Concept/Topic: *Architectural Significance*

Context

Every *software-intensive system*[1] has an architecture; the architecture of such system pertains to hardware and software and can be analyzed, synthesized, evaluated and documented via multiple *viewpoints* (see separate concept fact sheet) in an effort to manage complexity.

All architecture is design, but not all design is architecture according to Grady Booch[2]; hence, it has to be decided what is in and out of scope of any architectural activity. The notion of *architectural significance* has this purpose; it is a property of the requirements that are elicited/stated for a particular system and/or project, as well as design elements and decisions considered aling the way. First and foremost, these requirements are non-functional ones (also known as quality attributes); however, limiting architectural significance to these quality attributes is an oversimplification. Functional requirements can be architecturally significant as well, for instance if a feature request leads to the need for an additional external data provider that has to be integrated via an Application Programming Interface (API).

Definition and Assessment

Architectural significance can be characterized by looking at the purpose of software architecture (manage complexity, reduce project risk, enhance developer productivity, ensure long-time quality and changeability) and its translator/communicator role between external and internal stakeholders, including business people and technologists. Criteria to assess whether any given requirement, constraint, decision or other concern is architecturally significant have been discussed occasionally in the literature (on a rather high level of abstraction). Three major themes can be distinguished:

- Economic significance, i.e., cost and risk as for instance pointed out by G. Booch, P. Eeles, and E. Poort (see links at the bottom).
- Technical difficulty and novelty, "shared understanding" about the "hard stuff" (R. Johnson, M. Fowler), also identified by P. Eeles (see below).
- Dependencies, both organizational ones and technical ones, and contextual relevance (G. Hohpe, P. Kruchten).

A requirement that has architectural significance with respect to one or more of these themes is called an *Architecturally Significant Requirement (ASR)*. In addition to ASRs, architectural significance can also be discussed for the *elements* of/in a system and for related *decisions* (which brings us back to the two definitions of software architecture, see separate concept fact sheet on Software Architecture Fundamentals[3]).

ASR Checklist

Indicators for architectural significance of *requirements* that have been reported in the literature include (adapted and revised from the Wikipedia page on ASRs[4]):

1. The requirement is directly associated with high business value and/or technical risk.

[1] A software-intensive system is any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole according to IEEE 1471.
[2] As quoted here: http://handbookofsoftwarearchitecture.com/?page_id=70
[3] ./ZIO-ArchitectureFundamentalsFactSheet.pdf
[4] https://en.wikipedia.org/wiki/Architecturally_significant_requirements

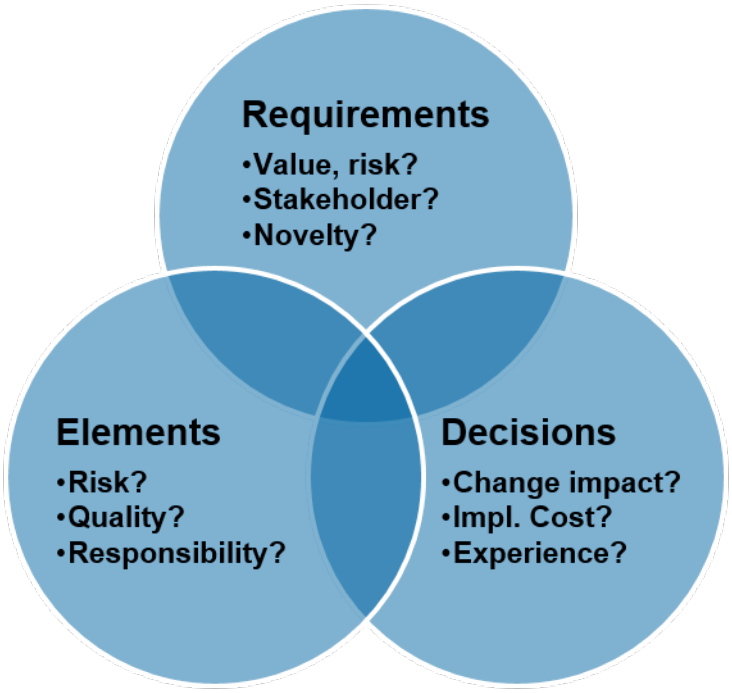

Figure 1: *Three ASR Dimensions: Requirements, Architectural Elements, Decisions*

2. The requirement is a concern of a particularly important stakeholder (for instance, an influential external stakeholder such as project sponsor or compliance auditor); its is governed by an official Service Level Agreement (SLA).
3. The requirement has a first-of-a-kind character: already existing components have different responsibilities and therefore do not address it; the team has never built a component that satisfies this particular requirement.
4. The requirement has Quality-of-Service (QoS) characteristics that deviate from those already satisfied by the evolving architecture substantially (e.g., by an order of magnitude).
5. The requirement has caused critical situations, budget overruns or client dissatisfaction in a previous project with a similar business/technical context.

Criteria for Architectural Elements

P. Eeles (of IBM Rational) argued that an architecture is concerned with significant elements (which can be components, connectors or any other items appearing in one of the definitions of software architecture listed and compared in his introductory article[5]. His checklist for architectural significance of *elements* is:

1. "The element relates to some critical functionality of the system, e.g., monetary transactions
2. The element relates to some critical property of the system, e.g., reliability
3. The element relates to a particular architectural challenge, e.g., external system integration
4. The element is associated with a particular technical risk
5. The element relates to a capability that is considered to be unstable
6. The element relates to some key element of the solution, e.g., login mechanism"

These criteria are met by many design concerns; additional qualification and prioritization is required in practice (for instance, risk- and cost-based).

[5] https://www.ibm.com/developerworks/rational/library/feb06/eeles/index.html

Questions about Decisions

S. Toth asks these questions to classify the architectural significance of *decisions* that justify if and how a requirement or other concern is met (on page 73 of his book "Vorgehensmuster für Softwarearchitektur", in German):

1. Is the decision hard to change later?
2. Is the decision expensive to implement/execute upon?
3. Are demanding, qualitative requirements stated (high security level, high availability, high performance)?
4. Are requirements difficult to map to existing (solutions, experiences)?
5. Is the experience in the solution space weak (in the team)?

The above questions take inspiration from G. Fairbanks, the author of "Just Enough Software Architecture", who sees the ability to reason about potential failures (and how to avoid) them as the main difference between architecture and design.

Assessment Matrix

The criteria for the significance of requirements, elements, decisions can be combined, for instance in a two dimensional matrix mapping external vs. internal stakeholder concerns (yielding four quadrants):

	low difficulty/novelty (internal concerns)	*high difficulty/novelty (internal concerns)*
high cost/risk (external stakeholder concern)	medium to high significance	high significance
low cost/risk (external stakeholder concern)	low significance	low to medium significance

Assessing the architectural significance requires context information and is somewhat subjective and therefore hard to measure absolutely; while a binary yes/no rating can be given, a relative high-medium-low scoring is often used in practice. *Context* matters (as defined and elaborated by P. Kruchten in a blog post[6] and more detailed journal paper referenced there).

Examples

The following table scores some requirements and constraints according to their architectural significance:

[6] https://philippe.kruchten.com/2011/02/10/the-frog-and-the-octopus-go-to-snowbird/

Requirement	Score	Explanation (Rationale)
"Data retention policy of 10 years required to achieve regulatory compliance"	*high*	Violation of this requirement would lead to fines; redesign might require change of database technology and hosting model
"Technical constraint to prefer a particular messaging middleware and backend API"	*medium-high*	If a different architectural element is chosen, additional licensing and training cost arise; standardized APIs promise interoperability (which has to be proven)
"Deployment pipeline automation"	*low-medium*	Needed to be able to test, deploy, release often, but not something end users and project sponsors are willing to care about and pay for usually (so team-internal concern)
"Name of Java class wrapping access to backend"	*low*	Decision that is not visible to external stakeholders; simple to change in IDEs that support refactoring

Application in Products and Projects

Typically, architectural significance is assessed by intuition (of the involved architects, other other team members performing architectural activities), sometime also called "gut feel" (in practitioner jargon) or "tacit knowledge" (in science lingo). Several checklists can be found in books and online resources (see questions/examples above and references below). An architectural significance tag or score can be added to the backlog entry is the agile task planning tool (or other requirements management tool); the assessment can be fed into project management and team self organization effort (for example, sprint planing meetings and retrospectives, daily standup meetings). In a software evolution and maintenance context, this leads to the notion of technical debt (details of which are out of scope of our lecture for the time being).

Tips and Tricks

- Agree on criteria early, prioritize quality attributes and define critical success factors for your project/product development effort (and use these to assess architectural significance of stakeholder concerns and decisions required)
- If you cannot agree on the significance of a particular requirement, keep it as a backlog item or maintain a parking lot of possibly significant issues for later consideration.
- Follow an outside-in approach, starting from the project sponsor and other key stakeholders, as well as external interfaces that you will have to consume (and therefore rely on).

Follow-On Topics and Concepts

- SMARTER[7] Non-Functional Requirements (NFRs)
- Risk- and Cost-Driven Architecting (RCDA)[8], a method developed and taught by an architect from the Logica CGI consulting firm.

[7] https://www.projectsmart.co.uk/smart-goals.php
[8] http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=48584

- Just Barely Good Enough (JBGE) metaphor in Scott Ambler's Agile Modeling[9] set of practices.

More Information

- Wikipedia page on ASRs: https://en.wikipedia.org/wiki/Architecturally_significant_requirements
- OpenUP[10] has a concept called Architectural Significance: http://epf.eclipse.org/wikis/openup/core.tech.common.extend_supp/guidances/concepts/arch_significant_requirements_1EE5D757.html
- Peter Eeles, http://www.architecting.co.uk/presentations/Architecting%20Large-Scale%20Systems.pdf and https://www.ibm.com/developerworks/rational/library/feb06/eeles/
- George Fairbanks, Just Enough Software Architecture, https://www.infoq.com/articles/fairbanks-jesa
- Stefan Toth, Vorgehensmuster für Software-Architektur, Seite 72f., http://www.swamuster.de/musteruberblick/die-basis-fur-architekturarbeit/ (in German)

[9] http://agilemodeling.com/essays/barelyGoodEnough.html
[10] http://epf.eclipse.org/wikis/openup/