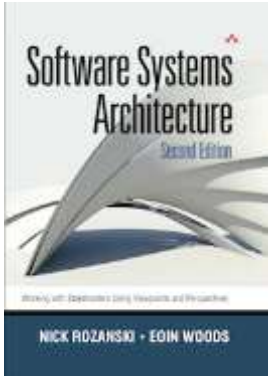
Introduction to Software Architecture

Emmanuel Letier

References

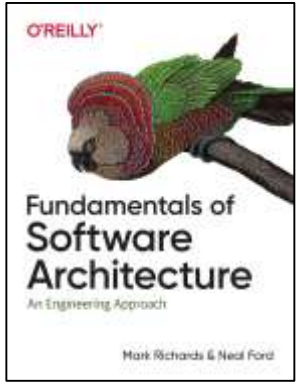
N. Rozanski and E. Woods, *Software Systems Architecture : Working With Stakeholders Using Viewpoints and Perspectives, 2nd Ed.,* Addison-Wesley, 2012.

http://www.viewpoints-and-perspectives.info/



Mark Richards and Nearl Ford, *Fundamentals of Software Architecture: An Engineering Approach, O'Reilly, 2020*.

Video lectures on https://learning.oreilly.com/



Software Architecture

(Woods & Rozanski, *Software Systems Architecture, 2012)*

A software system's architecture is *the set of principal design decisions* made about the system. It includes decisions about the system's

- externally visible behaviours (functionalities) and quality properties (e.g. performance, availability)
- static structures (its internal design elements and their arrangement)
- dynamic structures (its run-time elements and their interactions)
- architectural principles (core ideas that guide the definition of the architecture)

Why care about software architecture?

Every system has an architecture, whether it is documented and understood or not.

A good architecture makes it easier to

- satisfy the functional and quality requirements
- understand how the software works
- analyze the software properties
- test the software
- maintain and evolve the software

A bad architecture makes all these things much harder, and sometimes impossible.

Architecture and Quality Requirements

- Architecture decisions impact quality requirements
  - performance, evolvability, availability, security, cost, …

Example: An online auction system

- a simple client-server architecture
  - handles a few simultaneous auctions and users (e.g. for a school or small art gallery)
- a large-scale globally distributed architecture
  - handles millions of simultaneous auctions and users across the world with high availability, security and performance (e.g. ebay)

Architecture Tradeoffs

- Designing an architecture involves trade-offs
  - every architecture decision has pros and cons
  - every technology has pros and cons
  - stakeholders goals are conflicting

- A good architecture is one that successfully addresses the concerns of its stakeholders and, when those concerns are in conflict, balances them in a way that is acceptable to the main stakeholders.

Architecture as guide rails

- An architecture imposes constraints on what the system's internal elements can and must do

  E.g.

  – the application must use a given component for authentication
  – components of layer i can only use components of layer i+1

- Architecture constraints act as guide rails: developers' freedom is purposefully restricted in order to facilitate
  – satisfaction of quality requirements
  – coordination between developers
  – understanding and analysis of the whole system
  – testing and debugging

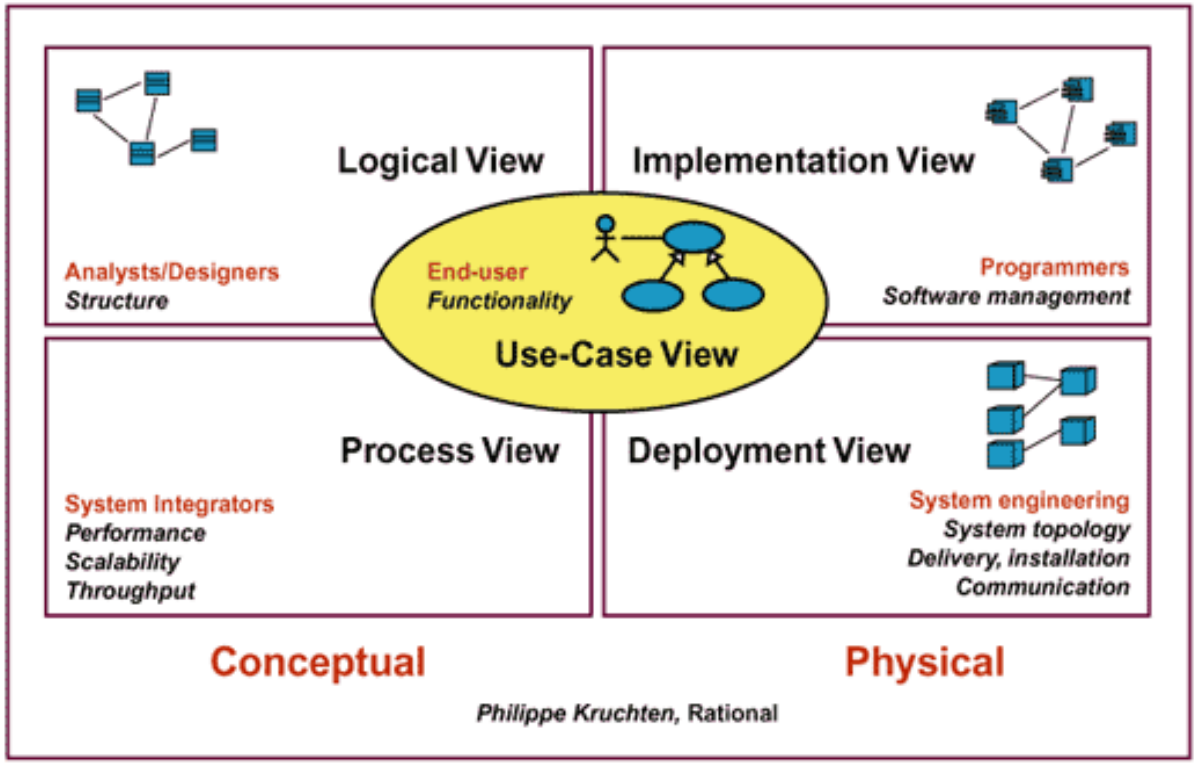
Architecture descriptions as a guidebook

An architecture description is like a guidebook for people who work on the software: clients, developers (current and future), testers, sysadmins, ...

A good architectural description is one that effectively and consistently communicates key aspects of the architecture to the appropriate stakeholders.

Different sections (viewpoints) address different concerns of different people.

Kruchten's 4+1 Architecture Viewpoints (1995)



Rozanski and Woods Viewpoints

http://www.viewpoints-and-perspectives.info/home/viewpoints/

Role of the Software Architect

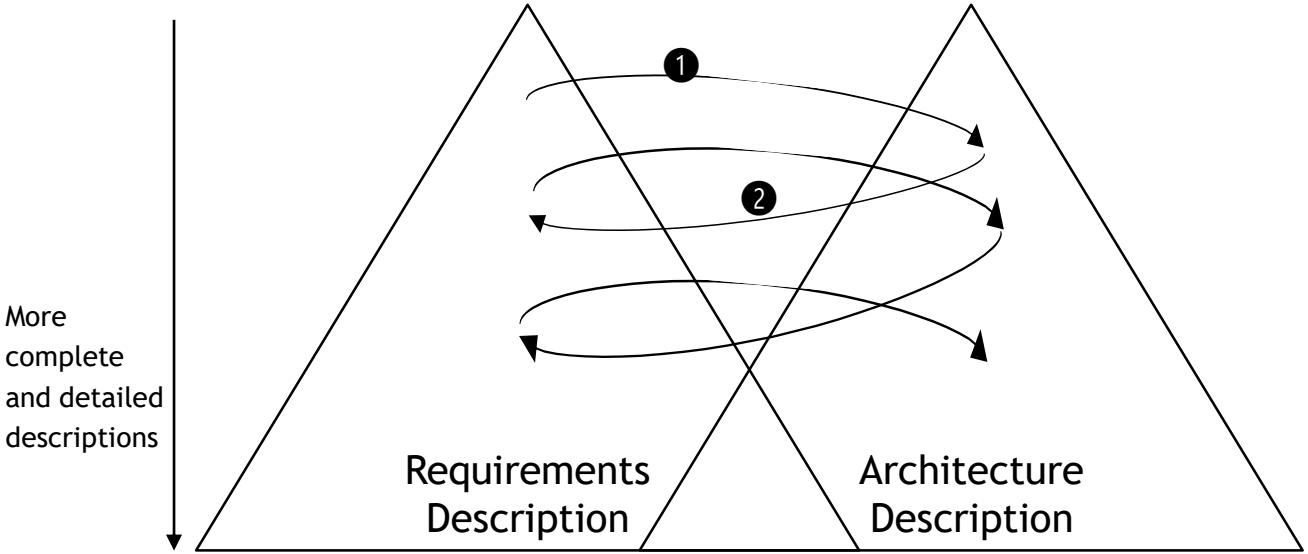
Four main responsibilities:

1. identify and engage stakeholders
2. understand and capture their concerns
3. create and take ownership of the architectural description
4. take a leading role in the realisation of the architecture

Rozanski and Woods, Software Systems Architecture, Addison Wesley, 2012

The Architecture Definition Process

Requirements and Architecture: the Twin Peaks Model

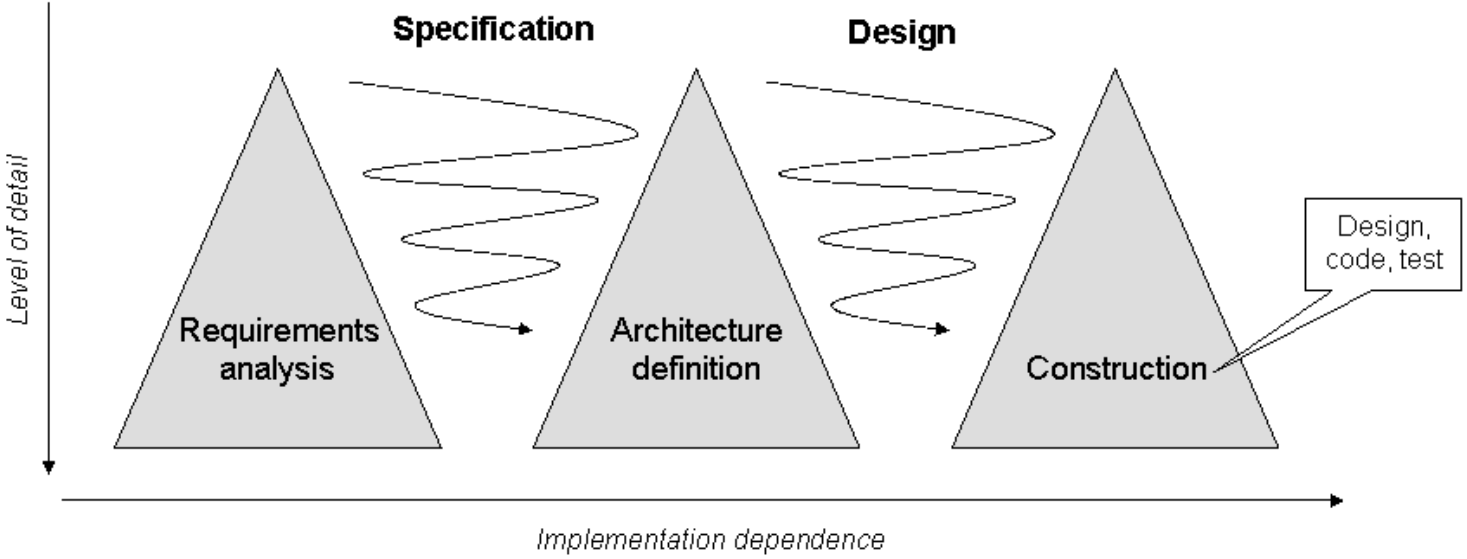


❶ Requirements inform definition of architecture

❷ The architecture reveals costs, tradeoffs, omissions and new opportunities for the envisioned requirements

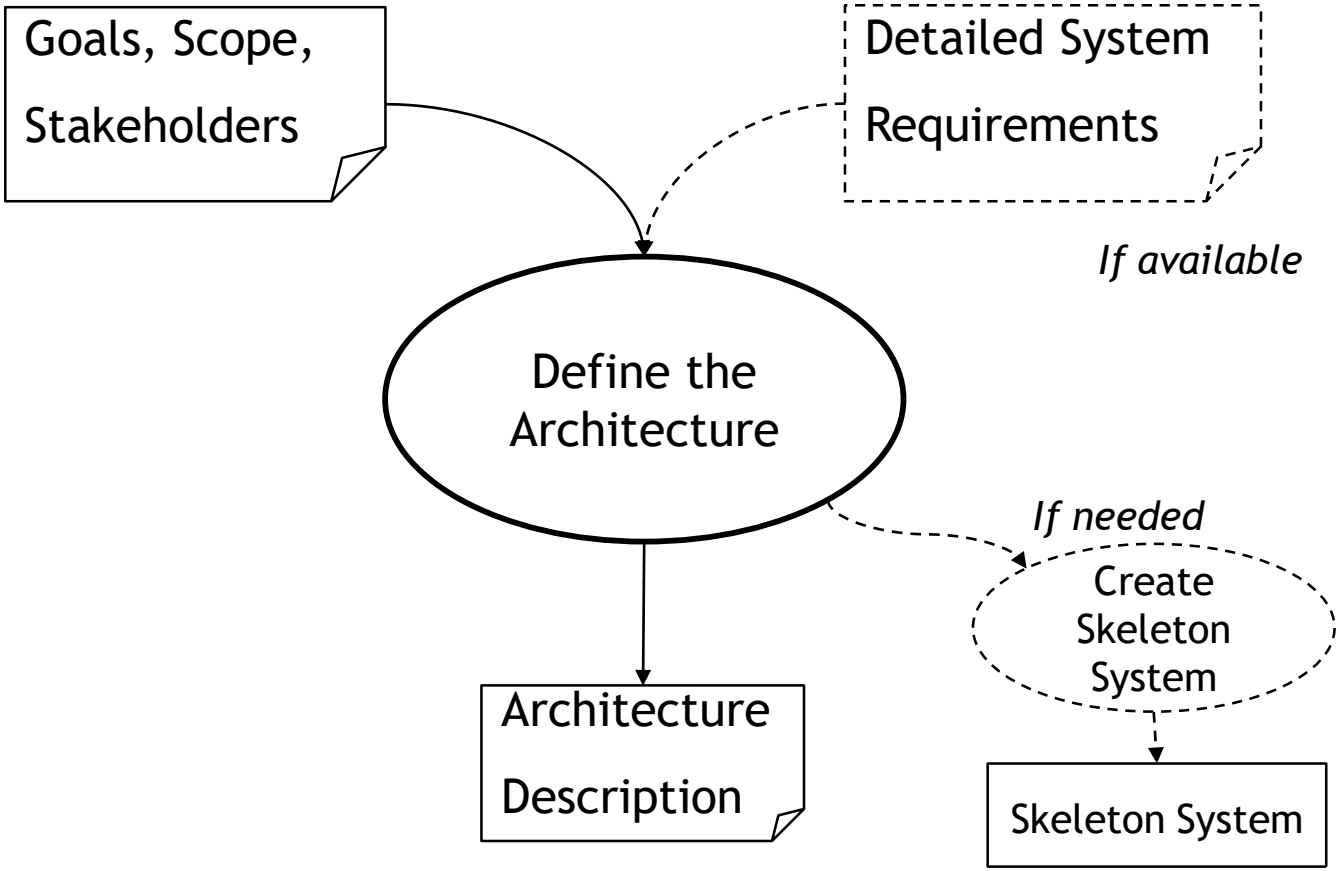
The Three Peaks Model

The elaboration of the requirements and architecture is intertwined with the software construction.

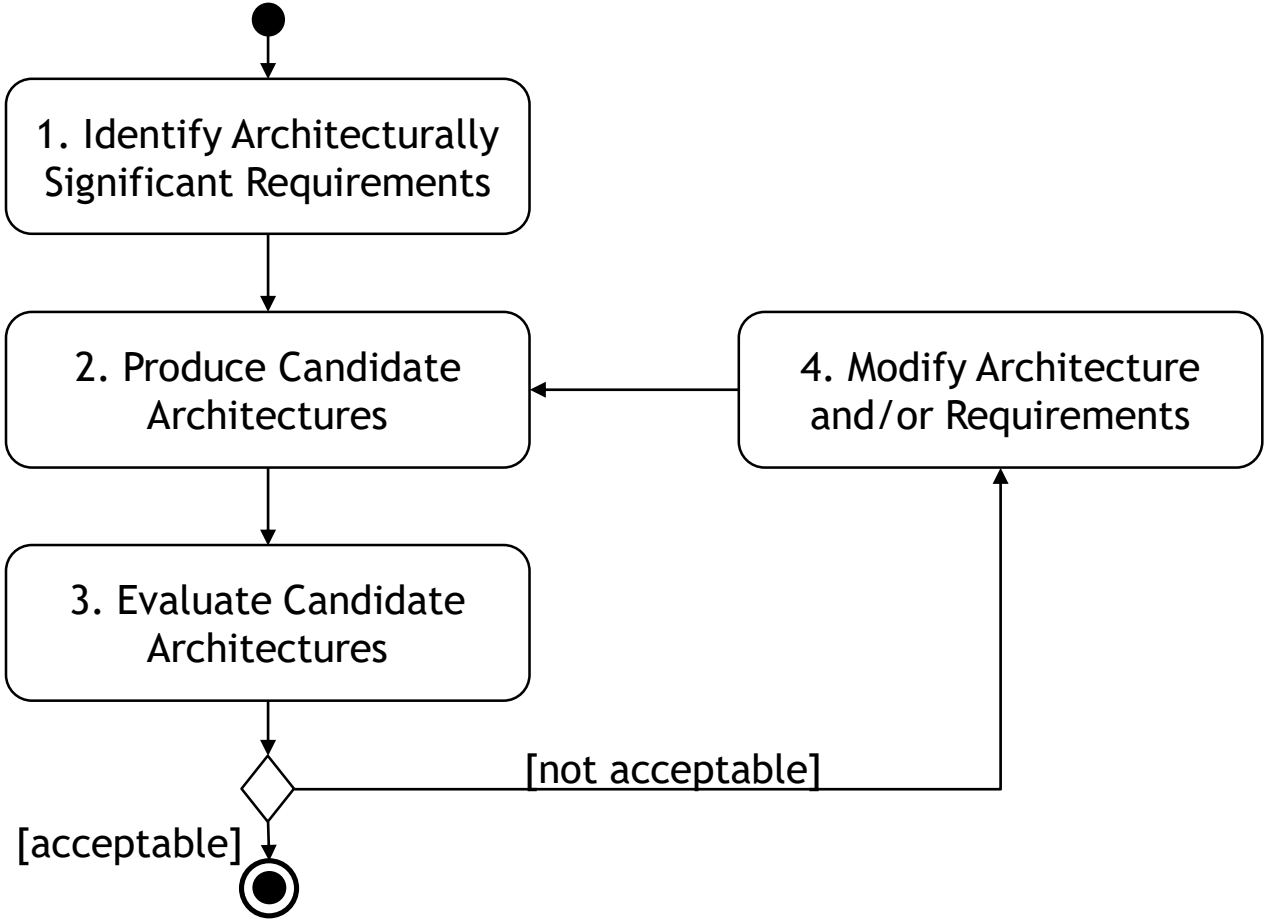


Rozanski, Nick & Woods, Eoin. Software Systems Architecture. Addison-Wesley, 2005.

Context of the Architecture Definition Process



Architecture Definition Process



1. Identify Architecturally Significant Requirements

Architecturally significant requirement (ASR): a requirement that has a significant impact on architectural decisions; the outcome of the architectural decisions would be very different if the requirement was missing or different.

Synonyms: architectural concerns, architecture characteristics.

By contraposition, a requirement is **not** architecturally significant if the presence or absence of this requirement does not affect the outcome of architectural decisions.

ASR Characteristics

ASR are those than can “break” an architecture

- A missing core functionality that requires major changes to the existing architecture
- A missing quality requirement that cannot be met with the existing architecture

Exercise

Which of these requirements for an air pollution monitoring system are architecturally significant?

1. The system shall be able to display air temperature in Celsius or Fahrenheit according to the user preference.
2. The system shall record air pollution levels at all 10,000 monitoring stations every 0.1 seconds.
3. User Story: search for monitoring stations by postcode

   As a visitor on the air pollution website
   I want to find monitoring stations close to my postcode
   So that I can view past and current pollution levels close to where I live

Heuristics: Likely ASR are those that refer to …

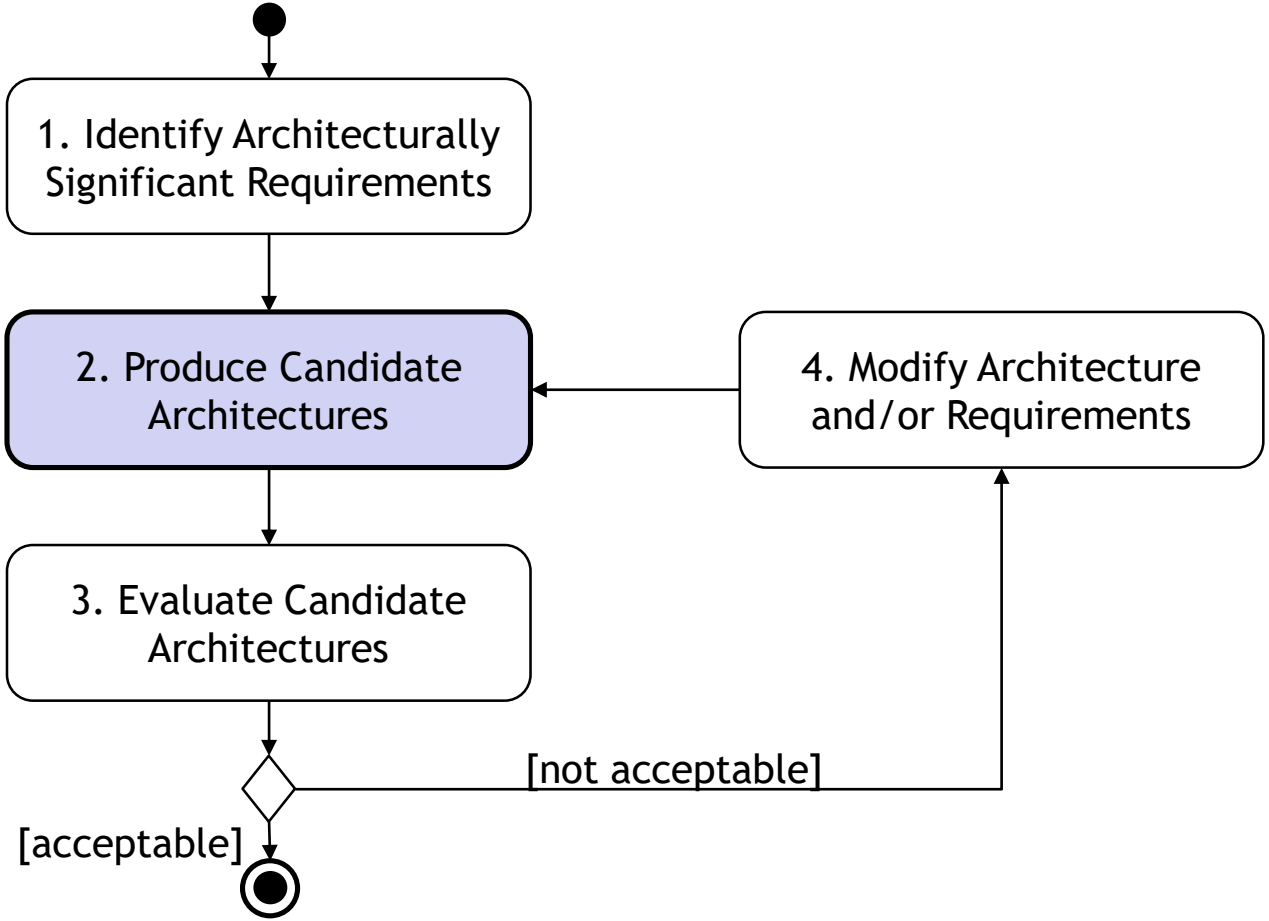
Chen et al. Characterizing Architecturally Significant Requirements.
IEEE Software, 2013

1. Core features
   - Features essential to the project's main goals
2. Quality requirements
   - Performance, availability, security, evolution, etc.
3. Constraints
   - Budget and schedule constraints
   - Legacy systems
   - Implementation and technology constraints
4. Application environment
   - Internet, corporate network, embedded hardware, virtual machines, mobile devices, etc.
   - Systems running in different environments often have vastly different architectures

Observations

- For a given system, only a small subset of requirements are architecturally significant
  - around 10 to 20 rather than 100s or 1000s
- In many requirements documents, ASR tend to be neglected, described vaguely, or hidden within other requirements

Architecture Definition Process

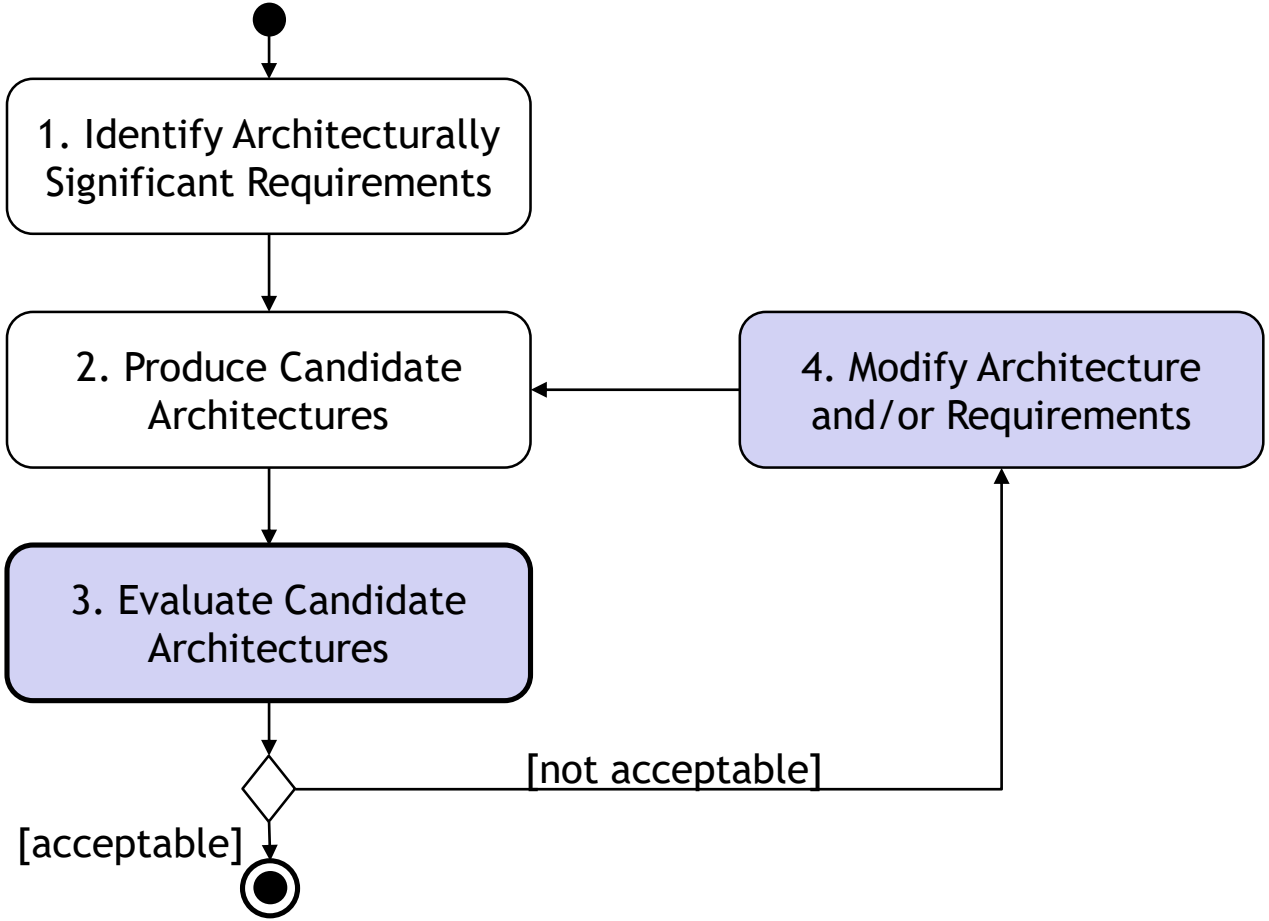


2. Produce Candidate Architectures

- Decompose the system into functional elements (a.k.a. components) with well-defined responsibilities.

- Identify architectural choices and create models for one or more candidate architectures.
  - Choose an appropriate architectural style (e.g. layered, plugin, pipeline, event-driven, service-oriented, microservices, etc.)

Architecture Definition Process



3-4. Evaluate and Rework Architecture

Architectural perspectives =

- guidelines for defining perspective-specific requirements (e.g. performance, availability, security, evolution, etc.).
- techniques for evaluating architecture against these perspective-specific requirements
- architecture tactics for modifying an architecture to satisfy the perspective-specific requirements (e.g. improve performance by using caching, multiple servers and load balancing).

Watch the following videos

Search for the lecture series in https://learning.oreilly.com/

1. What is Software Architecture? (11:30)
2. Understanding the Expectations of an Architect (11:39)
3. Thinking like an Architect (14:36)
4. Identifying Architecture Characteristics (21:05)
5. Analysing Architecture Tradeoffs (13:02)

Total: around 70 minutes

or read Chapters 1, 2, 4 and 5 of “Fundamentals of Software Architecture”, O’Reilly, 2020.

Next Weeks Topics

- Architecture models: how to describe architecture using multiple viewpoints
  - context, functional, deployment, development.
- Architecture Styles: common structure with known characteristics and tradeoffs
  - layered, plugin, event-driven, pipepline, service-oriented, microservices, etc.
- Architectural perspectives: common techniques to evaluate and improve specific qualities
  - performance, scalability, availability, security, evolution, cost.