Architecture Development Guide

Owner	Allen Baker.
Contributors	Allen Baker.
Date Created	13 April 2001
Last Modified	27 May 2014
Current Revision	1.0.2

Table of Contents

1. Introduction 3
2. What is “Architecture” 4
   2.1 The Relationship between Architecture and Design 5
3. The General Approach 6
   3.1 The Process 6
      3.1.1 When Do We Stop 7
   3.2 The Relationship between Design and Requirements 8
   3.3 The Artifacts of Design 9
      3.3.1 The Design Specification 9
      3.3.2 The Presentation Material 9
4. Design Specification Standard 10

1. Introduction

This document explains an approach to defining a software architecture. It also explains the reasoning behind the approach and defines the artifacts that the Architecture Team should produce.

2. What is “Architecture”

Many of the terms used by people in the field of Software Engineering are ill defined. However, to avoid confusion and interminable argument, it is essential to be able to efficiently and unambiguously identify concrete concepts. For this reason, we must nail down terms that designate concepts and agree to use the terms consistently. Often, it is impossible to select a term that everyone agrees with to represent a specific concept. Nevertheless, definitions must be set and we must move on, knowing that some people will disagree with certain choices.

First, we must settle on a definition of the term “architecture”. While there is no standard definition of the term “architecture”, there is certainly no shortage of proposed definitions. To be able to move forward, the team uses the Software Engineering Institute’s definition of “architecture”:

> ***The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.***

By "externally visible" properties, we are referring to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. The intent of this definition is that a software architecture must abstract away some information from the system (otherwise there is no point looking at the architecture, we are simply viewing the entire system) and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction.

Let's look at some of the implications of this definition in more detail.

First, *architecture defines components*. The architecture embodies information about how the components interact with each other. This means that architecture specifically *omits* content information about components that does not pertain to their interaction.

Second, the definition makes clear that *systems can comprise more than one structure*, and that no one structure holds the irrefutable claim to being *the* architecture. By intention, the definition does not specify what architectural components and relationships are. Is a software component

- An object?
- A process?
- An information structure?
- A library?
- A database?
- A sub-system?
- A commercial product?

It can be any of these things and more.

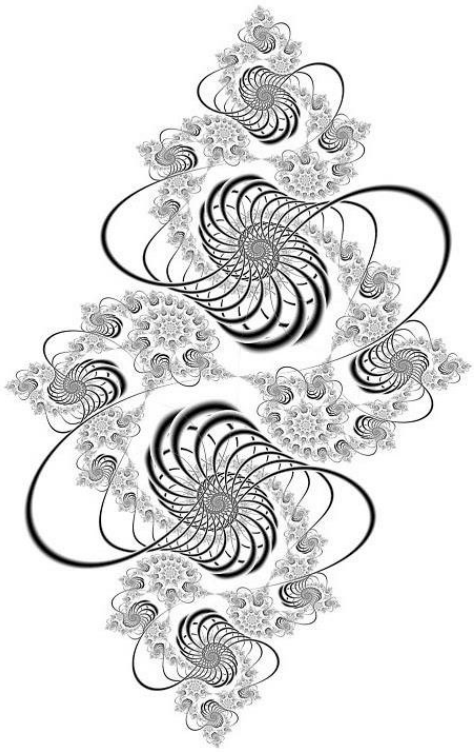
Third, the definition implies that *every software system has an architecture*, because every system can be shown to be composed of components and relations among them.

Fourth, *the behavior of each component is part of the architecture*, insofar as that behavior can be observed or discerned from the point of view of another component. This behavior is what allows components to interact with each other, which is clearly part of the architecture. Hence, most of the box-and-line drawings that are passed off as architectures are in fact not architectures at all. They are simply box-and-line drawings.

2.1 The Relationship between Architecture and Design

Clearly, by the definition above, architecture *is* design, and the process of developing an architecture is a design process. The difference between architecture and design is that "architecture" connotes a specialized type of design.

According to common usage, the term "architecture" refers to the type of design that is large, encompassing, and broad. However, this connotation is relativistic. The architecture of a system includes the overall view of its significant components, how they are organized, and how they interrelate, along both structural and behavioral dimensions. According to usage, the design of an individual component of a system is not an architecture as long as the component is seen within the context of its enveloping system. However, the design of the component *is* often considered an architecture when the component is viewed independently of the greater system, especially if the component is large and complex. Indeed, one can even describe the architecture of an elementary algorithm, as long as the point-of-view is only the algorithm and nothing else.



Regardless of the scale of the component, the design (or equivalently, the architecture) of the component addresses the issues of the component's significant sub-components, how they are organized, and how they interrelate. Additionally, the design of any component involves living within the stylistic and metaphoric constraints that the component inherits from the design of its parent component and may involve defining additional stylistic and metaphoric constraints that are passed down to the sub-components. What changes with scale is the component's level of abstraction and scope of impact.

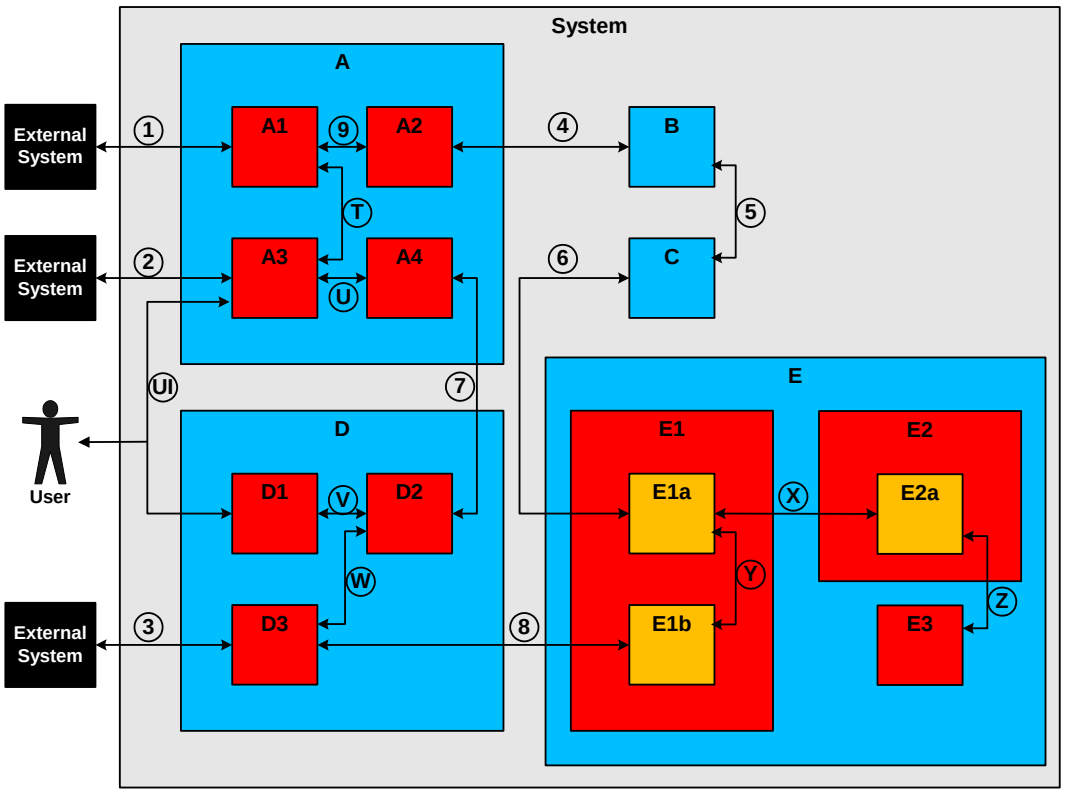
Plainly, what is viewed as architecture from one vantage point is viewed as design from a broader vantage point. This pattern of relationship between architecture and design is fractal-like in that it repeats at all scales and the choice of scale at which "design" is called "architecture" is arbitrary.

This being the case, the approach to developing an architecture is the same as the approach to developing a design.

3. The General Approach

The approach to system design is recursive and reflects the hierarchical nature of systems.

You can view the illustration below as an abstract representation of a hypothetical system. The large gray rectangle represents the whole system. Each nested rectangle represents a lower-level component. Arrows represent interfaces.



Systems are usually hierarchical structures of components like this. At the top of the hierarchy is the system as a whole comprising a small set of interconnected high-level objects. These objects may be static data structures, dynamic algorithms, or composite objects (objects that encapsulate lower level objects). The next lower level consists of the system's high-level objects, each of which may be composed of a small set of interconnected lower-level objects. From Architects who design using levels of abstraction to Programmers who decompose their code into hierarchical sets of objects, Developers use the principle of "design in order of increasing detail" at all levels of system design, regardless of the design paradigm (structured, object-oriented, etc.). [1]

3.1 The Process

The design process is recursive. A system (or component) is decomposed into several lower level, detailed components and the interactions among those components are

[1] A component does not have to be a piece of code. It may also be a data structure or a data model. In this case, the same hierarchical approach applies with higher-level abstractions being composed of lower level abstractions and concrete data.

defined. Then each of the lower-level components is put through the same decomposition and definition process. This results in the identification of even lower-level components, which in their turn are subjected to the same process, and so on down to the system's lowest level objects.

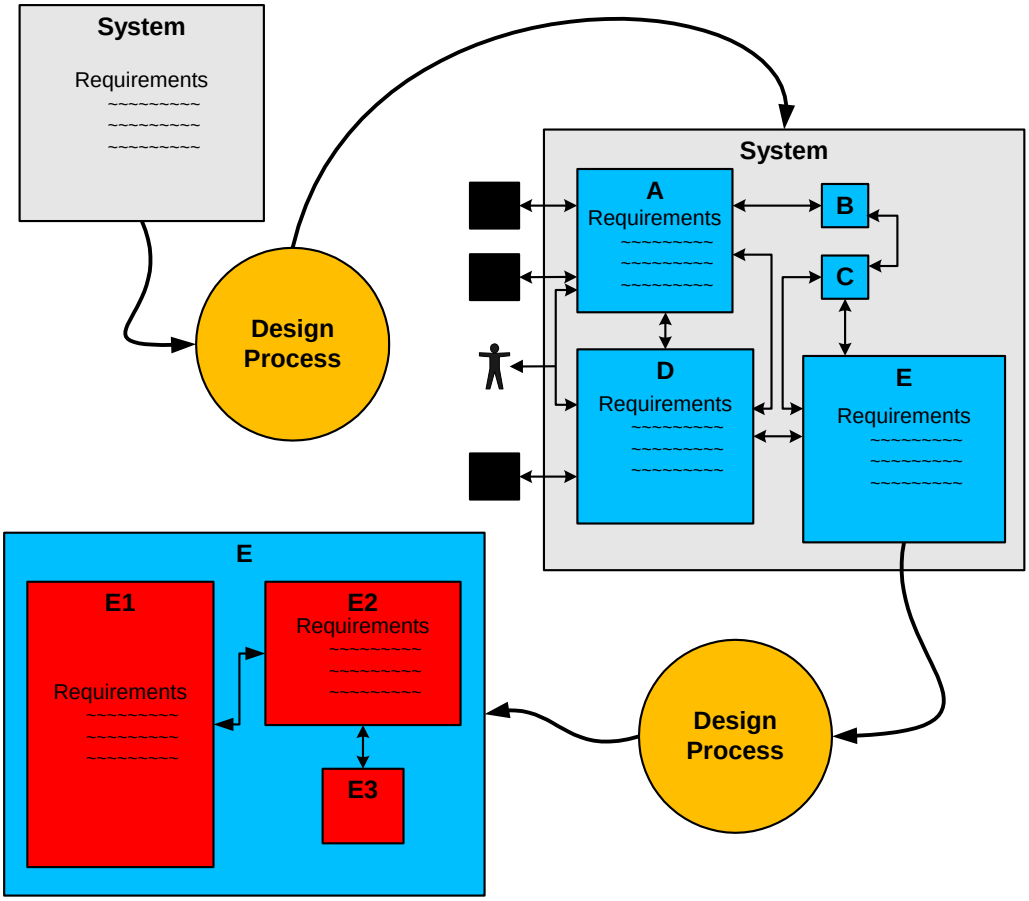
Designing each level entails describing the level's constituent components in terms of their functional purpose; meaning the roles that they play within the level. It includes explaining *why* each component exists and articulating the requirements for each. Designing each level also entails rigorously defining the interfaces among the components that comprise the level.

3.1.1 When Do We Stop

For an architecture project, there is an expectation that the team will not carry this design process to the ultimate level of detail. However, it is not easy to pinpoint the level of detail at which the architecture effort is done. The heuristic that the team should use to decide when to stop decomposing into more detail is:

the team should decompose to the level necessary to identify and describe all components that will be assigned to subordinate teams for design elaboration, and no further.

3.2 The Relationship between Design and Requirements



The process of design is tightly coupled with the process of requirements definition and their relationship to each other is not accurately reflected in the usual life cycle models. The “Waterfall” and “Spiral Waterfall” models of a sequential relationship between the two is inaccurate. The model in which design and requirements are overlapping but separate is also inaccurate.

The accurate understanding is that the life cycle is recursive. The requirements process defines the requirements for the system as a whole. The design process allocates the system requirements to high-level objects and defines the interfaces between those objects. For each high-level object, the allocated requirements are elaborated and additional requirements are derived if they are necessary to meet the allocated requirements. Another design process step allocates the high-level object’s requirements to lower-level objects. This process alternation repeats until the system is completely designed. The output of the requirements process is input to the next stage of the design process, and the output of the design process is input to the next stage of the requirements process.

3.3 The Artifacts of Design

In addition to Design Specifications, the artifacts of design include presentation material tailored to specific audiences.

3.3.1 The Design Specification

The component descriptions and interface definitions are recorded in a Design Specification (or equivalently, an Architecture Specification). Each level gets its own Design Specification. A Design Specification is a document and contains the substantive outcome of the design process for the level it represents.

For each level, the team maintains a Design Specification containing all the information about the design. It is detailed, unambiguous, and technical. The complete set of Design Specifications is a complete definition of the system.

3.3.2 The Presentation Material

As the Designers (or equivalently, the Architects) develop the design, they must regularly communicate it to two different audiences: Customers and Users. The Customers for the design are the Business people that pay for the design and to whom it represents business-enabling intellectual capital. Customers are a generally non-technical audience. The Users of the design are the Developers that will actualize a system based upon the design. Users are a technically expert audience.

Both audiences deserve their own tailored view of the evolving design. For Customers, the team maintains a single presentation that describes the evolving design at a conceptual level. At any point in time, it is an accurate view of the current state of the design that is in the Design Specification. It describes the design from the perspective of how it affects the business and with the level of technical detail suitable for that purpose. Each member of the team uses the same presentation material when describing the design to Customers. This guarantees that everyone on the team is delivering a consistent message to Customers. It relieves the team members from having to develop and maintain their own individual presentation material. Moreover, it guarantees that there is always a presentation “ready to go” and available to any team member that is asked to give a presentation on the design on short notice.

Similarly, for Users, the team maintains a single presentation that describes the evolving design at a technical level. It describes the design from the perspective of how a Developer would work within it to build a system

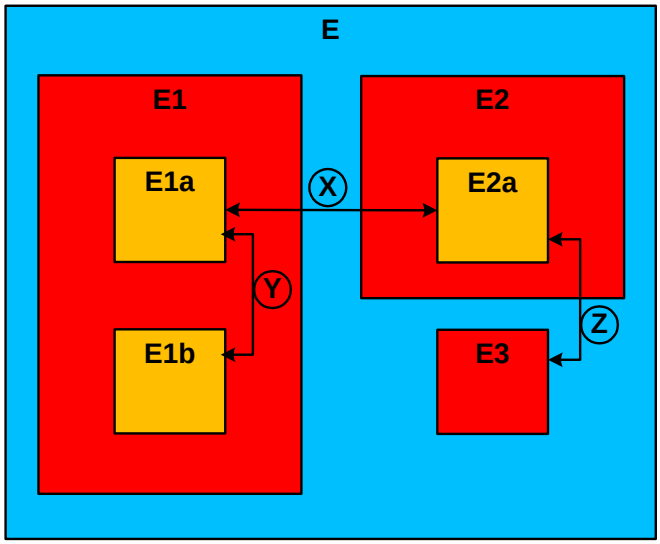
Both of these presentations should be organized in a fashion resembling the hierarchical nature of the design itself. This allows the team member to select a subset of the presentation that represents a branch of the design tree to present to an audience that is only interested in a portion of the design.

4. Design Specification Standard

As part of an architecture project, the team should create a Design Specification Standard that is a template and set of rules for a Design Specification.

The Design Specification Standard should support and encourage the recursive design process. The resulting Design Specification should describe every component at every level in the system hierarchy the same way. The component is first described from an external perspective (the black box view). As the component is decomposed, the resulting decomposition is also recorded in the Design Specification. The decomposition is a description of the component from the internal perspective (the white box view).

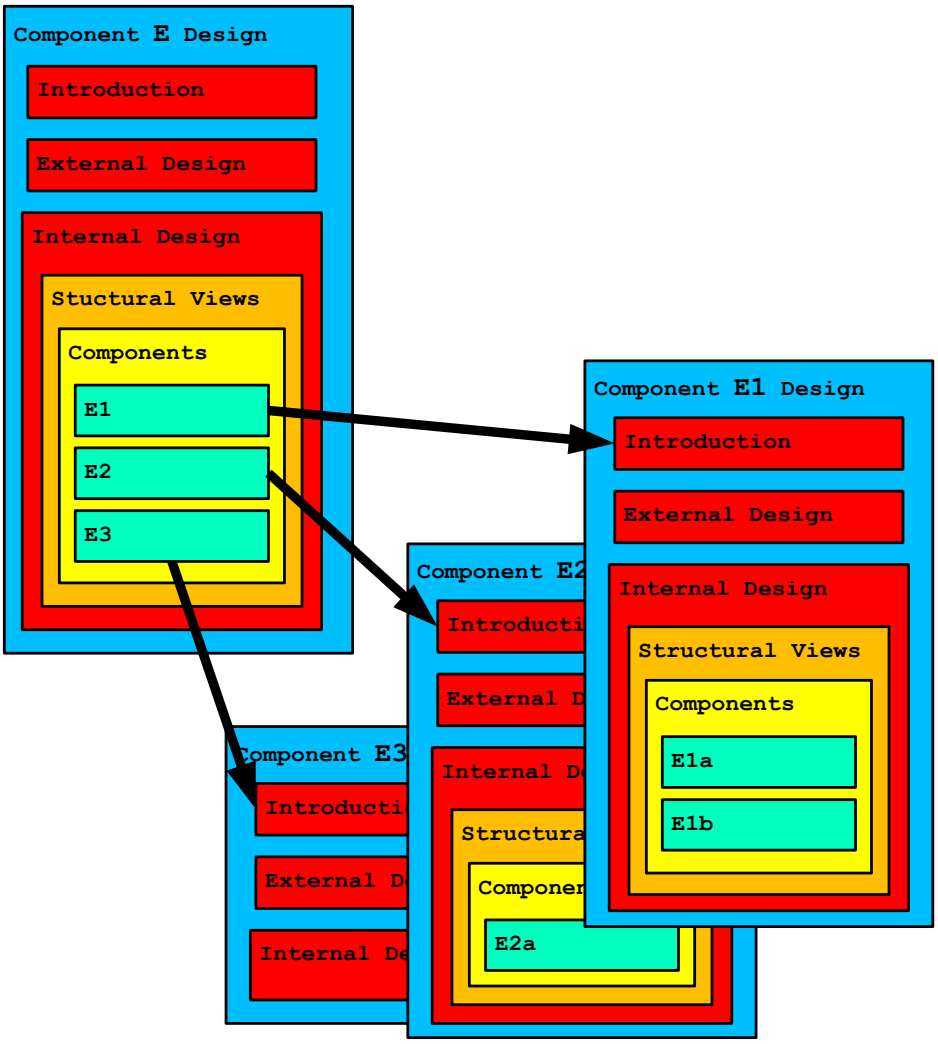
Here is how the standard would support the recursive design process. Focusing on the blue component (E) in the illustration below: E's external design defines its externally discernable characteristics and is recorded in the External Design section of the E's Design Specification. E's internal design defines its inner workings and structure and is recorded in the Internal Design section of E's Design Specification.



Components: E's internal design identifies E's three sub-components (E1, E2 and E3) and the interfaces among them (just X and Z in this case). It describes E1, E2 and E3 in terms of their functional purpose -- the roles that they play within E. It explains **why** E1, E2 and E3 exist and it articulates the requirements that E has for each.

Interfaces: In addition, E's internal design rigorously defines the interfaces that come into view when E's internal workings are exposed (only interfaces X and Z because interface Y doesn't come into view until E1's internal workings are exposed).

Now for the recursive part: A new Design Specification is created for each of E's sub-components (E1, E2 and E3). Just as was done with E, E1, E2 and E3 are defined by their external and internal designs. Taking E1 as an example: E1's external design defines its externally discernable characteristics. E1's internal design defines its inner workings and structure and identifies the two sub-sub-components (E1a and E1b) along with the interfaces between them. It rigorously defines the interfaces that come into view in E1 (only interface Y in the diagram above because X and Z were defined in E's internal design). **…**



As illustrated in the above diagram, the Introduction section of the Design Specification ties the design of one component back to the higher-level design that first identified the component. In fact, you can derive much of the content of a sub-component's Introduction section from the higher-level component's description of that sub-component's functional purpose and role.

This approach to specification can result in a swarm of Design Specifications if you do not apply common sense to the process. It is not desirable to generate separate Design Specification documents for every trivial component in the system design hierarchy. Where it makes sense, you can recursively absorb sub-component Design Specifications into the Components section of the higher-level component's Design Specification like this:

Component E Design
- Introduction
- External Design
- Internal design
  - Structural Design
    - Components
      - **Component E1 Design**
        - Introduction
        - External Design
        - Internal design
          - Structural Views
            - Components
              - **Component E1a Design**
                - :
                - :
              - **Component E1b Design**
                - :
                - :
      - **Component E2 Design**
        - Introduction
        - External Design
        - Internal design
          - Structural Views
            - Components
              - **Component E2a Design**
                - :
                - :
      - **Component E3 Design**
        - Introduction
        - External Design
        - Internal design
          - Structural Views
            - :
            - :