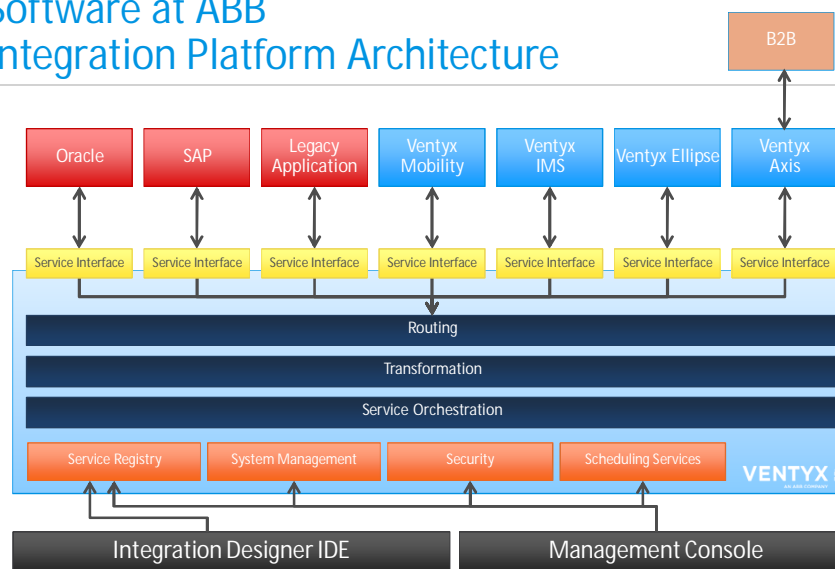
## Software at ABB

### Solution Approach Using Enterprise Processes



**VENTYX**   
AN ABB COMPANY

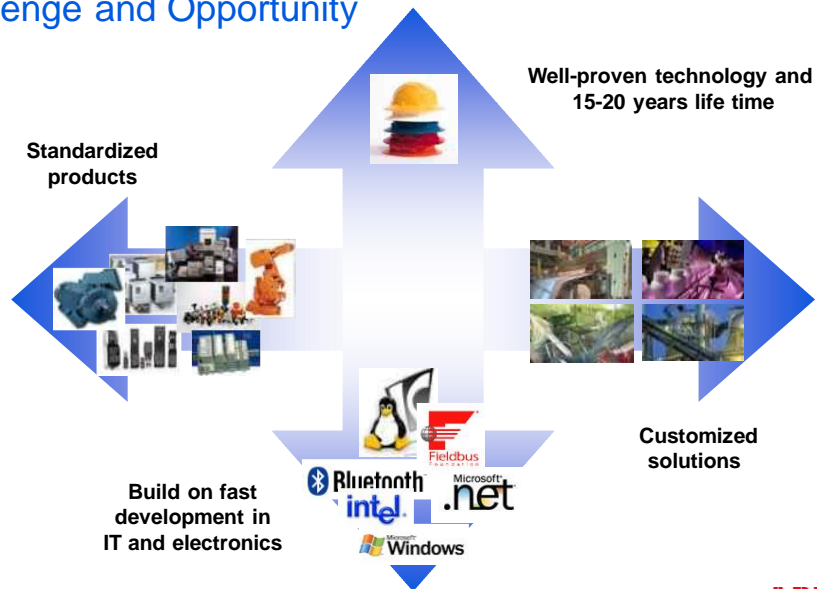
## Software at ABB Integration Platform Architecture



**VENTYX**   
AN ABB COMPANY

| ©2012 Ventyx, An ABB Company | 15

## Software at ABB Challenge and Opportunity



© ABB Group  
May 10, 2012 | Slide 16

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

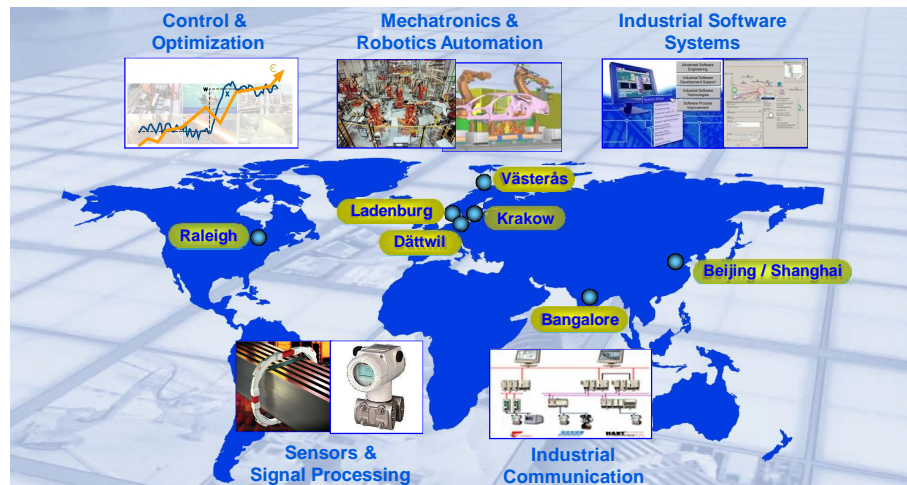
- **Safety and security**
  - E.g. Security guidelines from North American Electric Reliability Corporation (NERC)
  - E.g. Stuxnet threat (for entire industry)
- **Remote locations**
  - Unmanned plants
  - Extreme environmental conditions
- **Diversity and lifetime of installed base**
  - 1000s of 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 Group  
May 10, 2012 | Slide 17

ABB



## ABB Corporate Research Centers Automation Research Programs



© ABB Group  
May 10, 2012 | Slide 18



## Industrial Software Systems (ISS) Program

Power and productivity for a better world™

ABB in der Schweiz

Home Über uns Produkte und Dienstleistungen Medien Job und Karriere ABB Group

Sitemap Anmeldung

### Technology

- Technology
- + Global lab power
- Global lab automation
- Control and optimization
- Industrial communication
- Industrial Software Systems**
- Mechatronics
- Sensors and signal processing
- + Corporate research centers
- + Technology Ventures
- + Publications
- + Select technologies
- Cyber security
- DC power

### Industrial Software Systems Program

The ABB Corporate Research program Industrial Software Systems (ISS) supports ABB product development units with world-class expertise in information/software technology, software engineering, and computer science in making effective use of novel information technologies and in applying state-of-the-art software engineering methods in order to build products that are attractive to ABB customers.

The Industrial Software Systems program fulfills its mission of ensuring ABB competitiveness and technology leadership through projects in four research areas:

- **Software engineering** creates methods and tools to increase quality and reduce cost of ABB internal software development.
- **Software architecture** researches new ways of designing, building, and maintaining complex software systems over multi-decade lifecycles.
- **IT systems architecture** explores the usage of secure and highly scalable HW/SW systems composed of ABB as well as COTS/FOSS subsystems for information processing and analysis.
- **Human-computer interaction** investigates user experience and situational awareness in the context of industrial systems.

The ISS program team in five locations worldwide (Västerås/SE, Raleigh/US, Bangalore/IN, Ladenburg/DE, Baden/CH), works with software development units across all ABB divisions and geographic areas. Interdisciplinary projects are executed in collaboration with other ABB Corporate Research programs as well as partners in universities.

Last edited 2012-01-10

Suche

+ Seite bewerten

+ Diese Seite empfehlen

### Kontakt

→ Software research contact

### Links

- ABB Corporate Research Center locations
- ABB Software Research Grants
- Software engineering research
- Security research
- ISS publications
- Open positions

<http://www.abb.com/softwareresearch>

© ABB Group  
May 10, 2012 | Slide 19





## Agenda

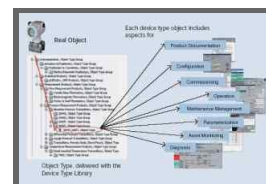
- Context
  - Software and software research at ABB
  - Distributed process control systems
  - Domain-specific design challenges
- **Sample projects and initiatives**
  - Software sustainability
  - Q-ImPRESS performance modeling
- Architectural Knowledge Management (AKM) practices
  - Decision rationale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

© ABB Group  
May 10, 2012 | Slide 20

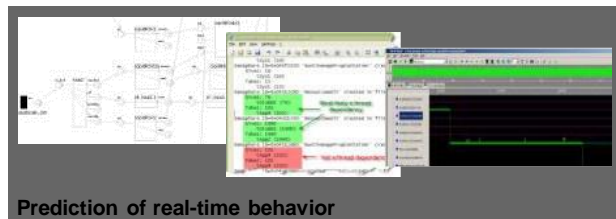


## Software Architecture Key for Sustainable Performance

*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.*

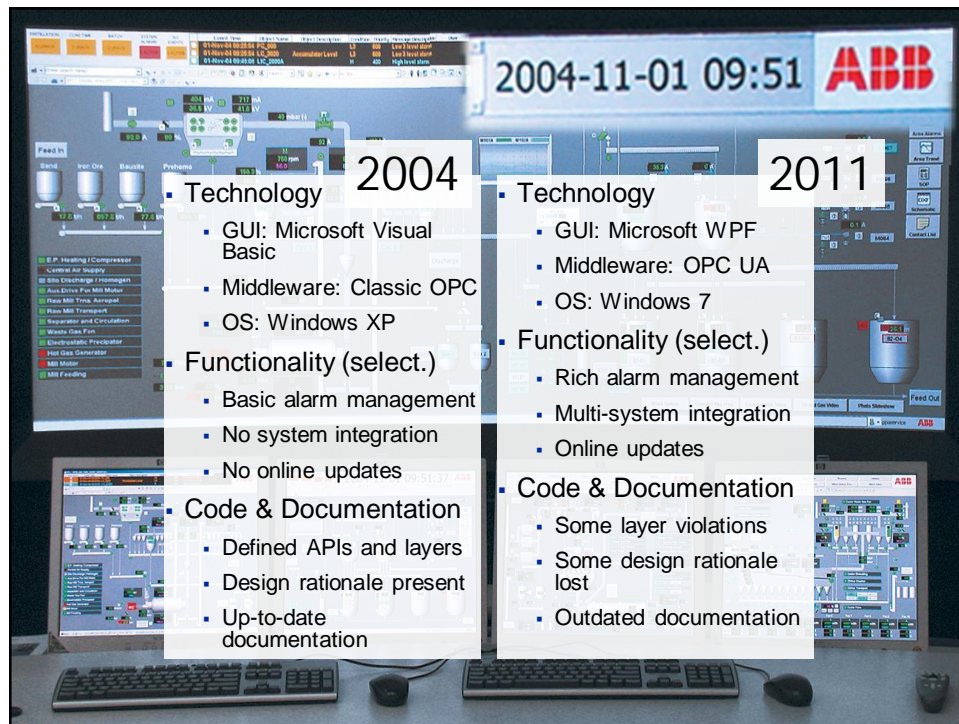


**An object model for the integration of the physical and the logical world**



© ABB Group  
May 10, 2012 | Slide 21

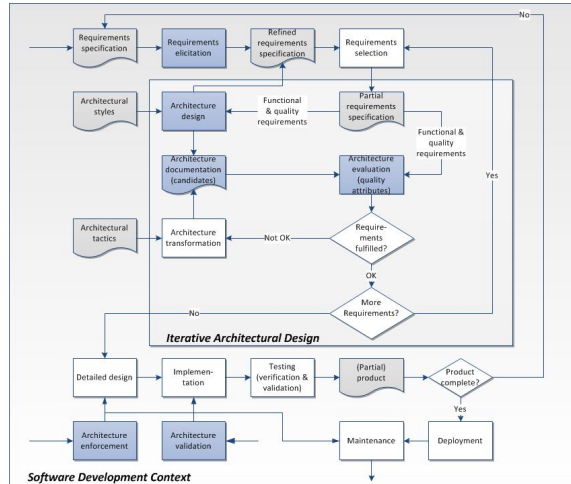




## Software Sustainability

- 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

© ABB Group  
May 10, 2012 | Slide 24



## 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



© ABB Group  
May 10, 2012 | Slide 25



# 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 0: 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 in minutes.
- At least informal information about the quality attribute requirements



### Outputs:

© ABB Group  
May 10, 2012 | Slide 26



## 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 a la patterns
  - Motivation, short description
  - Tool pointers, literature references
  - Risks

|  |           |
|--|-----------|
| <b>3 REQUIREMENTS</b>  | <b>9</b>  |
| 3.1 REQUIREMENTS MANAGEMENT  | 9         |
| 3.2 SUSTAINABLE REQUIREMENTS TRACING                                 | 10        |
| 3.3 REQUIREMENTS RISKS   | 11        |
| 3.4 REQUIREMENTS CHECKLIST   | 11        |
| <b>4 ARCHITECTURE</b>  | <b>12</b> |
| 4.1 PATTERNS / REFERENCE ARCHITECTURES / TACTICS / STYLES            | 12        |
| 4.2 ARCHITECTURE-LEVEL MODIFIABILITY ANALYSIS, ALMA                  | 13        |
| 4.3 SOFTWARE PRODUCT-LINES   | 14        |
| 4.4 ARCHITECTURE-CENTRIC MODEL-DRIVEN SOFTWARE DEVELOPMENT, AC-MOSD  | 15        |
| 4.5 ARCHITECTURE RISKS   | 16        |
| 4.6 ARCHITECTURE CHECKLIST   | 16        |
| <b>5 DESIGN</b>  | <b>17</b> |
| 5.1 CO-METRICS, ISIS   | 17        |
| 5.2 BAD SMELLS, ANTPATTERNS  | 18        |
| 5.3 REFACTORING  | 19        |
| 5.4 DESIGN RISKS   | 20        |
| 5.5 DESIGN CHECKLIST   | 20        |
| <b>6 IMPLEMENTATION</b>  | <b>21</b> |
| 6.1 CODE DOCUMENTATION, NAMING CONVENTIONS                           | 21        |
| 6.2 CLEAN CODE PHILOSOPHY  | 22        |
| 6.3 AUTOMATED IMPLEMENTATION (XTUML)                                 | 23        |
| 6.4 IMPLEMENTATION RISKS   | 24        |
| 6.5 IMPLEMENTATION CHECKLIST   | 24        |
| <b>7 VERIFICATION &amp; VALIDATION, TESTING</b>                      | <b>25</b> |
| 7.1 UNIT TESTING, CONTINUOUS INTEGRATION TESTING, REGRESSION TESTING | 25        |
| 7.2 SUSTAINABLE TEST BIDS  | 26        |
| 7.3 VERIFICATION & VALIDATION, TESTING RISKS                         | 27        |
| 7.4 VERIFICATION & VALIDATION, TESTING CHECKLIST                     | 27        |
| <b>8 MAINTENANCE</b>   | <b>28</b> |

### 5.3 Refactoring

|                            |   |                                 |   |
|----------------------------|---|---------------------------------|---|
| <b>Application effort:</b> | Low (automated support for a large number of refactorings in modern IDEs)                     | <b>Relevance for evolution:</b> | Keeping code readable and modifiable improves its understandability, maintainability and evolvability |
| <b>Learning effort:</b>    | Low (although plenty of refactorings exist, most of them are easy to understand and remember) | <b>Addressed problem:</b>       | Internal quality of a system and system understandability   |
| <b>General validation:</b> | +   | <b>ABB internal validation:</b> | Checklists & further reading: 11.9  |

© 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?

© ABB Group  
May 10, 2012 | Slide 28



## 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

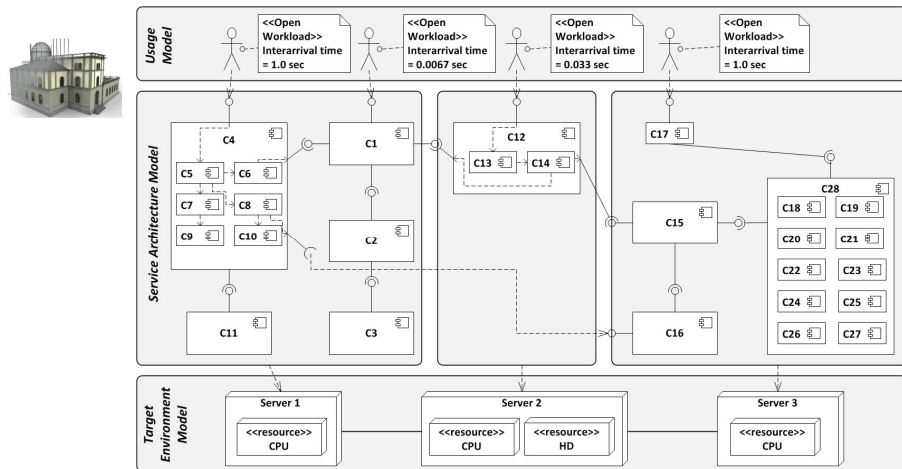
#### Validating the model

- Created Q-ImPreSS model in workbench
- Applied Q-ImPreSS consistency checker
- Discussed the model with architects

© ABB Group  
May 10, 2012 | Slide 29



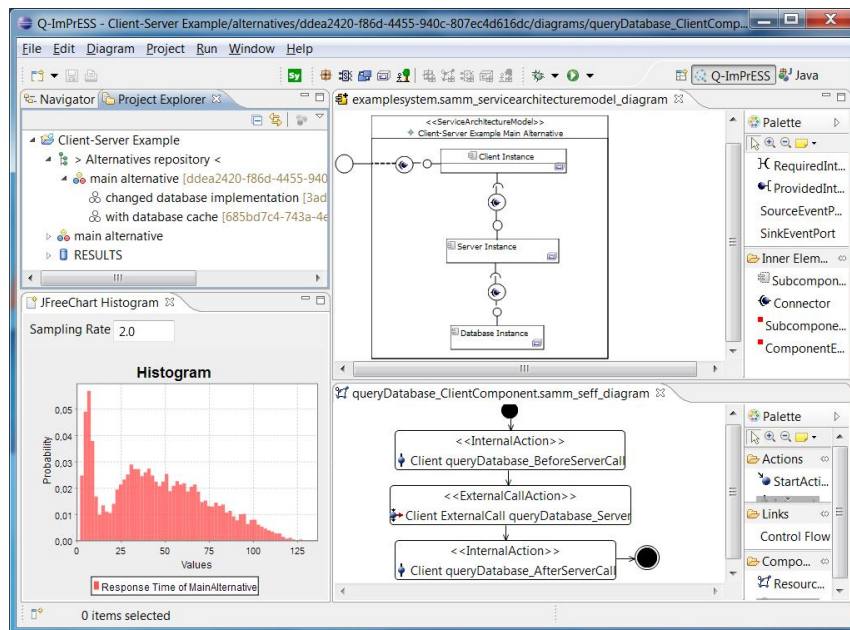
## Q-ImPRESS Model of an ABB Process Control System Manual Model Creation



© ABB Group  
May 10, 2012 | Slide 30

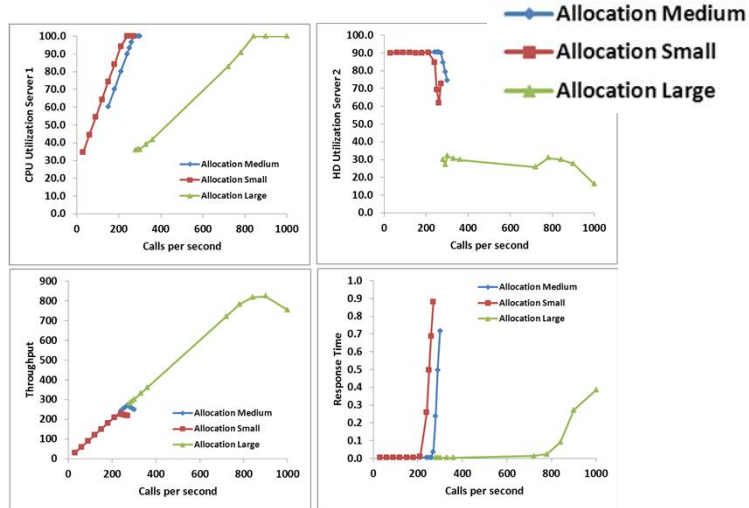


## Q-ImPRESS Workbench





## Performance Prediction Sample Predictions for Different Design Alternatives



© ABB Group  
May 10, 2012 | Slide 32



## Performance Prediction Results: Measurements vs. Simulation Results



| Workload | PerfMon Measured | SimuCom Prediction | Error (%) | LQNS Prediction | Error (%) |
|----------|------------------|--------------------|-----------|-----------------|-----------|
| 30       | 17.146           | 12.467             | 27.288    | 12.464          | 27.305    |
| 60       | 26.681           | 22.366             | 16.174    | 22.343          | 16.260    |
| 90       | 31.902           | 32.347             | 1.395     | 32.322          | 1.317     |
| 120      | 39.016           | 42.432             | 8.754     | 42.329          | 8.490     |
| 150      | 51.929           | 51.943             | 0.027     | 51.760          | 0.326     |

### Pro

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

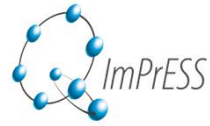
### Con

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

© ABB Group  
May 10, 2012 | Slide 33



## 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

[http://www.infoq.com/news/2011/04/palladio\\_tool](http://www.infoq.com/news/2011/04/palladio_tool)

© ABB Group  
May 10, 2012 | Slide 34



## 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

© ABB Group  
May 10, 2012 | Slide 35



## 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

[Endnotes][Bibtex]

More than just lots of UML diagrams, architectural knowledge is embodied in the dozen of architectural design decisions, and their numerous relationships. This paper presents a taxonomy of such decisions. Philippe Fruchten, "An Ontology of Architectural Design Decisions," in: Jan Bosch (ed.), *Proc. of the 2nd Workshop on Software Variability Management*, Groningen, NL, Dec. 3-4, 2004

[Read the Article >](#)

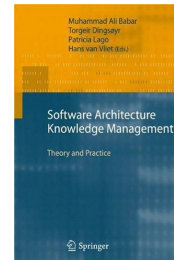
- A. Jansen, J. Bosch: [Software Architecture as a Set of Architectural Design Decisions](#) (2005)
  - An architectural (design) decision is "a description of the set of architectural additions, subtractions and modifications to the software architecture, the rationale, and the design rules, design 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."

© ABB Group  
May 10, 2012 | Slide 36



## Architectural Knowledge Management (AKM) (2/2)

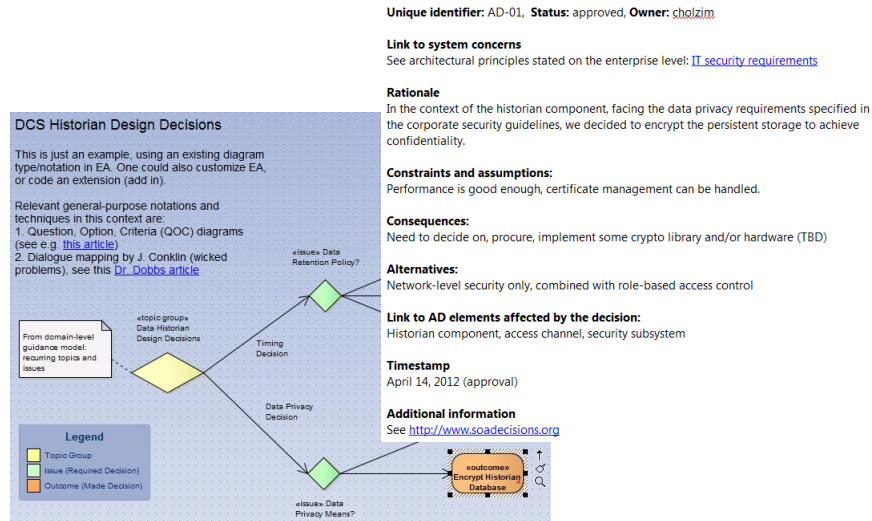
- SHARK workshops and WICSA/QoSA conference tracks since 2006
  - *Architectural Knowledge = Architecture Design + Architectural Decisions* (P. Kruchten, P. Lago, H. van Vliet, QoSA 2006)
- Special Issue: [Architectural Decisions and Rationale](#), Journal of Systems and Software 82(8), 2009 (editorial and four papers, e.g., [one from presenter](#))
- Book: [Software Architecture Knowledge Management – Theory and Practice](#), Springer (2009)
  - 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)



© ABB Group  
May 10, 2012 | Slide 37



## An Example of an Architectural Decision (Modeled in Sparx Systems Enterprise Architect)



© ABB Group  
May 10, 2012 | Slide 38



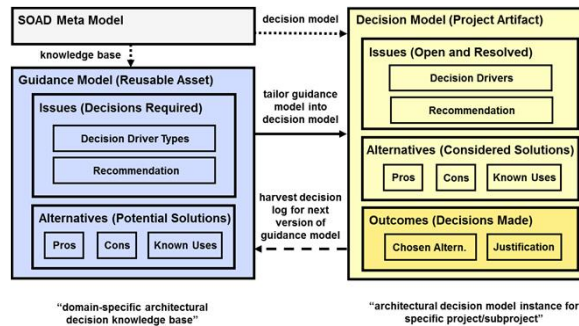
## Many Metamodels – Few Models

- 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)
  - Informal coverage in [integration patterns](#) book and [SWEBOK](#) (2004)
  - Guidance model for SOA partially published in [PhD thesis](#) and [tutorials](#)

© ABB Group  
May 10, 2012 | Slide 39



## SOAD (2006-2011): Generic Metamodel



Source: O. Zimmermann, [Architectural Decisions as Reusable Design Assets](#), IEEE Software, vol. 28, no. 1, pp. 64-69, Jan./Feb. 2011.

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

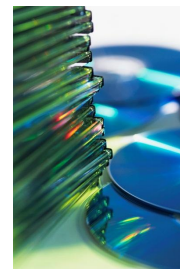
<http://soadecisions.org/soad.htm>

© ABB Group  
May 10, 2012 | Slide 40



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

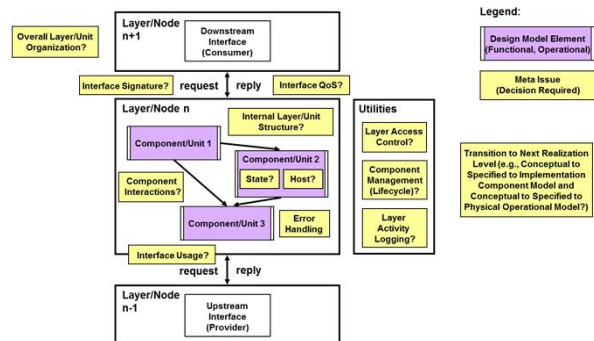
- Conceptual issues (e.g. patterns):**
  - Data point selection (granularity, sampling rate)?
  - Data retention policy (duration, protection)?
  - Database style (flat file/relational/document-oriented)?
- Technology issues (e.g. RFCs):**
  - Query language?
  - Remoting protocol?
  - Encryption algorithm?
- Vendor asset issues (e.g. products, open source):**
  - OS, MW, HW choices (make or buy)?
  - Implementation providers for selected technologies?
  - Backup and restore system?



© ABB Group  
May 10, 2012 | Slide 41



## SOAD (2006-2011): Recurring Issues



Source: O. Zimmermann, [Architectural Decisions as Reusable Design Assets](#). IEEE Software, vol. 28, no. 1, pp. 64-69, Jan./Feb. 2011.

- 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

© ABB Group  
May 10, 2012 | Slide 42



## Recurring Issues (1/2)

| Artifact  | Decision Topic                             | Recurring Issues (Decisions Required)   |
|---|--|---|
| Enterprise architecture documentation [SZ92, ZTP03] | IT strategy                                | Buy vs. build strategy, open source policy  |
|   | Governance                                 | Methods (processes, notations), tools, reference architectures, coding guidelines, naming standards, asset ownership  |
| System context [CCS07]                              | Project scope                              | External interfaces, incoming and outgoing calls (protocols, formats, identifiers), service level agreements, billing |
| Other viewpoints [Kru95]                            | Development process                        | Configuration management, test cases, build/test/production environment staging                                       |
|   | Physical tiers                             | Locations, security zones, nodes, load balancing, failover, storage placement   |
|   | Data management                            | Data model reach (enterprise-wide?), synchronization/replication, backup strategy                                     |
| Architecture overview diagram [Fow03, CCS07]        | Logical layers                             | Coupling and cohesion principles, functional decomposition (partitioning)   |
|   | Physical tiers                             | Locations, security zones, nodes, load balancing, failover, storage placement   |
|   | Data management                            | Data model reach (enterprise-wide?), synchronization/replication, backup strategy                                     |
| Architecture overview diagram [Eva03, Fow03]        | Presentation layer                         | Rich vs. thin client, multi-channel design, client conversations, session management                                  |
|   | Domain layer (process control flow)        | How to ensure process and resource integrity, business and system transactionality                                    |
|   | Domain layer (remote interfaces)           | Remote contract design (interfaces, protocols, formats, timeout management)   |
|   | Domain layer (component-based development) | Interface contract language, parameter validation, Application Programming Interface (API) design, domain model       |
|   | Resource (data) access layer               | Connection pooling, concurrency (auto commit?), information integration, caching                                      |
|   | Integration                                | Hub-and-spoke vs. direct, synchrony, message queuing, data formats, registration                                      |

Source: O. Zimmermann, [An architectural decision modeling framework for service oriented architecture design](#). PhD thesis, Stuttgart University, 2009.

© ABB Group  
May 10, 2012 | Slide 43





## Recurring Issues (2/2)

| Artifact                                 | Decision Topic            | Recurring Issues (Decisions Required)  |
|--|---------------------------|--|
| Logical component [ZTP03]                | Security                  | Authentication, authorization, confidentiality, integrity, non-repudiation, tenancy  |
|  | Systems management        | Fault, configuration, accounting, performance, and security management               |
| Logical component [ZZG+08]               | Lifecycle management      | Lookup, creation, static vs. dynamic activation, instance pooling, housekeeping      |
|  | Logging                   | Log source and sink, protocol, format, level of detail (verbosity levels)            |
|  | Error handling            | Error logging, reporting, propagation, display, analysis, recovery                   |
| Components and connectors [ZTP03, CCS07] | Implementation technology | Technology standard version and profile to use, deployment descriptor settings (QoS) |
|  | Deployment                | Collocation, standalone vs. clustered  |
| Physical node [YRS+99]                   | Capacity planning         | Hardware and software sizing, topologies   |
|  | Systems management        | Monitoring concept, backup procedures, update management, disaster recovery          |

Source: O. Zimmermann, [An architectural decision modeling framework for service oriented architecture design](#), PhD thesis, Stuttgart University, 2009.

© ABB Group  
May 10, 2012 | Slide 44



## 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>*

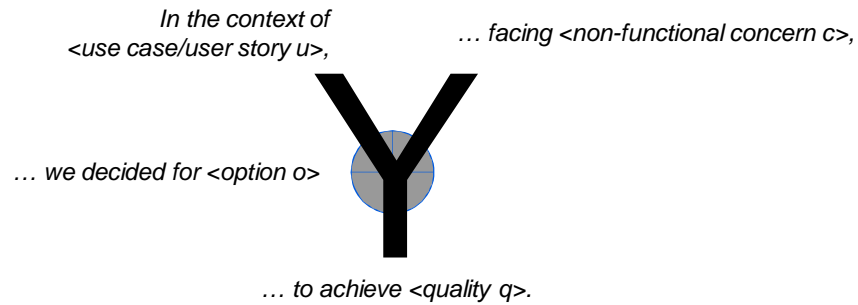
**(WH)Y?**

- 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

© ABB Group  
May 10, 2012 | Slide 45



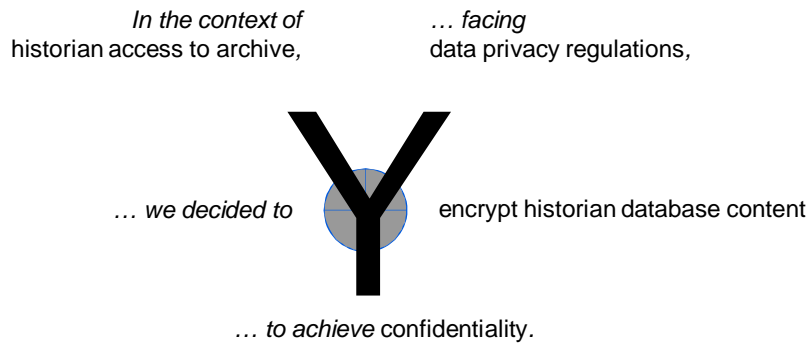
## The Y-Approach to Decision Capturing Skeleton



© ABB Group  
May 10, 2012 | Slide 46



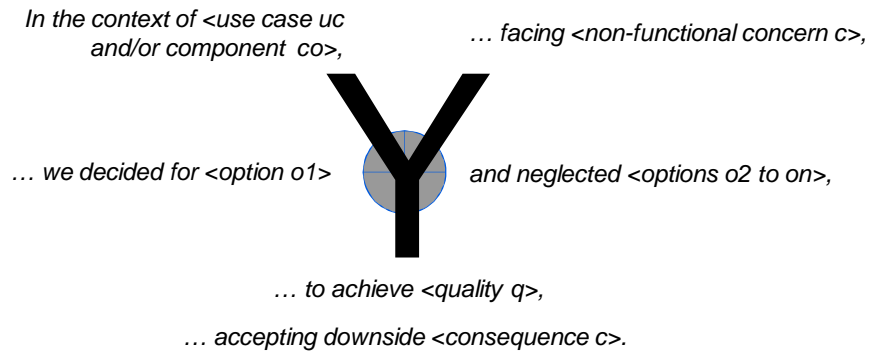
## The Y-Approach to Decision Capturing Example



© ABB Group  
May 10, 2012 | Slide 47



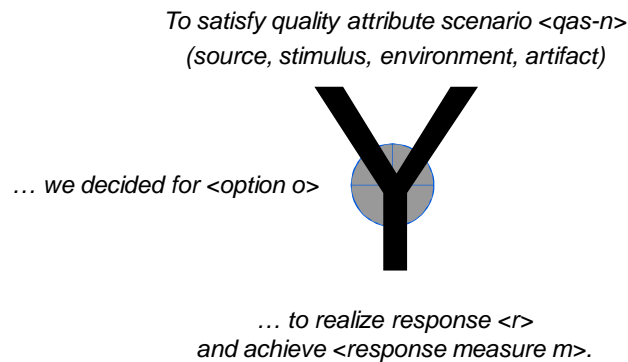
## The Y-Approach to Decision Capturing Extended Skeleton



© ABB Group  
May 10, 2012 | Slide 48



## The Y-Approach to Decision Capturing Quality Attribute Scenario (QAS) Variant of Skeleton



© ABB Group  
May 10, 2012 | Slide 49



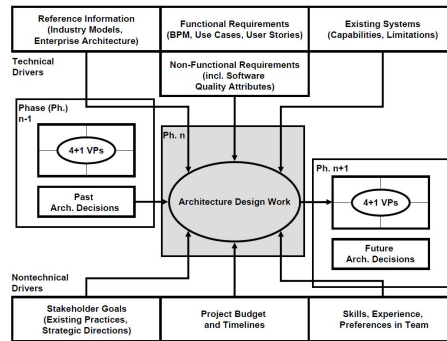
## Valid and Invalid Justifications Food for Architectural Evaluations/Reviews!

### • 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



Source: O. Zimmermann, [An architectural decision modeling framework for service oriented architecture design](#), PhD thesis, Stuttgart University, 2009.

© ABB Group  
May 10, 2012 | Slide 50



## Good and Bad Justifications, Part 1

| Decision driver type                                 | Valid justification   | Counter example   |
|--|---|---|
| <b>Wants and needs of external stakeholders</b>      | Alternative A best meets user expectations and functional requirements as documented in user stories, use cases, and business process model.                                      | 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. |
| <b>Architecturally significant requirements</b>      | Nonfunctional requirement XYZ has higher weight than any other requirement and must be addressed; only alternative A meets it.  | 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.                                    |
| <b>Conflicting decision drivers and alternatives</b> | 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. | 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.   |

Source: Zimmermann O., Schuster N., Eeles P., [Modeling and Sharing Architectural Decisions, Part 1: Concepts](#), IBM developerWorks, 2008

© ABB Group  
May 10, 2012 | Slide 51



## Good and Bad Justifications, Part 2

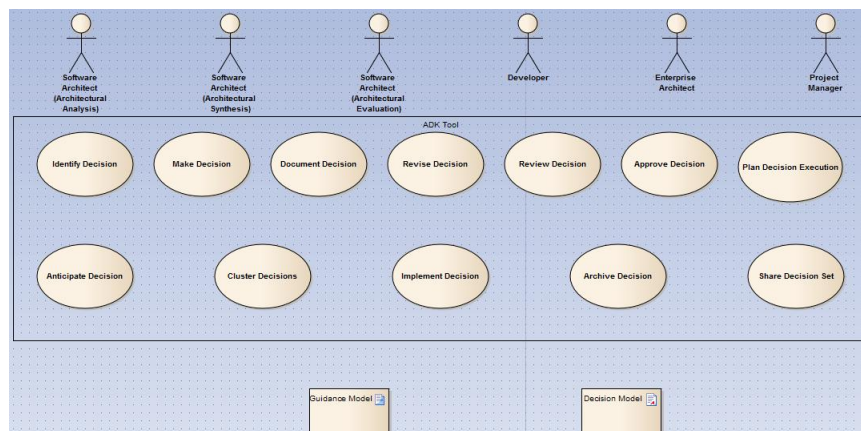
| Decision driver type  | Valid justification  | Counter example  |
|---|--|--|
| <b>Reuse of an earlier design</b>   | Facing the same or very similar NFRs as successfully completed project XYZ. Alternative A worked well there. A reusable asset of high quality is available to the team.  | We've always done it like that.<br>Everybody seems to go this way these days; there's a lot of momentum for this technology.   |
| <b>Prefer do-it-yourself over commercial off-the-shelf (build over buy)</b> | 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 off-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. | Price of software package seems high, though we did not investigate total cost of ownership (TCO) in detail.<br>Prefer to build our own middleware so we can use our existing application development resources. |
| <b>Anticipation of future needs</b>   | Change case XYZ describes a feature we don't need in the first release but is in plan for next release.<br>Predict that concurrent requests will be x per second shortly after global rollout of the solution, planned for Q1/2009.  | Have to be ready for any future change in technology standards and in data models.<br>All quality attributes matter, and quality attribute XYZ is always the most important for any software-intensive system.   |

Source: Zimmermann O., Schuster N., Eeles P., [Modeling and Sharing Architectural Decisions, Part 1: Concepts](#), IBM developerWorks, 2008

© ABB Group  
May 10, 2012 | Slide 52



## 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!

© ABB Group  
May 10, 2012 | Slide 53



## 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!

© ABB Group  
May 10, 2012 | Slide 54



## 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**

© ABB Group  
May 10, 2012 | Slide 55





## 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)*

© ABB Group  
May 10, 2012 | Slide 56



## 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>

© ABB Group  
May 10, 2012 | Slide 57



## Recommended Reading

- Jansen/Bosch, *Software Architecture as a Set of Architectural Design Decisions*, WICSA 2005. 5th Working IEEE/IFIP Conference on Software Architecture
  - Motivation for decision capturing, basic definitions
- ISO/IEC/IEEE 42010:2011, *Systems and software engineering—Architecture description*.
  - Rationale – what to capture, how to capture
- Zimmermann O., *Architectural Decisions as Reusable Design Assets*. IEEE Software, vol. 28, no. 1, pp. 64-69, Jan./Feb. 2011
  - From documentation to design guidance, SOA examples
- Hollender, *Collaborative Process Automation Systems*, ISA 2010
  - Domain-specific quality attributes and decisions

© ABB Group  
May 10, 2012 | Slide 58



Power and productivity  
for a better world™

