Architectural Analysis

Dalibor Dvorski (ddvorski@conestogac.on.ca)
School of Engineering and Information Technology
Conestoga College Institute of Technology and Advanced Learning

Architectural Analysis

- can be viewed as a specialization of requirements analysis, with a focus on requirements that strongly influence the “architecture”
  - e.g. identifying the need for a highly-secure system
- the essence of architectural analysis is to identify factors that should influence the architecture, understand their variability and priority, and resolve them
- the difficult part is knowing what questions to ask, weighing the trade-offs, and knowing the many ways to resolve an architecturally significant factor, ranging from benign neglect, to fancy designs, to third-party products
- a good architect earns her salary by having the experience to know what questions to ask and choosing skillful means to resolve the factors
- concerned with the identification and resolution of the system’s non-functional requirements in the context of the functional requirements

Architectural Analysis

Why is architectural analysis important?

- reduce the risk of missing something centrally important in the design of the systems
- avoid applying excessive effort to low priority issues
- help align the product with business goals

When do we start architectural analysis?

In the UP, architectural analysis should start even before the first development iteration, as architectural issues need to be identified and resolved in early development work. Failure to do so is a high risk. e.g. Deferring an architecturally-significant factor such as “must be internationalized to support English, French, and Croatian” or “must handle 500 concurrent transactions with on-average one-second response time” until late in development is a recipe for pain and suffering. → However, since the UP is iterative and evolutionary – *not waterfall* – we start programming and testing in early iterations before all the architectural analysis is complete.

Architectural Analysis

- **variation point**: variations in the existing current system or requirements, such as the multiple tax calculator interfaces that must be supported
- **evolution point**: speculative points of variation that may arise in the future, but which are not present in the existing requirements
- architectural analysis is concerned with the identification and resolution of the system's non-functional requirements (e.g. security), in the context of the functional requirements (e.g. processing sales)
    - it includes identifying variation points and evolution points

Examples of Issues to be Identified and Resolved at Architectural Level

How do reliability and fault-tolerance requirements affect the design?

e.g. In the NextGen POS, for what remote services (e.g. tax calculator) will fail-over to local services be allowed? Why? Do they provide exactly the same services locally as remotely, or are there differences?

How do the licensing costs of purchased subcomponents affect profitability?

e.g. The producer of an excellent database server wants 2% of each NextGen POS sale if their product is used as a subcomponent. Using their product will speed development (and time to market) because it is robust and provides many services, and many developers know it, but at a price. Should the team instead use another less robust, open source database server? At what risk? How does it restrict the ability to charge for the NextGen product?

Examples of Issues to be Identified and Resolved at Architectural Level

How do the adaptability and configurability requirements affect the design?

e.g. Most retailers have variations in business rules they want represented in their POS applications. What are the variations? What is the “best” way to design for them? What is the criteria for the best? Can NextGen earn more money by requiring customized programming for each customer (and how much effort will that be?), or with a solution that allows the customer to add the customization easily themselves? Should “more money” be the goal in the short-run?

Examples of Issues to be Identified and Resolved at Architectural Level

How does brand name and branding affect the architecture?

A little-known story is that Microsoft's Windows XP was not originally named "Windows XP". The name was a relatively last-minute change from the marketing department. You may appreciate that the operating system name is displayed in many places, both as raw text and as a graphic image. Because the Microsoft architects did not identify a name change as a likely evolution point, there was no Protected Variation solution for this point, such as the label existing in only one place in a configuration file. Therefore, at the last minute, a small team scoured the millions of lines of source code and image files, and made hundreds of changes.

Common Steps in Architectural Analysis

1. Identify and analyze the non-functional requirements that have an impact on the architecture. Functional requirements are also relevant (especially in terms of variability or change), but the non-functional are given thorough attention. In general, all these may be called **architectural factors** (i.e. architectural drivers).
    - This step could be characterized as regular requirements analysis, but since it is done in the context of identifying architectural impact and deciding high-level architectural solutions, it is considered a part of architectural analysis in the UP.
    - In terms of the UP, some of these requirements will be roughly identified and recorded in the Supplementary Specification or use cases during inspection. During architectural analysis, which occurs in early elaboration, the team investigates these requirements more closely.
2. For those requirements with a significant architectural impact, analyze alternatives and create solutions that resolve the impact. These are **architectural decisions**.
    - Decisions range from “remove the requirement” to a custom solution to “stop the project” to “hire an expert”.

The Science: Identification and Analysis of Architectural Factors

Architectural Factors:

Any and all of the FURPS+ requirements may have a significant influence on the architecture of a system, ranging from reliability, to schedule, to skills, and to cost constraints.

However, the factors with the strongest architectural influence tend to be within the high-level FURPS+ categories of functionality, reliability, performance, supportability, implementation, and interface. It is usually the non-functional quality attributes (such as reliability or performance) that give a particular architecture its unique flavor, rather than its functional requirements. e.g. The design in the NextGen POS to support different third-party components with unique interfaces, and the design to support easily plugging in different sets of business rules.

In the UP, these factors with architectural implications are called *architecturally significant requirements*.

The Science: Identification and Analysis of Architectural Factors

Quality Scenarios:

When defining quality requirements during architectural factor analysis, *quality scenarios* are recommended, as they define measurable (or at least observable) responses, and thus can be verified. It is not much use to vaguely state “the system will be easy to modify” without some measure of what that means.

Quantifying some things, such as performance goals and mean time between failure, are well known practices, but quality scenarios extend this idea and encourage recording all (or at least, most) factors as measurable statements.

Quality scenarios are short statements of the form <stimulus> <measurable response>. e.g. 1. When the completed sale is sent to the remote tax calculator to add the taxes, the result is returned within 2 seconds “most” of the time, measured in a production environment under “average” load conditions. e.g. 2. When a bug reported arrives from a NextGen beta test volunteer, reply with a phone call within 1 working day.

Note: “most” and “average” will need further investigation and definition by the NextGen architect; a quality scenario is not really valid until it is testable, which implies fully specified.

The Science: Identification and Analysis of Architectural Factors

Writing these quality scenarios can be a mirage of usefulness. It's easy to write these detailed specifications, but not to realize them. Will anyone ever really test them? How and by whom? A strong dose of realism is required when writing these; there's no point in listing many sophisticated goals if no one will ever really follow through on testing them.

Focus on writing quality scenarios for the important goals, and follow through with a plan for their evaluation.

e.g. 1. In an airline reservation system, consistently fast transaction completion under very high load conditions is truly critical to the success of the system – it must definitely be tested.

e.g. 2. In the NextGen POS, the application really must be fault-tolerant and fail over to local replicated services when the remote ones fail – it must definitely be properly tested and validated.

e.g. 3. The case of Google, their search engine, and database normalization.

The Art: Resolution of Architectural Factors

One could say the *science* of architecture is the collection and organization of information about the architectural factors, as in the factor table. The *art* of architecture is making skillful choices to resolve these factors, in light of tradeoffs, interdependencies, and priorities.

Adept architects have knowledge in a variety of areas (e.g. architectural styles and patterns, technologies, products, pitfalls, and trends) and apply this to their decisions.

The art of the architect is knowing what battles are worth fighting – where it's worth investing in designs that provide protection against evolutionary change.

Promotion of Architectural Patterns and Themes in Architectural Analysis

The most common mechanism to achieve low coupling, protected variation, and a separation of concerns at the architectural level is the Layers pattern (which has been introduced previously). This is an example of the most common separation technique – modularizing concerns into separate components or layers.

The first theme to note is that “architectural” concerns are especially related to non-functional requirements, and include an awareness of the business or market context of the application. At the same time, the functional requirements (e.g. processing sales) cannot be ignored; they provide the context within which these concerns must be resolved. Further, identification of their variability is architecturally significant.

A second theme is that architectural concerns involve system-level, large-scale, and broad problems whose resolution usually involves a large-scale or fundamental design decisions; e.g. the choice of – or even use of – an application server.

Promotion of Architectural Patterns and Themes in Architectural Analysis

A third theme in architectural analysis is interdependencies and tradeoffs. e.g. Improved security may affect performance or usability, and most choices affect cost.

A fourth theme in architectural analysis is the generation and evaluation of alternative solutions. A skilled architect can offer design solutions that involve building new software, and also suggest solutions (or parallel solutions) using commercial or publicly available software and hardware. e.g. Recovery in a remote server of the NextGen POS can be achieved through designing and programming “watchdog” processes, or perhaps through clustering, replication, and fail-over services offered by some operating system and hardware components. Good architects know third-party hardware and software products.

Assigned Readings

1. Chapter 32: More SSDs and Contracts from
   Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.
2. Chapter 33: Architectural Analysis from
   Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.
   - Pay attention to the presented factor tables, use cases, and technical memos, as well as how architectural analysis fits into the UP.