# Teaching Software Architecture Design – Building Intuition

Gaurav Agerwala
gagerwal@andrew.cmu.edu
gagerwal@alumni.cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Len Bass
lb3a@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

## ABSTRACT

Design is a fundamental activity in the creation of software systems. It can be a daunting task since design decisions have long term implications to the system being built. These decisions are especially daunting for students who have limited experience and knowledge in building software systems. In this paper, we propose a new design method and course structure that aims to demystify the process of software design and helps students learn to think analytically while building intuition about the trade-offs being made during the design phase.

## CCS CONCEPTS

• **Software and its engineering** → *Software design engineering*; • **Social and professional topics** → **Adult education**; **Software engineering education**.

## KEYWORDS

Software Architecture, Software engineering education, Software Design method

**ACM Reference Format:**
Gaurav Agerwala and Len Bass. 2024. Teaching Software Architecture Design – Building Intuition. In *2024 International Workshop on Designing Software (Designing '24), April 15–14, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3643660.3643943

## 1 INTRODUCTION

We propose a new lightweight software architecture design method which enabes the users of the method to focus on the design decisions being taken, and their trade-offs. We used the design method as a basis to teach an architecture design course to graduate students. The complexity and verbosity of current design methods hinder students from quickly gaining practical design experience. Our proposed method enables students to focus on designing and analyzing potential solutions, rather than getting lost in the details of a specific design methodology.

Existing design methods such as Architecture Centric Design Method(ACDM)[4] and Attribute Driven Design(ADD)[8] are detailed and verbose requiring books to describe. The proposed method enables us to focus student efforts on practicing, designing, and analyzing, instead of spending time learning the details of a design method.

Designing a system essentially involves making a series of design decisions, where we create a hypothesis that is tested in each iteration of the process (generate and test)[2]. The goal of the test phase is to analyze the feasibility of the design. As students test a hypothesis and discover that it fails, it improves their intuition about the feasibility of the design. Defeasible argumentation[7] helps to discover the reasons for the failure of a hypothesis while analyzing it. This is done primarily by using critical questions to find flaws in the hypothesised design.

The next section will describe the structure of the course conducted, followed by a brief overview of the design method we are calling SADM (Software Architecture Design Method)[3]. We will elaborate on how we ensure that students are learning to analyze a design and develop intuition. Finally, we will conclude with our experience teaching architecture design to students, and recommendations for enhancing design education.

## 2 COURSE CONTEXT

A 7 week course titled 'Advanced Architecture Design' was taught to terminal Master's students at Carnegie Mellon University. A software architecture course was a prerequisite to this course. The prerequisite taught students how to identify and categorize architectural drivers, styles, views, and design primitives. It also taught students how to document and evaluate architectures. Most students taking the course had also taken courses in requirements analysis, quality verification, and project management. Students had a minimum of 2 years of experience as software engineers.

The goal of the course was to enable the students to propose a design hypothesis and then analyze that hypothesis to determine whether it satisfies the use cases, quality requirements, and constraints for the system under consideration.

The course was structured as a flipped classroom[6], where the class met two times every week. On the first class-session of each week, we discussed an important quality attribute along with the common tactics used to satisfy the quality attribute. For the second class session, we provided students with use cases, quality attributes, and constraints of a system they will be responsible for designing and analyzing. This enabled us to discuss the domain of the system in depth, and the architectural considerations we would need to make for such a system. Domains included finance, automotive, software platform (e.g. Highly Distributed File System), telephony, and manufacturing. The students were then asked to design an architecture for the specified system, and provide a reasoning for each of their choices. We also conducted quizzes in the first class-session of the week to encourage students to read the material carefully before class.


*Designing '24, April 15–14, 2024, Lisbon, Portugal*

ACM ISBN 979-8-4007-0563-2/24/04
https://doi.org/10.1145/3643660.3643943

## 3 METHOD OVERVIEW

There has been significant effort in the community to create processes that aid in designing software systems. Methods like the Architecture Center Design Method (ACDM) cover the entire software life-cycle and include steps that are aimed at determining architectural drivers, evaluating design, documenting and testing the system. SADM focuses on the core activities of identifying and analyzing a high-level design based on given requirements.

We took inspiration from ADD. Aspects of ADD included in SADM such as its focus on the design phase, achieving the satisfaction of quality attribute requirements and its iterative nature.

Most existing design methods have clear, specific, and verbosely documented steps that one should follow. These methods are useful when one is creating a system that is safety critical or a system that has to be checked and approved by a group of people. However, most systems do not need such explicitly defined design methods. SADM is a tool to help students build intuition. To achieve this, SADM has been kept intentionally minimal, encouraging students to practice designing systems, instead of learning the steps in the method. To ensure that students learn to think analytically about this and produce an acceptable design, our method treats analysis as an essential step of the design process.

A common problem in designing new systems is that there are many unknowns. Resolving these unknowns can take the form of conducting research, finding the correct design primitive, performing experiments, or even making a future decision before making the current one. SADM ensures that the designer has a mechanism to deal with these unknowns by documenting them and deciding the activity that will be performed to resolve the unknown. These are called process steps in the method and enable the designer to consciously push a decision to the future when the designer hopefully has more information. SADM is an iterative method for decomposing a system or a component. A detailed description of the method and other course materials can be found here[3]. 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. As the method proceeds, decisions must either be made or assigned to a process step. Process steps can include different types of activities as described below:

(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 the 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".

## 4 ANALYZING A DESIGN

Analyzing a hypothesis against a set of requirements and constraints lead to the following desirable results:

- Cultivating an engineering approach to making design decisions
- Providing confidence to the designer that their design will satisfice the requirements
- Encouraging discussions related to the trade-offs of selecting one design option over another

When we analyze a hypothesis, we want to make sure that our analysis is effective and not easy to challenge. A concept from argumentation theory called defeasible argumentation [7] states that arguments can be accepted for now, based on the evidence we have, but they can be changed later if new evidence appears. An analysis is defeasible when it is reasonable but not logically certain. This is similar to how architecture design decisions work, because they may have to be revised if the designer discovers something in the future that contradicts the assumptions or claims made when analyzing a proposed design. Technical explanations offered by engineers at renowned tech companies in public blog posts and articles, are cases of defeasible arguments. To build confidence in an argument students are taught to ask questions that have the potential of contradicting the argument. These questions, called critical questions give students confidence in the feasibility of the proposed design, as they aim to gather all the evidence that might weaken the argument.

Argumentation schemes are commonly used structures in argumentation theory that represent typical forms of argument. If an argument fits the structure of a certain scheme, and its premises are reasonable, then the conclusion should be accepted unless a critical question can be raised that shows a flaw in the argument. Associated with each scheme are critical questions that can be asked to potentially invalidate one of the premises. If the designer cannot satisfactorily answer these critical questions, then the conclusion is considered unsupported. Architects tend to focus on why a design works, over why it might fail. Employing critical questions correctly can help detect the drawbacks of a proposed design, enabling students to identify weaknesses in a design that may have been otherwise overlooked. Every identified weakness in the design, will build intuition about the techniques used to generate the hypothesis. Using argumentation schemes to analyze a proposed design is beneficial since it forces students to think critically about the proposed design.

Architectural arguments can take 2 main forms depending on the requirement. Satisfaction of a use case is typically done using a scenario-based argument, while the satisfaction of quality attributes is typically done using a tactic-based argument.

### 4.1 Analyzing Use Cases

Satisfaction of use cases is argued by defining the steps that would be performed by the components of the system described as a

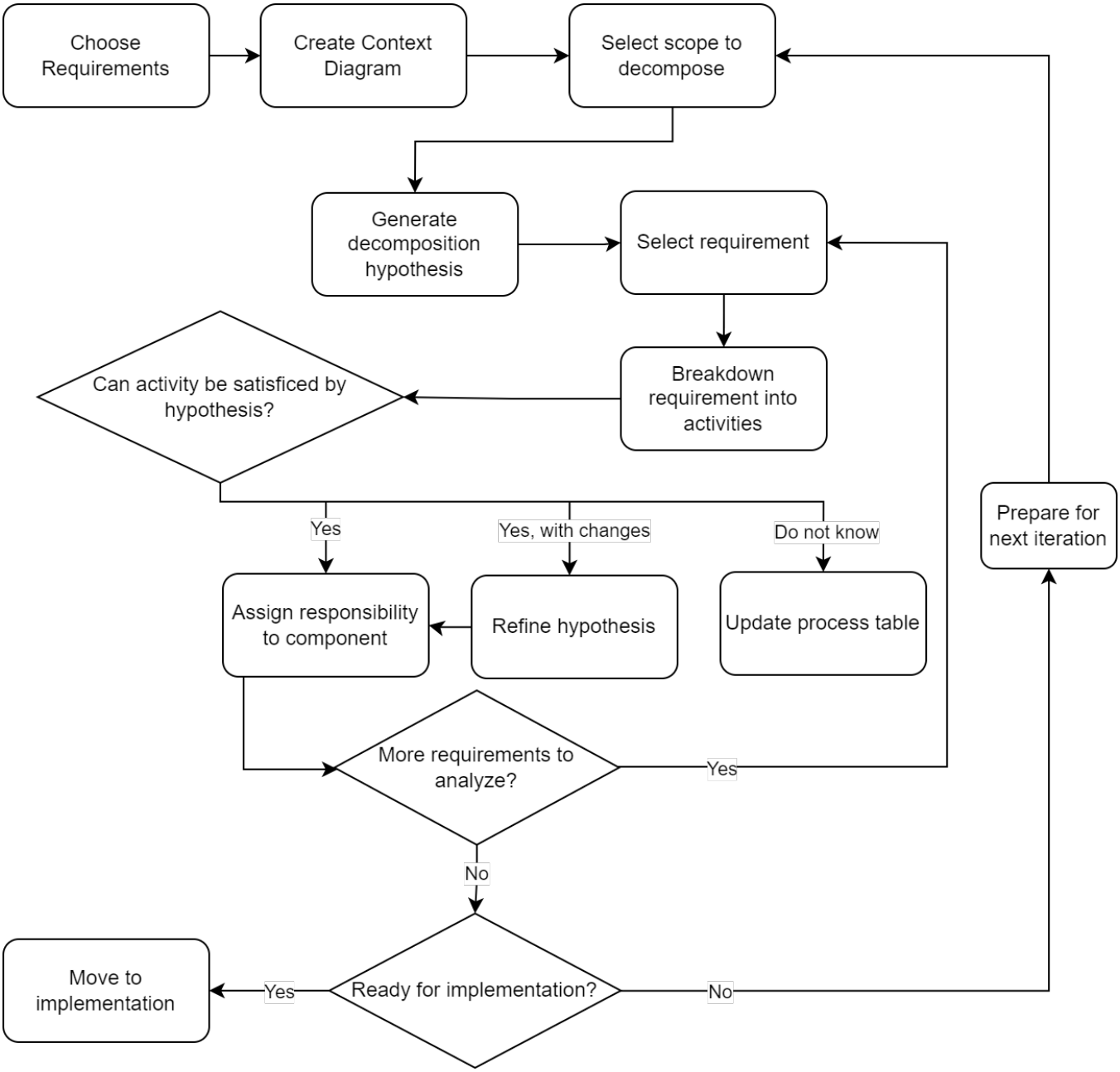


**Figure 1: Overall flow SADM**

scenario. If the scenario is a possible behavior of the hypothesised architecture and the scenario contributes to satisfying a use case, students can conclude that the hypothesised architecture contributes positively to the solution. Students are encouraged to think about the steps in the scenario and answer critical questions about them. For example:

- What are the exceptions to the steps in the scenario that invalidate its contributions to the use case?
- Are there any side effects of the proposed architecture that contribute negatively to, or prevent the steps from occurring?
- Does the proposed architecture contribute negatively to other requirements of the system (use cases or quality attribute requirements)?
- Are any constraints violated during the execution of the scenario, or its exceptions?

### 4.2 Analyzing Quality Attributes

Arguing the satisfaction of quality attributes is more challenging since years of codified knowledge is used to make a compelling argument. Arguments like, "google.com uses this technique so it must be scalable" are frequently used by inexperienced designers. To ensure that the decisions being made follow sound engineering judgement and reasoning, we use artifacts like reference architectures and architectural patterns to argue the satisfaction of quality attributes. Sources such as Microsoft's architecture center[1] make reference architectures readily available for many domains and application types. Part II of Software Architecture in Practice[5] provides some patterns based on the quality attribute they help support. Using artifacts like these, gives the designer confidence that their proposed design, will help satisfy the given quality attribute. To make this argument the designer needs to do the following:

- Show how the proposed architecture is an instance of the documented pattern, or reference architecture.
- Argue that the pattern or reference architecture is known to satisfy the specified quality attribute.

The critical questions the students are encouraged to ask about arguments in this format include:

- Does the proposed architecture include other patterns that contribute negatively to the quality attribute?
- Does the proposed pattern/reference architecture prevent another requirement in the system?

## 5 RESULTS

Our goal of ensuring that the method was lightweight and easy to learn was met since we taught the design method in one class session, and students effectively applied it to complete course assignments. Each assignment introduced the students to the requirements and constraints of a system that they designed.

To aid the students in understanding the method, and building intuition incrementally, successive assignments required students to complete larger parts of the design process. The first assignment provided the students with a hypothesis, and analysis that argued the satisfaction of the hypothesis. The students were required to prepare for a future iteration which involved assigning responsibilities to components defined in the hypothesis. The second assignment provided the students with a hypothesis for a given set of requirements, and the students were responsible for arguing that the hypothesis was sufficient, as well as preparing for the next iteration. Subsequent assignments required students to generate the decomposition hypothesis themselves based on the given use cases, quality attribute requirements, and constraints. This incremental approach enabled students to clearly understand the steps of the design method. Their ability to use it effectively gave us confidence in their understanding of the method.

Since each assignment was based on a system in a different domain, the students were exposed to varied systems. We observed that most students from computer science backgrounds had limited knowledge of domains like manufacturing, automobiles, and telecommunications. The second class session of every week which involved a discussion about the system being built, was helpful in bridging the knowledge gap, as well as answering questions about the requirements of the system being designed. A variety of domains enabled us to prioritize different quality attributes for each assignment. For example, the highly distributed file system prioritized availability, while the manufacturing system prioritized modifiability. This gave the students exposure to a variety of tactics which helped them understand the trade-offs that can be made based on the priorities of the system under consideration. It also helped them develop intuition about the priorities of quality attributes in different domains and the techniques used to achieve different quality attribute requirements.

Process steps enabled us to circumvent technical challenges that would require experimentation and research, such as the automotive system requiring a mechanism to use the onboard cameras to detect a passing vehicle. For the sake of simplicity, such tasks could be assigned to a component of the system, and deffered to a future iteration using process steps. This enabled students to focus on the overall design without spending time on the domain specific implementation details.

Using argumentation schemes and critical questions to argue the satisfaction of a requirement enabled students to structure their arguments coherently, simplifying the grading process.

## 6 FUTURE ENHANCEMENTS

We received mixed feedback from students about designing for systems in different domains. While some students appreciated the variety in domains, other students found it challenging to learn about a new domain every week. Some students also expressed a preference for designing software platforms such as message queues, and databases. For future iterations of the course, we intend to reduce the variety in domains to enable students to focus on the trade-offs in the design.

Another improvement we will make to the course in future iterations is to include quality attributes that are important to AI systems such as explainability and cost. We will also create assignments that require students to design systems with more AI components.

## 7 CONCLUSION

Often, engineers chose the cool new thing without analyzing their own requirements and proposed solution thoroughly. Our belief is that good design results from a careful analysis of the needs of the system and the trade-offs of the design. Focusing on developing intuition for software architecture design with practical hands on experience, and critical analysis, better prepares students for the challenges in modern software development. SADM gives the students a framework, and common vocabulary to propose designs that are backed by sound engineering judgement without adding unnecessary overhead. Since the method enforces critical analysis, it also gives students confidence in the feasibility of their design choices, empowering them to propose design choices in the industry. The process table enables students to defer decisions that cannot be made without experimenting, or conducting research that is infeasible to conduct in the limited time available. The weekly assignments received positive feedback from students about the effectiveness of the method to design systems that will work as expected. The analysis techniques helped in building intuition about the trade-offs to consider, and confidence in the feasibility of the designed systems.

## REFERENCES


[1] [n. d.]. Azure Architecture Center. https://learn.microsoft.com/en-us/azure/architecture/browse/ Accessed on December 5, 2023.

[2] Len Bass. 2009. Generate and test as a software architecture design approach. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture*. 309–312. https://doi.org/10.1109/WICSA.2009.5290785

[3] Len Bass. 2023. Software Architecture Design Method. (2023). https://github.com/len-bass/architecture-design

[4] Anthony J. Lattanze. 2008. *Architecting Software Intensive Systems* (1 ed.). Auerbach Publications.

[5] Paul Clements Len Bass and Rick Kazman. 2021. *Software Architecture in Practice* (4 ed.). Addison-Wesley.

[6] Jalal Nouri. 2016. The flipped classroom: for active, effective and increased learning – especially for low achievers. *International Journal of Educational Technology in Higher Education* (2016). https://doi.org/10.1186/s41239-016-0032-z

[7] José Miguel Cañete Valdeón, Antonio Ruiz Cortés, and Miguel Toro. 2016. Defeasible Argumentation of Software Architectures. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. 115–121. https://doi.org/10.1109/WICSA.2016.48

[8] Humerto Vervantes and Rick Kazman. 2016. *Designing Software Architectures - A Practical Approach*. Addison-Wesley Professional.


Received 7 December 2023