Metrics for Architectural Synthesis and Evaluation – Requirements and Compilation by Viewpoint

An Industrial Experience Report

Olaf Zimmermann
The Open Group Distinguished (Chief/Lead) IT Architect, Institute for Software
University of Applied Sciences of Eastern Switzerland (HSR FHO)
Rapperswil, Switzerland
ozimmerm@hsr.ch

***Abstract*—During architectural analysis and synthesis, architectural metrics are established tacitly or explicitly. In architectural evaluation, these metrics are then consulted to assess whether architectures are fit for purpose and in line with recommended practices and published architectural knowledge. This experience report presents a personal retrospective of the author's use of architectural metrics during 20 years in IT architect roles in professional services as well as research and development. This reflection drives the identification of use cases, critical success factors and elements of risk for architectural metrics management. An initial catalog of architectural metrics is compiled next, which is organized by viewpoints and domains. The report concludes with a discussion of practical impact of architectural metrics and potential research topics in this area.**

***Index Terms*— architectural metrics, architectural metrics management, architectural reviews, enterprise information systems, integration, patterns, viewpoints.**

I. Introduction

Software architects in product development and professional services perform activities in three categories: architectural analysis, architectural synthesis and architectural evaluation [10]. Architectural Metrics (AMs) pertain to artifacts created during architectural analysis and synthesis; in architectural evaluation, these metrics are then consulted to assess whether architectures are fit for purpose and in line with recommended community practices and the architectural knowledge that represents the state of the art in a domain (this evaluation activity is complementary to assessing whether an architecture meets the architecturally significant requirements that came out of architectural analysis).

In this experience report, I reflect on 20 years in IT architect roles in professional services as well as research and development to identify the AMs that I repeatedly applied during this time. Some of these AMs are rather generic, while others depend on the business context and the technical domain the system under construction is concerned with. The reflection yields (a) an identification of use cases, critical success factors and elements of risk regarding AM Management (AMM) and (b) a catalog of AMs organized by viewpoints and domains/styles. It also highlights research areas and challenges.

II. Sources of Metrics Knowledge and Example

Let me first share selected project context information in this section and then present an example to establish the roots of the architectural metrics introduced in subsequent sections.

A. *Context: Project Experience and IT Architect Roles*

The first time I acted as Information Technology (IT) Architect was in 1995, when I led the definition of a client-specific solution architecture for a telecommunications network management system together with a senior architect. Since then, I served in various IT architect roles in professional services and software product development from Subject Matter Expert (SME) for certain technologies and platforms to subsystem architect (application architect, integration architect) to lead architect and technical project manager. The developed software included middleware, code generators, and, primarily, client/server Enterprise Information Systems (EIS). Some of these EIS leveraged SOA concepts and Web services technologies [23][24]. In recent years, my industrial research and development projects focused on design and decision making tools [20][21][22]; via action research and consulting assignments, I contributed to additional EIS architectures.

Table 1 identifies the lower and upper boundaries of typical project context information in the eight dimensions from [14].

TABLE 1. SAMPLE PROJECT CONTEXT DIMENSIONS

Dimension	Research Prototypes (Tools)	Enterprise-Scale Information Systems
System size	1000 Source Lines of Code (SLOC), standalone program	1 Mio. SLOC (and up), system of systems with 20 and more backend interfaces
System criticality	None, thrown away after experiment	Medium to high (no hard real time requirements, but business critical applications)
System age	New	10 years and more (up to 50-70 years for certain legacy applications and interfaces)
Team distribution	None	Asia/Pacific, Europe, USA
Rate of change	(Bi-)weekly software releases; startup mentality	e.g. government client: six months for one iteration over use case model; three years to go live with Version 1 of an e-business solution
Pre-existence of a stable architecture	None	Yes, e.g. IBM mainframe, Java Enterprise Edition (JEE); in-house frameworks at IT-savvy clients
Governance	Self-organized, agile practices	e.g. enterprise architecture frameworks, Stage Gate Model [19]
Business model	n/a	Fee-based (software solutions), licensing (software products)

The table entries indicate significant project diversity; any AMM approach and AM catalog must be able to deal with such diversity (e.g., via profiling and other tailoring means). To elaborate on the business model dimension, Table 2 differentiates between product and solution architects:

TABLE 2. TWO TYPES OF ARCHITECT ROLES AND THEIR DIFFERENCES

Aspect	**Product Architects (working for vendor)**	**Solution Architects (in professional services)**
Business model	Past: Software license sales plus maintenance fees (threat: open source)	Service fees (time and material) or fixed price contracts (software lot/craft)
	Future: SaaS fees	
Functional requirements	From product manager (top down), internal innovation (bottom up)	From users, from sponsors (top down), use cases or user stories
Quality Attributes (NFRs)	Many profiles, generic or flexible	Single set (unless client is a software vendor and a product architect is served)
Design method	Many from waterfall to agile (today)	IBM Unified Method Framework, Unified Process, agile practices
Deployment	At customer, in cloud	Data center of client, IT outsourcer, cloud
Key Performance Indicators	Shipment schedule met, software quality, software sales	Client satisfaction, workforce utilization
Competition	External, internal	Independent consultants (freelancers), internal staff
	Open source project	

B. *Fictitious Enterprise Application Architecture*

PremierQuotes Inc. is an insurance company that acquired DirtCheap Insurance, another fictitious insurance company, and formed the PremierQuotes Group (PQG) to fulfill the growth expectations of its stakeholders. Figure 1 shows a simplified, but still realistic Enterprise Application (EA) landscape at PQG comprising of three connected EAs; both external interfaces and internal layer structure are shown from a logical-functional point of view.

PQG exposes its customer care, contract, and risk management applications to three types of external and internal human users, its customers, independent agents, and internal back office staff. There are three user channels, a customer self-service, an agent, and a back office channel. Each of these channels supports one or more business activities initiated by users: enquire, assess risk, calculate rate, sign contract, and create policy. These activities jointly realize a customer enquiry process. The applications work with a customer database, a policy backend, and a government information server, which appear in the system contexts of the applications. The enterprise applications interact with each other. For instance, the customer care application communicates with the contract application (e.g., when processing an enquiry).

The three physical tiers are the client tier, the mid-tier hosting presentation, domain, and resource (data) access logic, and the backend tier. World-Wide Web (WWW) infrastructure connects the client tier with the mid-tier (over the Internet for the customer self-service channel and the agent channel, over an intranet for the back office channel).

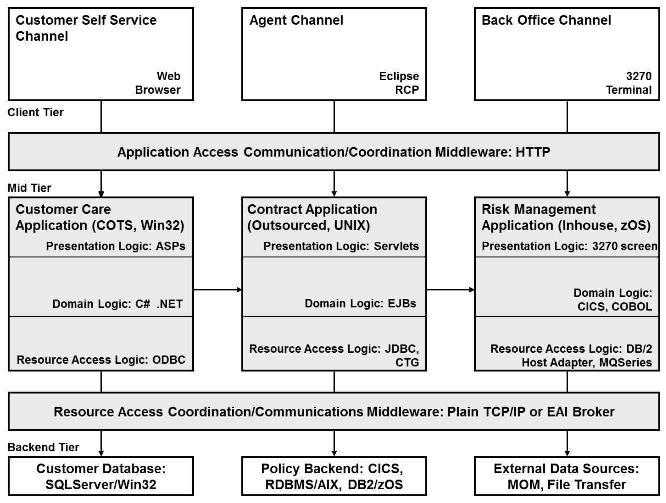
Customer Self Service Channel
Web Browser
Agent Channel
Eclipse RCP
Back Office Channel
3270 Terminal
Client Tier
Application Access Communication/Coordination Middleware: HTTP
Mid Tier
Customer Care Application (COTS, Win32)
Presentation Logic: ASPs
Domain Logic: C# .NET
Resource Access Logic: ODBC
Contract Application (Outsourced, UNIX)
Presentation Logic: Servlets
Domain Logic: EJBs
Resource Access Logic: JDBC, CTG
Risk Management Application (Inhouse, zOS)
Presentation Logic: 3270 screen
Domain Logic: CICS, COBOL
Resource Access Logic: DB/2 Host Adapter, MQSeries
Resource Access Coordination/Communications Middleware: Plain TCP/IP or EAI Broker
Backend Tier
Customer Database: SQLServer/Win32
Policy Backend: CICS, RDBMS/AIX, DB2/zOS
External Data Sources: MOM, File Transfer

Fig. 1. An enterprise information system landscape comprising of three connected systems, organized according to the Layers pattern.

Traditional Enterprise Application Integration (EAI) middleware is used to connect the mid-tier with the backend tier. The client tier contains all application components directly serving the users. Examples are Web browsers and rich client applications running on Personal Computers (PCs) used by customers, agents, and back office staff. The mid-tier comprises of the three applications shown in the system context diagram. These applications are logically layered into presentation, domain, and resource (data) access logic layers. Typical responsibilities of the mid-tier are input validation, processing control, session state management, calculations, and manipulations of enterprise resources. The backend tier stores enterprise resources persistently and coordinates concurrent access to the enterprise resources (i.e., customer profiles, offers, and policies). This tier hosts database servers, but also other systems which in themselves may be physically tiered, but located external to the company or in another organizational domain. The policy backend and the government information server are examples.

An example of a recurring design issue is session state management (e.g., think of a user session in the customer self-service channel). The three top-level design options (patterns) are client session state, server session state and database session state [9]. Client session state scales well, but has security and possibly performance problems; this does not change when moving to a cloud platform. Server session state uses main memory or proprietary data stores in an application server (e.g., an HTTP session in a JEE servlet container); this approach is no longer recommended when deploying to a cloud due to scalability and reliability concerns. Finally, database session state is well supported in many clouds, e.g., via highly scalable key-value storages (NoSQL). This decision has to be made or revisited when designing a cloud-native application; while the decision outcomes may vary, the issues and options to be considered stay the same (i.e., they recur). Multiple instances of this decision may exist per project (e.g., in multi-channel applications).

The PQG architecture serves as a running example in this report. It aggregates many design facets from real projects.

III. Requirements for Arch. Metrics Management

Use cases. During my time in professional services and product development, I defined and applied AMs to (a) make and justify architectural decisions, (b) categorize design problems and solutions regarding their business context and technical complexity and (c) compare similar architectures. Ordering these activities from the end of the project lifecycle to its start and applying the terminology from [10], this experience can be summarized in the following five AMM use cases:

1. Utilize metrics during *architecture and design reviews* to define scope and to assess architectural fitness and adherence to/deviation from recommended practices. This is an aspect of architectural evaluation [10].
2. *Indicate complexity and technical risk*, e.g., to be used as input to effort estimations and project management work. This is an aspect of architectural synthesis [10].
3. *Measure project progress* on a technical level (also during architectural synthesis).
4. Support architect during *transition* from design-time quality attribute specifications to runtime Service Level Agreements (SLAs) and contracts (still in architectural synthesis).
5. *Benchmark architectures* in domain (business) context [14] as a variant of architectural evaluation.

Critical Success Factors (CSFs). SWA metrics and metrics management solutions (methods, tools) should ensure:

- *Expressivity and elicitability*. AMs should be able to support the five AMM use cases effectively and efficiently. It must be possible to obtain them from software architecture documents and code with little extra effort.
- *Intuitivity*. AMs should be self-explanatory: both unit and unit of measurement as in physics must be defined, value ranges should be specified. The AM semantics should be defined at least informally (e.g., by way of examples and counter examples).
- *Unambiguity*. AMs should be well defined and use viewpoint and component/connector terminology, e.g., from IEEE 42010 [12], patterns books [6][9][11], a recognized design method [17], or from the literature about architectural styles (for domain-specific metrics).
- *Sensitivity*. Small changes in the architecture should not lead to radically different AM values (just like continuous functions in mathematics do not have gaps and do not "jump"). AMs should not produce any surprising and misleading analysis/evaluation results.

Elements of risk. Like most computer scientists, many software architects have a natural affinity to numbers. Hence, *misuse* and *blind faith* in AMS are primary risks for AMM. Another risk is *cheating (fraud)*: Metrics can be corrupted or misguided ("do not trust any statistic you did not fake yourself" is a popular quote, e.g., in economics). Hence, AM calibration is required, e.g., based on a mature domain model, reference architecture or pattern language. A resulting requirement for AM researchers and tool developers is to carefully manage expectations and to be candid about what can be achieved (i.e., what AMs can tell and cannot tell). They should recommend complementary techniques such as agile communication and collaboration tools in their AM usage manuals.

IV. General Architectural Metrics by Viewpoint

This section presents AMs by Viewpoint (VP). The 4+1 VPs from [13] are used to structure the section with other VPs blended in. These viewpoint models merely serve as organizing principles here; the presented metrics are also applicable when working with other viewpoint models.

Table 3 gives an overview, and the remainder of the section then walks through the table entries (within the context of the running example from the previous section). I limit the compilation to the three to five most relevant AMs per VP. Many more AMs could be identified; some of these candidates are mentioned in the discussion in Section VI.

When possible, I will describe the AMs in the context of the kinds of decision making process they supported. Since this is an experience report and not a research paper presenting scientifically validated results, certain gaps remain for some elements in the AM compilation (subject/input to discussion).

Scenario VP (SVP). The SVP deals with the architecturally significant parts of the functionality of the system under construction. In object-oriented analysis, use cases specify the functionality [4]; Non-Functional Requirements (NFRs) eliciting Quality Attributes (QAs) [2] complement use cases.

AM-S1: (a) Use Case Count and *(b) Use Case Weight.* My first AM is the number of use cases (a), followed by (b) the number of user-system interactions per use case. This metric is an effort and risk indicator; however, sometimes a project with 1000 use cases is less complex (and risky) than one with 10 use cases modelled; hence, sample use cases have to be studied to make sure that the critical success factors from above are met if this metric is used. Use case count and weight still are useful complexity indicators; unusual numbers might indicate an analysis smell or technical risk factor for the project.

In our running example from the insurance industry, five claims processing use cases are described (enquire, assess risk, calculate rate, sign contract, and create policy); real-world enterprise information systems may implement more than 100 such UCs. Use case modelling best practices recommend around 10 user-system interactions per use case scenario; longer scenarios should be split up to improve manageability of the specification and flexibility of the design.

TABLE 3. SOFTWARE ARCHITECTURE METRICS BY VIEWPOINT

Architectural Metric (AM)	Name	Type (Unit of Measurement)
Scenario Viewpoint	Number and weight of use cases Number of secondary actors (and cadence of external interface connections) Specificity and measurability of NFR/quality attribute specifications	Counter (1…1000) Counter and score Binary score
Logical Viewpoint	Number of external interfaces and number of interface invocations Number of components and connector per component	Counter Counter
Development Viewpoint	(out of scope of this report)	n/a
Process Viewpoint	Process Counter Process Coordination Means Interprocess Communication (IPC) and Remote Call Counter Application State and User Session State Workload Profile	Counter Index/Score Counters Size (Bytes) Aggregated (Complex)
Physical Viewpoint	Tier Counter Clustering Index	Counter Index/Score
Architectural Decision Viewpoint	Number of architecture design problems solved Number of options considered per problem	Counter Counter
Information Viewpoint	Data model size and structure (e.g., number of entities and entity relationships) Transaction management profile, e.g. number of system transactions and their size/duration	Index/Score Aggregated (Complex)
Patterns Metrics *(here: POSA, PoEAA, EIP books)*	E.g. number of layers, number of controllers in MVC pattern E.g. length and complexity of EIP integration flows	Counter Index/Score
Domain- and Style- Specific Metrics *(JEE, SOA, MOM, RDB)*	E.g. number of servlets, number of message channels E.g. number of SQL tables, queries, foreign key relationships	Counter Counter

AM-S2: Secondary Actor Count. Another AM in the SVP is the number of secondary actors in the use case: A high number often indicates that many external interfaces have to be consumed, which causes integration and test efforts and also adds to the technical project risk. I use this metric both during architectural synthesis and during architectural evaluation (e.g., to find out whether all integration requirements can be satisfied by the proposed/implemented architecture).

In the PQG example, three backend systems are shown, which can be seen as secondary actors; in the telecommunications case study [23], we dealt with 20 such systems; sometime, more than 100 or even 1000 systems have to be integrated (e.g., due to regulatory compliance requirements, or advanced reporting and archiving use cases).

AM-3. Smartness of NFRs. The mere number of QAs does not seem to be an expressive, intuitive and unambiguous metric (to reference three CSFs from Section III). However, an AM I have begun to use in recent years is a binary QA score answering two questions: (1) Is the NFR/QA specific enough (in terms of its context)? (2) Is the NFR/QA measurable? This is a simplified, lean version of Quality Attribute Scenarios (QAS) [2], which is almost as effective as full QASs according to my experience in consulting (and teaching). The SMART goal approach is often used in people and project management, and I adopted its first two facets (properties) for NFR/QA analysis work. Table 4 gives some examples of SMART and less SMART NFRs.

Logical VP (LVP). The LVP pertains to the functional decomposition of the system under construction (partitioning); the architectural elements that can be found in the LVP include components and connectors on varying levels of abstraction and refinement. Hence, the LVP AMs deal with these concepts.

AM-L1: (a) Number of External Interfaces and *(b) Number of Invocations per Interface* (both inbound and outbound). This AM is very important for effort estimation, risk management (from a course on architectural thinking: "external data and communication makes or breaks your project"); it can also be used for performance engineering and the definition and monitoring of subsystem-level Service Level Agreements (SLAs). Appropriate numbers vary by industry and application type; it is not possible to give generic recommendations here.

TABLE 4. SMART NFRS/QAS (EXAMPLES)

NFR/QA Example	Specific? (Rationale)	Measureable? (Rationale)
"In claims processing use case (agent channel), sub-second response times is required for 95% of the agent requests."	Yes	Yes
"The logger component has to be highly maintainable as it will be reused many times"	Yes	No (how to verify that requirement is satisfied?)
"24x7 up time"	No (sub-systems?)	Yes (but not realistic)
"Our software has to be very easy to use."	No (which software?)	No (what does "very easy" mean? who judges this?)

In the PQG example, there are three interfaces, which shown traffic in the order of a few transactions/requests per second. In online trading or control systems, hundreds or thousands of messages per second have to be processed per interface (these extreme differences indicate diversity).

AM-L.2: (a) Number of Components and *(b) Number of Connectors per Component.* This metric will only produce meaningful data if the notion of components and connectors is defined and agreed upon (in terms of their abstraction level and design method employed); if done properly, it can indicate the amount/degree of cohesion and coupling in the componentry (LVP). While this AM is hard to standardize, I typically review a carefully selected subset of components and component descriptions to get a feel for the practices in a development organization (and I adjust my metrics accordingly). If no examples from previous projects or enterprise architecture management group are available, I define them myself (and document my assumptions about component granularity and identification, specification, realization method).

In the PQG example, the layered LVP architecture in Figure 1 can be decomposed into 18 components.

Development VP. This VP is out of scope of this report – not because it is not important from an architect's point of view (faithful to the organizational pattern "architect implements"), but because it is well understood and covered elsewhere in the literature, e.g., object-oriented code metrics are elicited in [15].

Process VP (PVP). The PVP focusses on system dynamics, including operating system processes and their coordination, but also application workflows.

AM-P1: Number of Operating System Processes. This metric is useful for capacity planning: it can indicate hardware requirements, as well as service management needs (e.g., monitoring, configuration management, application security).

AM-P2: Process Coordination Means. This metric is an aggregated one, counting the use of parallel programming concepts, e.g., number of mutexes and semaphors, number of locks set and unset per time interval, etc. Many competing system qualities are affected: if too little coordination is done, accuracy and robustness will be compromised; if the solution is over-engineered and/or if inadequate means are chosen, the system becomes slow and hard to test and maintain over time.

AM-P3: (a) IPC Index and *(b) Remote Call Counter.* The number of open Interprocess Communication (IPC) features (e.g., TCP/IP socket connections, queues, shared memory, operating system pipes) and the number of message sent/received over these connections is important to know when for IT infrastructure design, e.g., in response to scalability requirements. According to M. Fowler, the best remote call is one you do not make [9] (which is easier said than done).

AM-P4: (a) Size of Application State and (b) *Size of User Session State*, as well as access profiles and state changes over time are important to know about when designing for scalability and robustness. A detailed description of this AM is out of scope here due to space constraints. Refer to [8] for a more detailed coverage of state management options in the context of cloud computing and Section V for information on HTTP sessions.

AM-P5: Workload Profile. The workload profile is an aggregated metric that should include number of requests per second (i.e, end user requests sent and/or responded to, both in normal operations and in failure situations) as well as an application's hardware footprint (storage and CPU demand). A detailed description of this AM is out of scope here due to space constraints. Five workload patterns commonly occurring in cloud solutions are described in the literature [8].

Deployment VP a.k.a. Physical or Operational VP (OVP). The OVP deals with the assignment of software components to hardware nodes and other IT infrastructure elements (network, storage devices) and the resulting IT infrastructure topology.

AM-O1: *Tier Counter.* This is a simple measurement for the distribution of processing logic and storage units – more complex scores might be more expressive, but it is not obvious how such advanced AMs could be obtained (so that they still meet the CSFs from Section III).

The tier design has a large impact on performance (latency vs. throughput), and also infrastructure and network security. In the PQG example, a three-tier structure is used. This is common in EIS design today; many 2-tier systems exist as well.

AM-O2: Clustering Index. This AM indicates the amount of redundancy in the deployment (i.e., the ratio of deployment units and logical components to nodes that host these deployment units). Clustered deployments are much more difficult to design, test, troubleshoot and modernize than standalone ones, but also more robust and performant (if set up properly). A clustering index of 0 means that there is no redundancy in the deployment and the value 100 means that each component is deployed exactly twice. The metrics has to be defined and measured per subsystem (or even component), with detailed arithmetic to be defined. All extremes from 0% to 100% redundancy occur in practice, depending on availability, failover and scalability requirements.

Architectural Decision VP (DVP). This VP is not part of the 4+1 model, but has been conceptualized by the Architectural Knowledge Management (AKM) research community since 2004. It deals with design rationale. If Architectural Decisions (ADs) are made explicit, two AMs for AKM are:

AM D-1: Architecture design problems solved, e.g., number of decisions made vs. number of decisions documented.

AM D-2: Options considered per problem, e.g., patterns or technologies or assets.

AKM metrics are subject to ongoing research [22]. The SOAD project reported on AKM metrics by example [20].

Information VP (IVP). This VP also is not part of the 4+1 model, but introduced in [18]. It deals with data representation and management.

AM I-1: *(a) Data Model Size and (b) Data Model Structure.* This AM may include the number of databases/schemes, number of database tables, number of columns and rows per table, number of concurrent *clients*, etc. These numbers impact performance and maintainability as well as migratability.

AM I-2: (a) Transaction Volume and (b) Transaction Weight. This AM covers the number of system transactions and scope of their boundaries, the amount of SQL statements executed within the transaction, and their execution time.

Descriptions of such metrics and recommended values can be found in the database and transaction processing literature.

V. Domain-Specific Metrics (Frameworks and Patterns)

Let me now transition from platform-independent to platform-specific metrics. Anything that appears in the domain model or pattern language for a technology could be measured, for instance, the key concepts in a framework or the components in a solution sketch. For instance, when integrating systems over the Web with RESTful HTTP, the number of resources and their nesting structure are of interest (as well as the number of states in finite state machine specifying the valid state transfers and the number and type of resource representations along with their media types and link types).

Patterns books. This subsection lists some key metrics that can be obtained from patterns that I applied on the projects. I do not aim for completion here, but highlight a few examples that I find particularly relevant, illustrative and representative.

AM POSA-1: Layers Usage. In the Layers pattern [3], the number of layers, the number of components per layer, the number of calls from layer to layer as well as the number of layering violations concern the architect (and maintainers of the system) due to their impact on maintainability.

AM PoEAA-1: MVC Scores. In the Model-View-Controller Pattern (MVC) that governs the design of many contemporary presentation layers, the number of controllers, the number of views per controller, and the size/weight of the model instances are relevant AMs. These numbers can help identify performance and scalability bottlenecks in the architecture and its realizations, and also indicate storage requirements.

JEE/SOA. When working with reference architectures such as Java Enterprise Edition (JEE) and architectural styles such as Service-Oriented Architecture (SOA), a number of style- and technology-specific AMs can be defined. I can only name the most relevant of these style- and domain-specific AMs here.

AM-JEE. JEE AMs include number of servlets and JSPs per Web container, EJBs per JEE container, EJB containers per application server, number of database connections and data sources number of database connections, connection pools and caches; size and structure of configuration elements, properties, e.g., in http.conf or web.xml files; number and cadence of backend connections vs. connections to peer systems (see PQG example in Section II). If other container technologies are used, these metrics can be adjusted, e.g., the number of Spring beans in a Spring container can be counted. Examples of typical component numbers are 10 to 30 servlets and EJBs per container; and 5 to 10 to 15 containers per server; HTTP sessions should only contain a few KB due to scalability issues. JMX MBeans (e.g., visible in JConsole) also qualify as AMs.

The reasons for defining and using these metrics are: they give an idea about system management needs, the required space on the heap, the startup times for middleware that has to process configuration files and annotations (which is often done via introspection/reflection, which are resource-intense and time consuming activities for a language runtime such as the Java virtual machine).

AM-SOA1. (a) Number of Service Endpoints and (b) Weight of Services (service granularity). Service metrics may also include: interface breadth and interface depth both from a technical point of view and a business point of view, and service versioning frequency. Metrics for message exchange formats such as JSON and XML metrics are applicable well.

AM-SOA2. (a) Number and (b) Complexity of Service Composition Workflows, including average execution time, number of process instances, and number of compensating events. The business process management community defines such metrics (e.g., in process mining).

EIP/MOM. Enterprise Integration Patterns (EIP) [11] describe asynchronous, loosely coupled communication via Message-Oriented Middleware (MOM). EIP/MOM metrics include number of application clients/endpoints (e.g., number of competing consumers), number of queues and messages in queue, rate of message production, rate of consumption; number of channels, e.g., n+m vs. n*m for EAI point-to-point vs. hub-and-spoke. Important AMs for the publish-subscribe-pattern are number of subscribers and number of topics.

EIP/MOM metrics used in integration flow design include length of routes and the number and complexity of message transformations. The nesting level of XML and JSON structures and the ratio of metadata vs. payload is also worth calculating and keeping track of (e.g., an overhead of 4 to 20 is attached to SOAP/HTTP in comparison to raw data to be exchanged according to my experience; I regularly enquire about this number when reviewing Web services solutions).

RDBMS/SQL. The following AMs have proven to be useful for me (among others): number of entities (on conceptual level) and tables (on logical/physical level), cardinalities (i.e., multiplicities) of relationships in entity-relationship diagrams or domain models (with average and extreme instantiations), number of database triggers and foreign keys, number of SQL statements defined and executed per subsystem/user service.

VI. Discussion

Missing metrics. One key aspect in evaluating architectures is whether it is appropriate for the architecturally significant requirements. None of the metrics compiled in this report really try to measure this; they can be only used in the sense of best practices. As a consequence, they are only useful if the user knows how to interpret them; domain-specific architecting experience is required for that.

One could also measure the component density in the LVP, e.g., the scope and content of classes-responsibilities-collaborations cards (subject to discussion). Regarding PVP AMs, one can also think of a performance tuning score capturing number and size of caches, caching performance (successful lookups vs. unsuccessful lookups). Other PVP AMs might deal with lifecycle events (start and stop of processes, retries, database commit vs. rollback, compensation routines fired, exceptions thrown and caught (per subsystem, per layer).

It is subject to discussion whether Strategic Outsourcing (SO), IT service management and DevOps metrics also belong to the OVP viewpoint, e.g., number of images, management scripts, help desk tickets, audit log records, etc. These AMs can also be seen as belonging to a separate management viewpoint (or perspective in Rozanski/Woods terminology [18]). The same holds for security metrics, including (but not limited to) connection attempts that were rejected, unsuccessful logins, and other security incidents and events. When reviewing architectures from an OVP, I sometimes also enquired about (or counted myself): number of firewalls by type, number of locations and network zones, number of nodes per location, number of deployment units per node, number of cluster members. Hardware specification as used in capacity planning can also be collected: CPU, RAM, disk, network adapters, etc.

Value and maturity. According to my experience, AMs can be a valid and powerful tool – if handled with care (see elements of risk presented in Section III and usage examples presented throughout Section IV). The presented metrics served me well to prepare reviews (e.g., with questionnaires and checklists).

Faithful to the nature of an industry experience report, the presentation of the metrics was subjective and anecdotal; I did not contribute validated research results in this article. Therefore, the presented metrics are not fully flushed out yet and do not satisfy all critical success factors from Section III yet. AMM research will be required to do so.

Feasibility. An initial reaction to my AM compilation might be that it has too many elements – in practice, it might be too hard to collect and to interpret all of these AMs. However, a subset can be chosen as needed (tailoring). Custom catalogs (profiles) can be defined; the purpose of my collection is to serve as a discussion starter and practitioner input to the workshop.

From my point of view it is an open research question how the usefulness (or the benefit for a specific question) of architectural metrics in general could be evaluated. It would also be worth to investigate the architectural decisions that the AMs they feed into; possibly there are there bad combinations of AM values that can serve as architectural smells [25].

VII. Summary and Outlook

In this article I reported in on my past and present usage of architecture metrics. I identified use cases, critical success factors and elements of risk for Architectural Metrics Management (AMM) and presented examples of useful metrics organized by viewpoints and domains. As such, this report provides food for thought for the workshop discussions, and intends to serve as input to an AMM research roadmap. Hopefully the use cases and critical success factors as well as the metrics compilation by viewpoint and domain, as well as the running example (a fictitious, but realistic enterprise information system landscape), will help researches to come up with automated approaches to metrics gathering, visualization and analysis, and may also help them to validate their research.

Additional metrics for other domains and styles can be identified, e.g., for distributed control systems, cloud computing, or microservices. In my own future work, I will continue to focus on the Architectural Decision Viewpoint, including some architectural metrics for it. Another promising direction is tool support for making non-functional requirements SMARTer (i.e., more specific and measurable). Finally, the interface between architectural synthesis and project management as well as architectural refactoring practices are additional areas that can benefit from architectural metrics.

References

[1] M. Ali Babar, T. Dingsøyr, P. Lago, H. van Vliet (eds.), Software Architecture Knowledge Management: Theory and Practice, Springer-Verlag, 2009.

[2] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Second Edition. Addison Wesley, 2003.

[3] F. Buschmann, R. Meunie, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture – a System of Patterns. Wiley, 1996.

[4] A. Cockburn, Writing Effective Use Cases. Addison-Wesley, 2001.

[5] P. Eeles, P. Cripps, The Process of Software Architeting, Addison Wesley, 2009.

[6] E. Evans, Domain-Driven Design. Tackling Complexity in the Heart of Software. Addison Wesley, 2003.

[7] G. Fairbanks, Just Enough Software Architcture: A Risk-Driven Approach. Marshal and Brainerd, 2010.

[8] C. Fehling, F. Leymann., R. Retter, W. Schupeck, P. Arbitter, Cloud Computing Patterns, Springer 2013.

[9] M. Fowler, Patterns of Enterprise Application Architecture. Addison Wesley, 2003.

[10] C. Hofmeister, P. Kruchten, R. Nord, J. H. Obbink, A. Ran, P. America, A General Model of Software Architecture Design Derived from Five Industrial Approaches. Journal of Systems and Software 80(1), Elsevier, 2007.

[11] G. Hohpe, B. Woolf, Enterprise Integration Patterns. Addison Wesley, 2004.

[12] ISO/IEC/IEEE, Systems and software engineering – Architecture description, ISO/IEC/IEEE 42010:2011(E), Dec. 1 2011

[13] P. Kruchten, The 4+1 View Model of Architecture, IEEE Software, Volume 12, Number 6, November 1995, pp. 42-50.

[14] P. Kruchten, The frog and the octopus: a conceptual model of software development, http://arxiv.org/ftp/arxiv/papers/1209/1209.1327.pdf

[15] M. Lanza, R. Marinescu, Object-Oriented Metrics in Practice – Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems, Springer, 2006.

[16] C. Manteuffel, D. Tofan, H. Koziolek, T. Goldschmidt, P. Avgeriou: Industrial Implementation of a Documentation Framework for Architectural Decisions. Proc. Of IEEE/IFIP WICSA 2014, IEEE Computer Society, Los Alamitos (2014), pp. 225-234.

[17] Open Unified Process (OpenUP), http://epf.eclipse.org/wikis/openup/

[18] N. Rozanski, E. Woods, Software Systems Architecture : Working With Stakeholders Using Viewpoints and Perspectives, Addison Wesley, 2005.

[19] The official site of the Stage-Gate®, http://www.stage-gate.com/

[20] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, N. Schuster, Managing Architectural Decision Models with Dependency Relations, Integrity Constraints, and Production Rules. Journal of Systems and Software, Elsevier. Volume 82, Issue 8, August 2009, pp. 1249-1267.

[21] O. Zimmermann, C. Miksovic, J. Küster, Reference Architecture, Metamodel and Modeling Principles for Architectural Knowledge Management in Information Technology Services. Journal of Systems and Software, Elsevier. Volume 85, Issue 9, Sept. 2012, , pp. 2014-2033.

[22] O. Zimmermann, L. Wegmann, H. Koziolek, T. Goldschmidt, Architectural Decision Guidance across Projects, Proc. of IEEE/IFIP WICSA 2015, , IEEE Computer Society, Los Alamitos (2015).

[23] O. Zimmermann, V. Doubrovski, Grundler, K. Hogg, Service-Oriented Architecture and Business Process Management in an Order Management Scenario: Rationale, Concepts, Lessons Learned. Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '05). ACM, 2005, pp. 301-312.

[24] O. Zimmermann, S. Milinski, M. Craes, F. Oellerman., Second Generation Web Services-Oriented Architecture in Production in the Finance Industry, Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '04). ACM, 2004, pp. 283-289.

[25] O. Zimmermann, Architectural Refactoring – A Task-Centric View on Software Evolution. IEEE Software, Volume 32, Issue 2, March-April 2015, pp. 26-29.