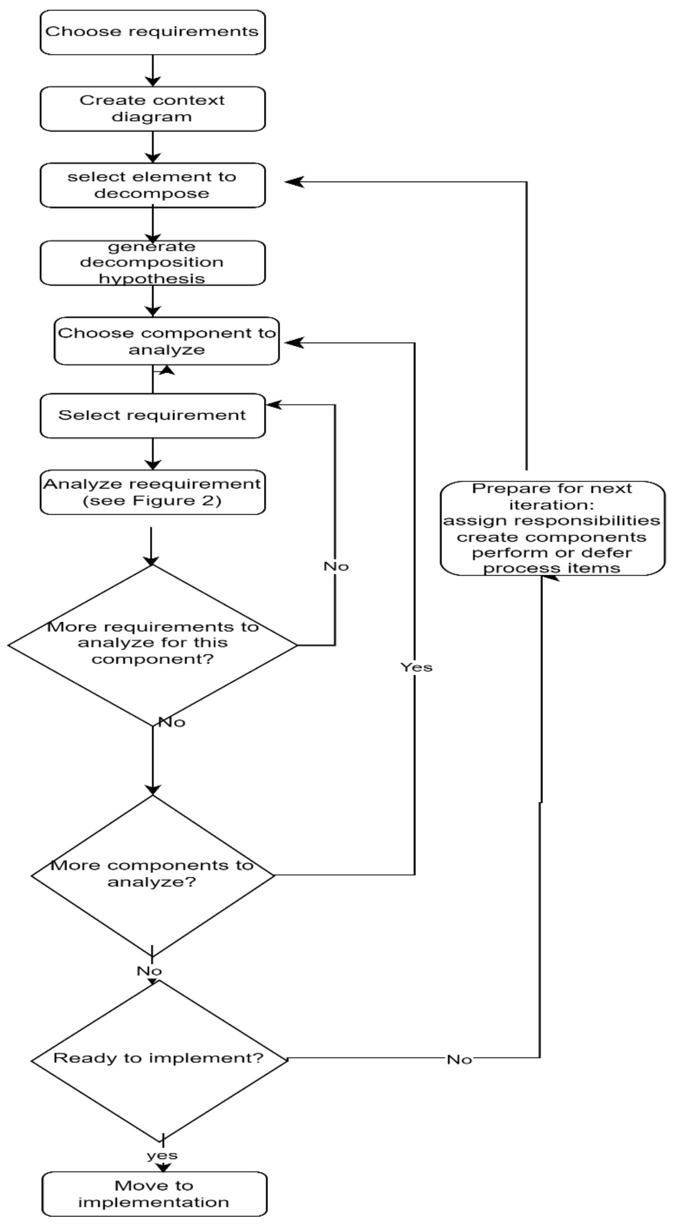
Software Architecture Design Method

The Software Architecture Design Method (SADM) is a systematic method for generating an architecture from a set of requirements. It is an iterative method for decomposing a system or a component. The method begins with a hypothesis for a decomposition and analyzes that hypothesis based on the enumerated requirements. The analysis results in modifications to the hypothesis (the iteration portion). Once a decomposition exists that satisfies the requirements, components from that hypothesis are further decomposed (the decomposition portion). The method halts when the current hypothesis is ready for implementation. Figure 1 gives the overall flow of the method. The inner loop is the iteration loop and the outer loop is the decomposition loop.

Figure 1: Overall flow of SADM


Choose requirements
Create context diagram
select element to decompose
generate decomposition hypothesis
Choose component to analyze
Select requirement
Analyze reequirement (see Figure 2)
More requirements to analyze for this component?
No
No
More components to analyze?
Yes
No
Ready to implement?
No
yes
Move to implementation
Prepare for next iteration: assign responsibilities create components perform or defer process items

What is shown in Figure 1 is essentially: “analyze all pairs of components and requirements” The flow is a method of sequencing through the pairs. An important portion of the method is the analysis of a requirement with respect to the current hypothesis. This is not shown in Figure 1 and is shown in Figure 2.

As the method proceeds, decisions will either be made or deferred. These decisions (or deferred decisions) should be recorded.

Recording the design. The design, as it is created, will have both a component view and an allocation view. Decisions about the design can be recorded either with a graphical representation or a tabular one as in Table 1. since the design will be modified as the method proceeds, a table can be easier to modify than a figure. On the other hand, a figure can be easier to understand.

Table 1: Current state of design

Component	**Responsibilities of the component**	**Quality attribute requirements on the component.**	**Components or external entities with which it communicates**	**Allocated to**

Deferring decisions. Some decisions cannot be immediately recorded in the design. These decisions are deferred until later steps. Two additional tables are used for deferred decisions.

An unassigned responsibility table contains the responsibilities discovered during the current iteration that are not immediately assigned to a component. For example, “the asset periodically publishes a status message”. The second table is a process table. Entries in both tables should be annotated with the identification of the requirement that caused the entry. This will help record the rationale for the entry.

Three types of activities are placed in the process table.

1. Deferred decisions. Some decisions may be deferred until further information becomes available from the analysis. For example, “use a hosted cloud solution (PaaS) or design your own solution” is a decision that may not be made immediately until one or more iterations have been completed.
2. Research activities. Some activities may involve performing research on documentation or using the internet. For example, “identify connected devices” involves understanding operating system services that may not be known at this point by the designer.
3. Testing activities. Some decisions may need to be determined through performing some type of tests. For example, “build a prototype to determine the performance of a component”.

The following sections describe SADM steps in more detail.

Choose Requirements

SADM assumes a set of use cases and quality requirements. Constraints can also be considered requirements if they affect the architecture. Each requirement should be identified so that it can be cross referenced during the analysis step. The amount of time taken to generate an architecture is proportional to the number of requirements. Each requirement leads to an analysis and to a response to that analysis.

Choosing which requirements drive the design is outside of the scope of SADM. A general rule is that those requirements that have the most impact on the design are the ones that should be used. Chapter 19 of the 4th edition of Software Architecture in Practice provides guidance in choosing the requirements to be used as input to SADM.

Create Context Diagram

A context diagram shows what is in the system (or component) to be decomposed as well as what external actors interact with the system or component. Managing the interaction with external actors will lead to responsibilities and communications that should be considered during the analysis.

Select Component to Decompose

SADM is a decomposition method. This step is a portion of the decomposition loop. The first time this is executed, the component will, typically, be the whole system. The method decomposes the system into components. On subsequent times through the loop, there will be several components to be decomposed. One of them at a time should be chosen.

Table 1 shows the input to this step. It captures what is currently known about the design and its components. As the iterations proceed, more information is added to the design.

Generate Decomposition Hypothesis

The prior step selected a component to decompose. This step will generate a hypothesis for decomposing the selected component. The hypothesis will be the subject of the analysis in the inner loop.

Several possibilities exist for generating the decomposition hypothesis.

- Use a reference architecture. Reference architectures are available for many domains or subdomains. A search for “reference architecture for [domain or subdomain]” or “software architecture for [domain or subdomain]” should generate possibilities.
- Use a documented architecture pattern. Each pattern includes a list of advantages and disadvantages of using that pattern. Architecture patterns can be organized based on the quality attributes that they support. Part II of Software Architecture in Practice provides some patterns organized in this fashion. The prioritized quality attribute requirements are used as guidance in choosing an appropriate pattern to begin.
- Use an architecture for a similar system in the domain. This may be based on experience or on blogs or other information available on the internet.

- If a modification of an existing system is being designed, use the architecture of the existing system.

The hypothesis should be entered into the current state of the design. Some design decisions may be constraints on the design such as “use the cloud” or “Use these pre-existing components”. These constraints should be in the table.

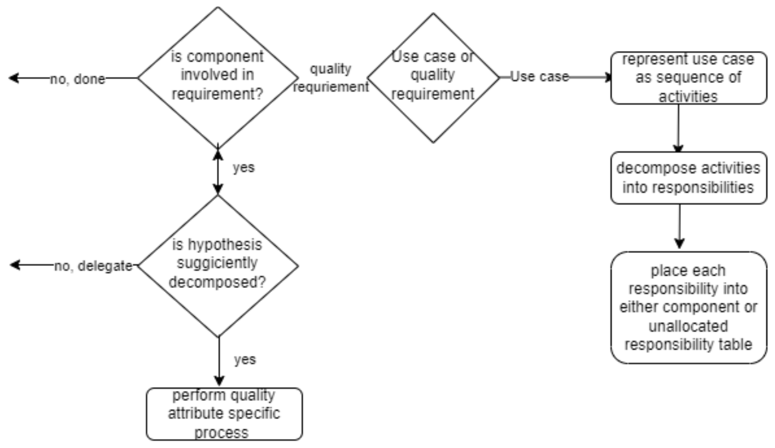
Select Requirement

There are a set of use cases and quality requirements that were identified in the first step of SADM. Each requirement in this loop will be analyzed using the decomposition hypothesis. This is the inner loop of Figure 1. Select one of these requirements as the next one to analyze.

Analyze Requirement

SADM distinguishes between use cases and quality attribute requirements. Figure 2 shows the two paths.

Figure 2: Analyzing requirements


no, done
is component involved in requirement?
quality requriement
Use case or quality requirement
Use case
represent use case as sequence of activities
decompose activities into responsibilities
place each responsibility into either component or unallocated responsibility table
yes
no, delegate
is hypothesis suggiciently decomposed?
yes
perform quality attribute specific process

Analyze use case

Use cases are satisfied by some sequence of activities. Activities are made up of a sequence of responsibilities. Some of these responsibilities can be immediately assigned to components in the hypothesis. Others may not have a natural home in the current hypothesis. Those that are assigned to a component should be recorded in the design description. Those that do not have a current natural home are recorded in the unassigned responsibilities or the process table.

The sequence of activities will help define the communication needs of a component in the hypothesis. Responsibilities to manage communication are added to components in the hypothesis. Allocation decisions can also be made in response to a use case or a constraint. In some cases, two components should reside on the same server or in the same pod. These decisions should be recorded in the design.

The analysis of a use case has the following steps:

- Express the use case as a series of activities. For example, a use case that describes a user performing an action would have the activities of user inputs command, system distributes command to appropriate component.
- Decompose activities into a series of responsibilities. Following the same example, UI receives command, marshals input, sends message to web server, web server unmarshals message, decides on recipient component, passes message on, recipient component processes message.
- Each responsibility is either allocated to an existing component or placed in the unassigned responsibility table. In our example, assuming in our example that the hypothesis has a UI and a web server, but recipient components have yet to be identified, the responsibilities up to sending the message are assigned to the UI, the responsibilities of unmarshalling, and deciding on recipient are assigned to the web server. Identify recipient components of decision is placed in the process table and "process message" is placed in the unassigned responsibility table.

Analyze quality requirement

The analysis of quality requirements will depend on the quality being required, the component being analyzed, and the state of the design. In every case, first consider these two possibilities:

- The decomposed component is not involved in the achievement of the quality attribute requirement. Iin this case, there is no analysis to be performed.
- The hypothesis is not sufficiently decomposed to perform an analysis in which case, the requirement is deferred to the next iteration. For example, an availability requirement on the system may be delegated to become a requirement on one of the components in the hypothesis. A latency requirement on the system may be delegated to several of the components in the hypothesis. The delegation is recorded in the design as a quality requirement for the component to which the requirement is delegated.

Given that the component is involved in the achievement of the quality requirement and the design is sufficiently decomposed to proceed, the process depends on the quality attribute. Each quality attribute has several "key questions" to help understand the limits of the current design.

We describe how the analysis proceeds for performance, availability, security, and modifiability. In general, use the tactics enumerated in Part II of Software Architecture in Practice to achieve the requirement.

- *Performance*. A performance requirement will, typically, be expressed in terms of throughput or latency in conjunction with a distribution of requests. There are three cases to consider.
    - The distribution of requests is unknown. In this case, add "determine load distribution" to process table.
    - It is unknown how many requests a particular component in the hypothesis can satisfy. In this case, add "build prototype of component to determine latency under load" to the process table.
    - A single instance of the component is inadequate to manage the requests. In this case, add "load balancer for [component] to the design and "determine autoscaling rules for [component]" to the process table.

  The key questions for performance are:

    - Is there sufficient computational capacity?
    - Is there sufficient network capacity?

  Allocation decisions may be made during the performance analysis. Allocation decisions should be recorded in the design. Allocation decisions will be based on the message transfer time between components. Messages transferred between components on the same VM or pod have transfer time measured in nanoseconds. Messages transferred over the internet have transfer times measured in milliseconds.

- *Availability*. Does the failure of a component in the hypothesis result in a violation of the requirement? If the answer is "no", nothing more is done. If the answer is "yes" and mitigation is already built into the design, then nothing more is done. If mitigation has not yet been built into the design, there are multiple options: hot spare, cold spare, multiple instances managed by a load balancer, degraded function from a different component. A process step is "choose availability option for [component]". Each choice has a detection responsibility, a transfer of control responsibility, and a recovery responsibility. These responsibilities should either be added to existing components or placed in the unassigned responsibility table.

  Redundancy may be accomplished by allocating additional servers. Allocation decisions should be recorded in the design.

  Key questions for availability are:

    - Failure of which components will result in a violation of the requirement?
    - Is there redundancy in the design for those components?

- *Security*. Are any of the components in the hypothesis a portion of the attack surface of the system? The attack surface includes components that manage credentials and network communication. If not, nothing more is to be done. If yes, use tactics from Chapter 11 of Software Architecture in Practice to mitigate against potential attacks.

  Key questions for security are:

    - Are credentials managed appropriately?
    - Is network communication secure?

- *Modifiability*. Are the responsibilities in the components of the hypothesis coherent? If not, then the component should be broken into portions where each portion has a set of coherent responsibilities. Add the new components and their responsibilities to the current design.

  Key questions for modifiability are:

    - What are the expected modifications to the system?
    - Do the expected modifications involve multiple components?

More Requirements to Analyze?

If all the requirements have not been analyzed for this hypothesis, move on to analyze another requirement if there is one. Otherwise, either implement the component or move to the next component.

Ready for Implementation?

If the design of the component is sufficiently detailed so that it can be handed over to an implementation team, then do so. Otherwise, prepare for the next iteration.

Prepare for Next Iteration

At this point in the method, the hypothesis has been analyzed for all enumerated requirements. Additional responsibilities may have been added, some to an existing component, some currently unassigned. Quality requirements may have been satisfied, may have added responsibilities, may have been decomposed into quality requirements for multiple components or may have been delegated to a component in the current hypothesis. Finally, there is a list of process steps to be performed.

Responsibilities in the unassigned responsibilities table should be attached to a component. This may be an existing component in the current hypothesis, or it may be a newly created component. Newly created components should be added to the current state of the design. The other aspects of the design (allocation, communication, and quality attributes) can be filled in here, if known, or can be filled in when the component is decomposed and analyzed during the next iteration.

Items in the process list can be performed at this time or can be deferred to be performed later.